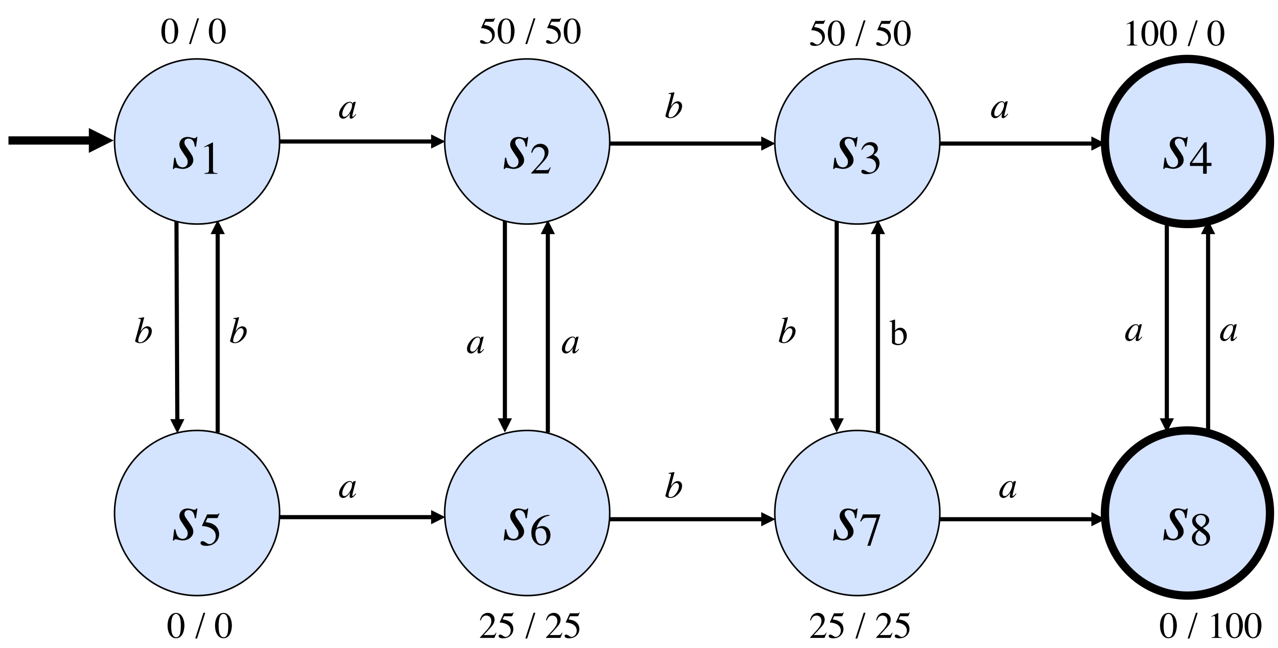
General Game Playing

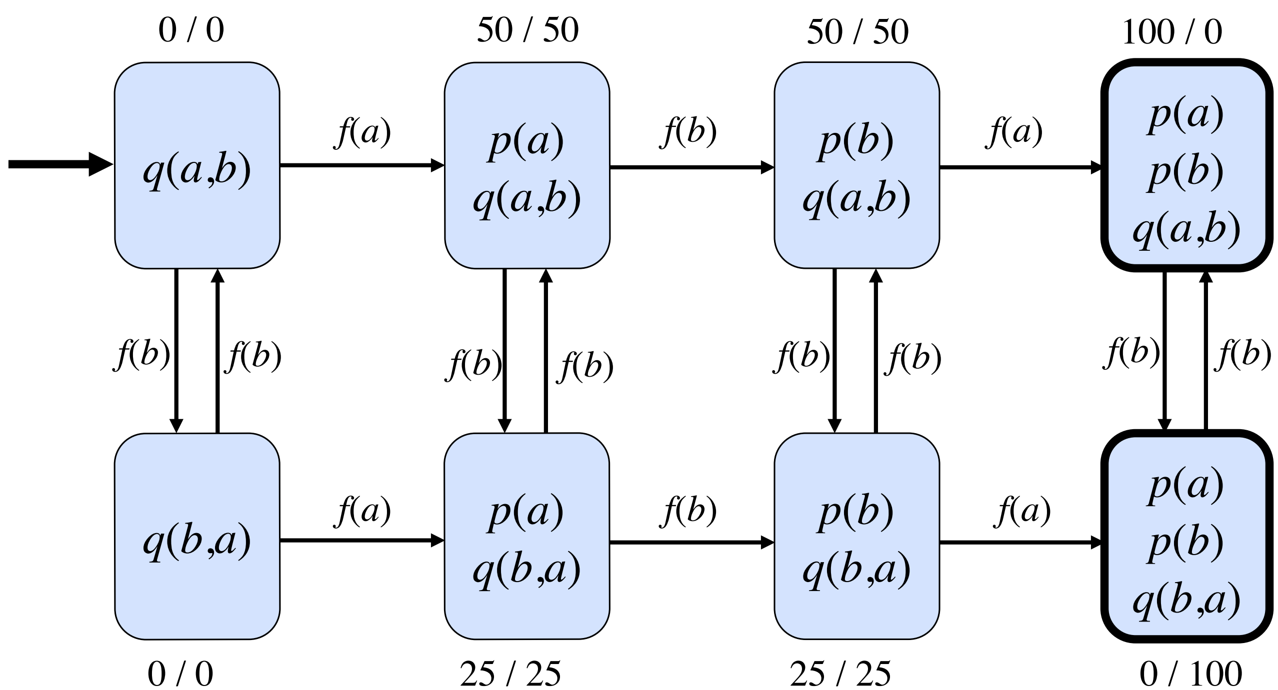
Game Description

Michael Genesereth
Computer Science Department
Stanford University

State Machine



Structured Actions and States



Logical Encoding

```
init(cell(1,1,b))
init(cell(1,2,b))
init(cell(1,3,b))
init(cell(2,1,b))
init(cell(2,2,b))
init(cell(2,3,b))
init(cell(3,1,b))
init(cell(3,2,b))
init(cell(3,3,b))
init(control(x))

legal(P,mark(X,Y)) :-
    true(cell(X,Y,b)) &
    true(control(P))

legal(x,noop) :-
    true(control(o))

legal(o,noop) :-
    true(control(x))
```

```
next(cell(M,N,P)) :-
    does(P,mark(M,N))

next(cell(M,N,Z)) :-
    does(P,mark(M,N)) &
    true(cell(M,N,Z)) & Z#b

next(cell(M,N,b)) :-
    does(P,mark(J,K)) &
    true(cell(M,N,b)) &
    (M#J | N#K)

next(control(x)) :-
    true(control(o))

next(control(o)) :-
    true(control(x))

terminal :- line(P)
terminal :- ~open

goal(x,100) :- line(x)
goal(x,50) :- draw
goal(x,0) :- line(o)

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

```
row(M,P) :-
    true(cell(M,1,P)) &
    true(cell(M,2,P)) &
    true(cell(M,3,P))

column(N,P) :-
    true(cell(1,N,P)) &
    true(cell(2,N,P)) &
    true(cell(3,N,P))

diagonal(P) :-
    true(cell(1,1,P)) &
    true(cell(2,2,P)) &
    true(cell(3,3,P))

diagonal(P) :-
    true(cell(1,3,P)) &
    true(cell(2,2,P)) &
    true(cell(3,1,P))

line(P) :- row(M,P)
line(P) :- column(N,P)
line(P) :- diagonal(P)

open :- true(cell(M,N,b))

draw :- ~line(x) &
        ~line(o)
```

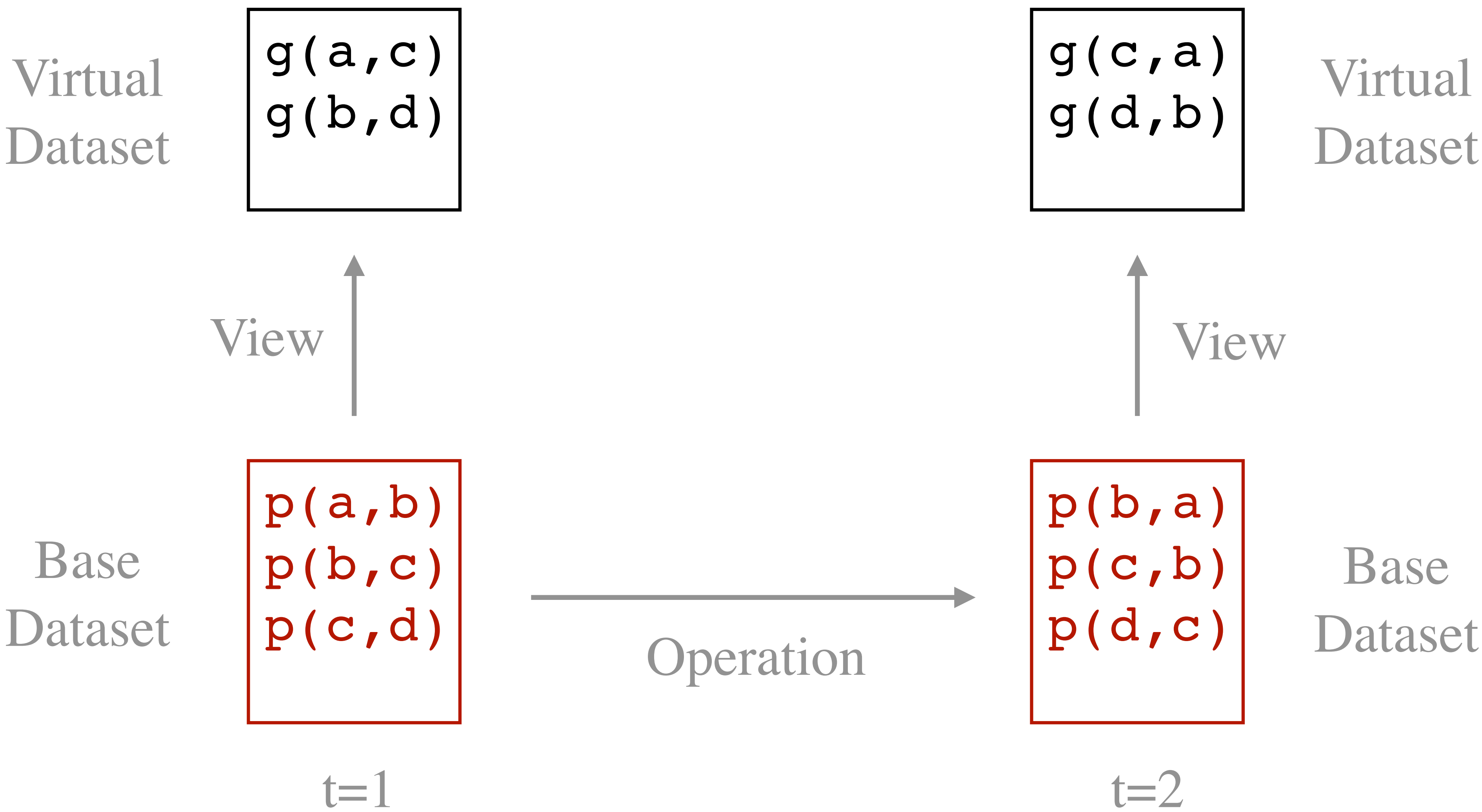
Game Description Language

Game Description Language (or GDL) is a formal language for encoding the rules of games.

GDL is widely used in the research literature and is used in virtually all General Game Playing competitions.

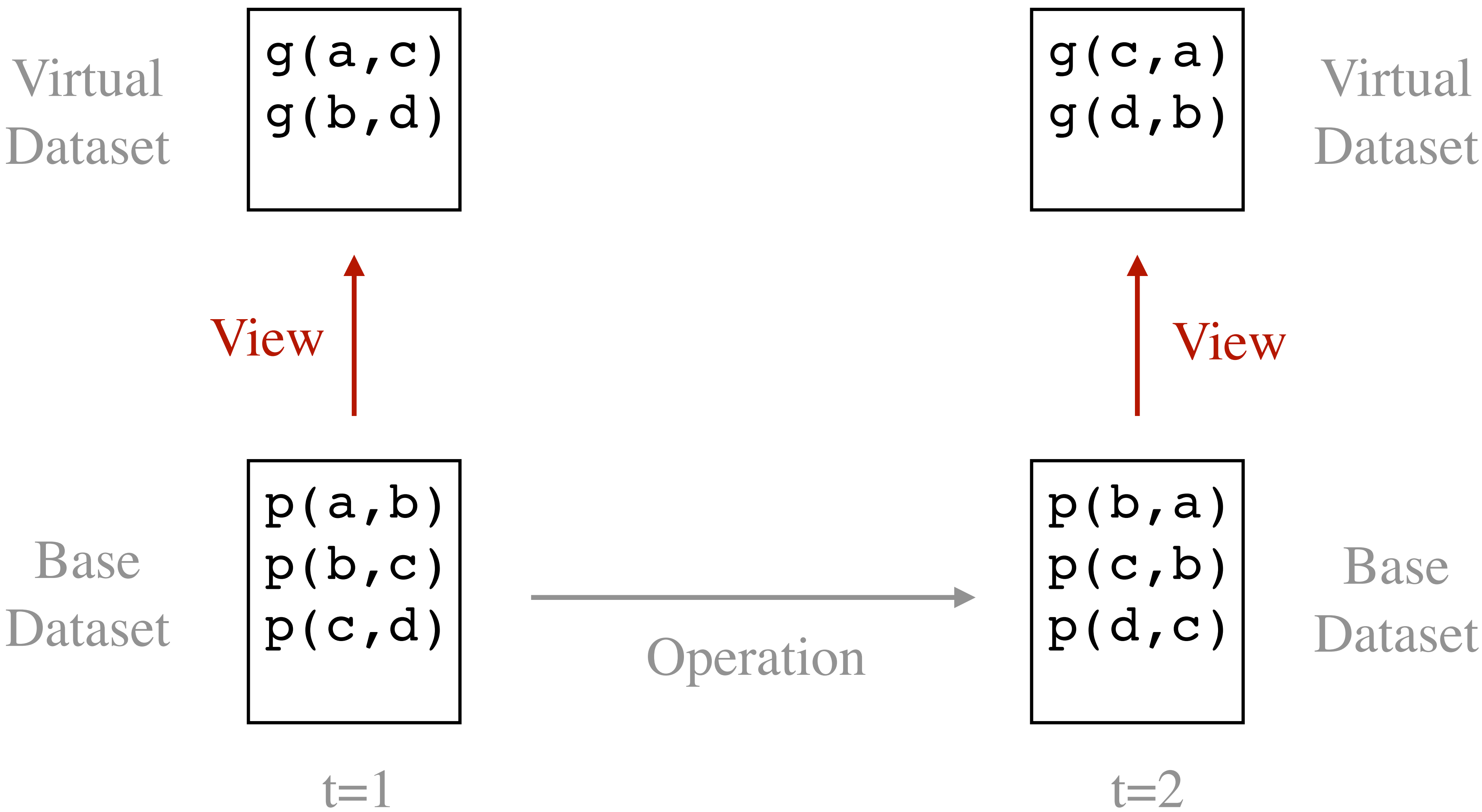
GDL is a specialization of a logic programming language called Epilog.

Datasets

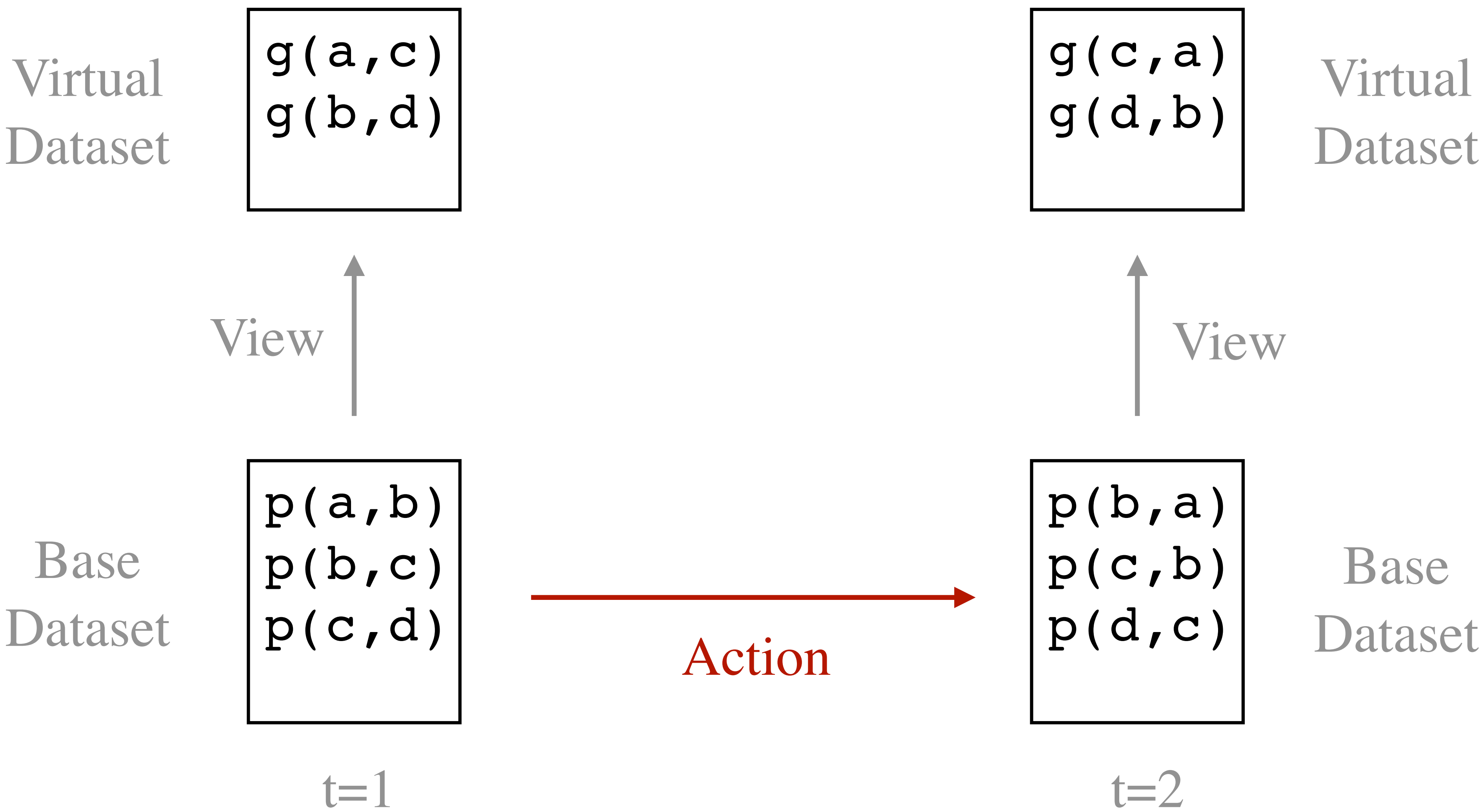


Virtual
Dataset
g(a,c)
g(b,d)
View
Base
Dataset
p(a,b)
p(b,c)
p(c,d)
t=1
Operation
g(c,a)
g(d,b)
Virtual
Dataset
View
p(b,a)
p(c,b)
p(d,c)
Base
Dataset
t=2

View Definitions



Operation Definitions



Datasets

Dataset

```
cell(1,1,x)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,o)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,x)
control(o)
```

Vocabulary

Constants are strings of lower case letters, digits, underscores, and periods.

Examples:

```
a, b, c, 1, 2, 3, king, white_knight
p, q, r, cell, adjacent
```

Non-examples:

```
Cell, white-knight
```

Types of Constants

Object constants represent objects.

`a, b, c, 1, 2, 3, king, white_knight`

Function constants represent functions.

`f, g, piece, list, set`

Relation constants represent relations.

`p, q, r, cell, adjacent`

Facts / Factoids / Data

A **fact / factoid / datum** is an expression formed from a predicate and *n* symbols enclosed in parentheses and separated by commas.

Symbols: `a, b`
Predicate: `p, q`

Sample Datum: `p(a,b)`
Sample Datum: `q(a)`

Herbrand Base

The **Herbrand base** for a vocabulary is the set of *all* factoids that can be formed from the vocabulary.

Symbols: `a, b`
Predicate: `p, q`

Herbrand Base:

`{p(a,a), p(a,b), p(b,a), p(b,b), q(a), q(b)}`

Datasets

A **dataset** is any set of factoids that can be formed from a vocabulary, i.e. a *subset of the Herbrand base*.

Herbrand Base:

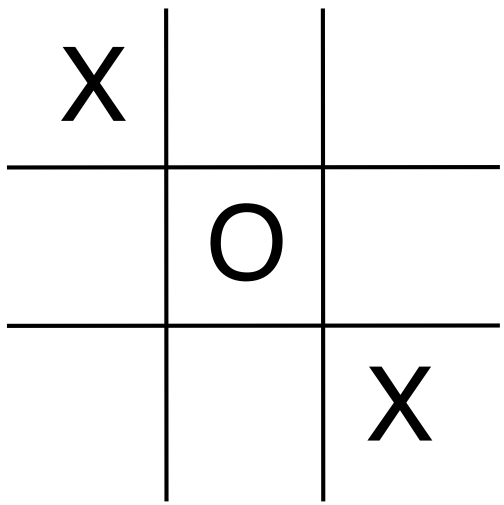
`{p(a,a), p(a,b), p(b,a), p(b,b), q(a), q(b)}`

Dataset: `{p(a,b), p(b,a), q(a)}`
Dataset: `{}`
Dataset: `{p(a,a), p(a,b), p(b,a), p(b,b), q(a), q(b)}`

We use datasets to characterize states of the world. The facts in a dataset are assumed to be true and those that are not in the dataset are assumed to be false.

States Represented as Datasets

We use datasets to characterize states of the world. The *facts in a dataset are assumed to be true* and those that are *not in the dataset are assumed to be false*.




```
cell(1,1,x)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,o)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,x)
control(o)
```

Problem

We sometimes want to talk about complex objects made up of simpler objects.

Examples:

the list of a, b, and c

the cell in row 2 and column 3

Solution: *Compound names*:

```
list(a,b,c)
piece(white,knight)
```

Types of Constants

Object constants represent objects.

```
a, b, c, 1, 2, 3, king, white_knight
```

Function constants

```
f, g, piece, list, set
```

Relation constants represent relations.

```
p, q, r, cell, adjacent
```

Compound Names (version 1)

A **compound name** is an expression formed from a *constructor* and *n symbols* enclosed in parentheses and separated by commas.

Symbols: `a,b`
Constructor: `f, g`

Compound Names: `f(a,b), f(b,a), g(a), g(b)`

This allows us to refer to complex objects made up of simple objects. How do we refer to complex objects made up of other complex objects?

Compound Names (version 2)

A **compound name** is an expression formed from a *constructor* and *n symbols* **or** *compound names* enclosed in parentheses and separated by commas.

Symbols: `a, b`
Constructor: `f, g`

Compound Names: `f(a,b), f(b,a), g(a), g(b)`
Compound Names: `f(g(a),b), g(f(a,b))`
Compound Names: `g(g(a)), g(f(g(a),g(b)))`
Compound Names: `g(g(g(a)))`
Compound Names: `g(g(g(g(a))))`

Data / Factoids

A **datum / factoid** is an expression formed from a predicate and *n symbols* **or** *compound names* enclosed in parentheses and separated by commas.

Symbols: `a, b`
Constructors: `f, g`
Predicate: `p`

Sample Datum: `p(a,g(a))`
Sample Datum: `p(f(a,b),g(b))`

The **Herbrand base** for a vocabulary is the set of all factoids that can be formed from the vocabulary.

Datasets

A **dataset** is any *set of factoids* that can be formed from a vocabulary, i.e. a subset of the Herbrand base.

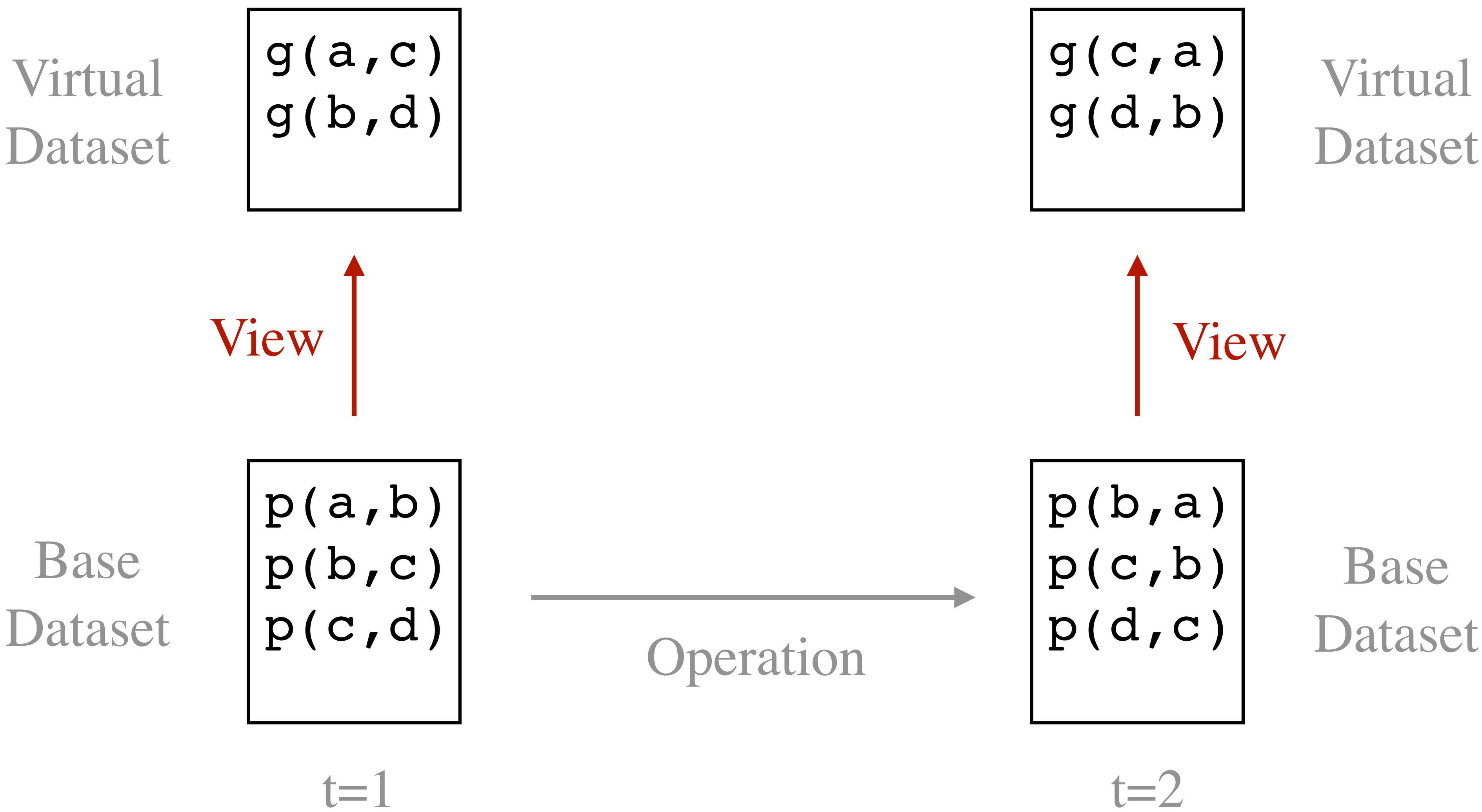
Symbols: `a, b`
Constructors: `f, g`
Predicates: `p, q`

Dataset: {`p(a,g(a)), p(g(b),a), q(f(a,b))`}

We use datasets to characterize states of the world. The *facts in a dataset are assumed to be true* and those that are *not in the dataset are assumed to be false*.

View Definitions

View Definitions



Constants and Variables

A **constant** is a string of lower case letters, digits, underscores, and periods.

```
a, b, c, 1, 2, 3, king, white_knight
f, g, piece, list, set
p, q, r, cell, adjacent
```

Same as before

A **variable** is either a lone underscore or a string of letters, digits, underscores beginning with an upper case letter.

```
X    Y23    Somebody   _
```

Rules

$$\underbrace{\texttt{r(X,Y)}}_{\textit{head}}\ \texttt{:-}\ \underbrace{\overbrace{\texttt{p(X,Y)}}^{\textit{subgoal}}\ \texttt{\&}\ \overbrace{\sim\texttt{q(Y)}}^{\textit{subgoal}}}_{\textit{body}}$$

The head of an instance of a rule is true if every positive subgoal is true and every negated subgoal is false.

Instances

An **instance of a rule** is a rule in which all variables have been consistently replaced by symbols or compound names.

Rule

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

Symbols

```
{a,b}
```

Instances

```
r(a,a) :- p(a,a) & ~q(a)
r(a,b) :- p(a,b) & ~q(b)
r(b,a) :- p(b,a) & ~q(a)
r(b,b) :- p(b,b) & ~q(b)
```

Rule Application (simplified version)

The **result** of applying r to dataset Δ is the set of all ψ such that
(1) ψ is the head of an arbitrary instance of r,
(2) every positive subgoal in the instance is in Δ, and
(3) no negated subgoal in the instance is in Δ.

The **closure** of a ruleset on a dataset is the result of
(1) applying the rules in the ruleset to facts in the dataset,
(2) adding the results to the dataset, and
(3) repeating until nothing new is added.

The full definition is slightly more complicated. See below.
http://logicprogramming.stanford.edu/miscellaneous/dlp.html

Example

Grandparents:

```
grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)
```

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
```

Example

Personhood:

```
person(X) :- parent(X,Y)
person(Y) :- parent(X,Y)
```

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
person(art)
person(bob)
person(cal)
person(cam)
person(bea)
person(cat)
person(coe)
```

Example (step 1)

Ancestors:

```
ancestor(X,Z) :- parent(X,Z)
ancestor(X,Z) :- parent(X,Y) & ancestor(Y,Z)
```

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
ancestor(art,bob)
ancestor(art,bea)
ancestor(bob,cal)
ancestor(bob,cam)
ancestor(bea,cat)
ancestor(bea,coe)
```

Example (step 2)

Ancestors:

```
ancestor(X,Z) :- parent(X,Z)
ancestor(X,Z) :- parent(X,Y) & ancestor(Y,Z)
```

Initial Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
ancestor(art,bob)
ancestor(art,bea)
ancestor(bob,cal)
ancestor(bob,cam)
ancestor(bea,cat)
ancestor(bea,coe)
ancestor(art,cal)
ancestor(art,cam)
ancestor(art,cat)
ancestor(art,coe)
```

`same, distinct, mutex`

Identity

`same(t1,t2)` is true iff t1 and t2 are *identical*

Difference

`distinct(t1,t2)` is true iff t1 and t2 are *different*

`mutex(t1,...,tn)` is true iff t1,...,t2 are *all different*

Examples

`same(a,a)`	is true
`same(a,b)`	is false
`distinct(a,a)`	is false
`distinct(a,b)`	is true
`mutex(a,b,c)`	is true

*Safety: Variables okay but no **unbound** variables allowed!!!*

Aggregates

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

Example

```
goal(X,L) :-
  p(X,Y) &
  countofall(Z,p(X,Z),L)
```

Result `{goal(a,2),goal(b,1)}`

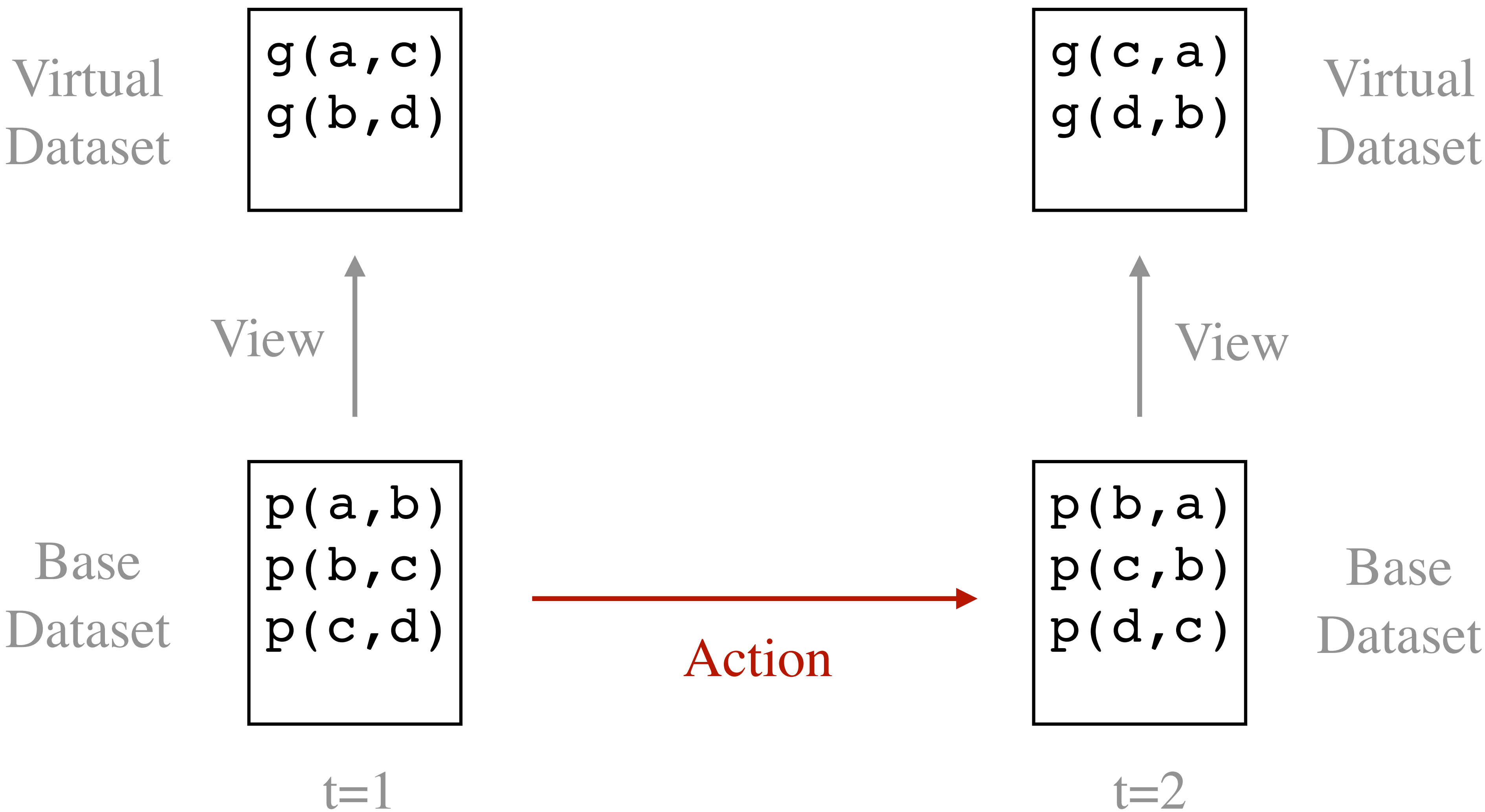
Example

```
goal(X,L) :-
  p(X,Y) &
  setofall(Z,p(X,Z),L)
```

Result `{goal(a,[b,c]),goal(b,[d])}`

Operation Definitions

Dynamics


Virtual Dataset
g(a,c)
g(b,d)
g(c,a)
g(d,b)
Virtual Dataset
View
View
Base Dataset
p(a,b)
p(b,c)
p(c,d)
Action
p(b,a)
p(c,b)
p(d,c)
Base Dataset
t=1
t=2

Operation Constants

Operation constants represent operations.

`move` - move a piece from one squre to another

`mark` - place a specific mark in a row and a column

Same spelling conventions as other constants.

Actions

An **action** is an application of an operation to objects.

In what follows, we denote actions using a syntax analogous to that of *compound names* and *factoids*, viz. an operation constant followed by n terms enclosed in parentheses and separated by commas.

Examples:

```
mark(1,2)
move(e2,e4)
```

Operation Definition Rule

head
(action)

conditions
(ordinary literals)

consequences
(base literals or actions)

```
c(X) :: p(X,Y) & q(X) ==> ~q(X) & c(Y)
```

If the head of an instance is performed and the conditions are true, then the negated consequences are removed and the positive consequences are added.

Degenerate Rules

Degenerate Rule

```
c(X) :: true ==> ~p(X) & q(X)
```

Shorthand

```
c(X) :: ~p(X) & q(X)
```

Operation Definitions

An *operation definition* is a finite collection of operation rules with the same operation in the head.

Example

```
c(X) :: p(X) & q(X)
c(X) :: ~r(X) ==> ~p(X) & r(X)
```

A *dynamic logic program* is a collection of view definitions and operation definitions.

Semantics

Given a rule set, the result of applying an action to a dataset Δ is the dataset that results from (1) *deleting all of the negative effects* of the action from the dataset and (2) *adding in all of the positive effects*.

$$\Delta - \mathit{negatives} \cup \mathit{positives}$$

Example

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

Rule:

```
u(X) :: p(X) & q(X) & ~r(X) ==> ~p(X) & r(X)
```

Action: `u(a)`

Negative effects: {`p(a)`}
Positive effects: {`r(a)`}

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

Example

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

Rule:

```
u(X) :: p(X) & q(X) & ~r(X) ==> ~p(X) & r(X)
```

Action: `u(b)`

Negative effects: none
Positive effects: none

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

Example

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

Rule:

```
u(X) :: p(X) & q(X) & ~r(X) ==> ~p(X)
u(X) :: p(X) & q(X) & ~r(X) ==> r(X)
```

Action: `u(a)`

Negative effects: {`p(a)`}
Positive effects: {`r(a)`}

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

Weird Case

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

Rule:

```
u(X) :: p(X) & q(X) ==> ~r(X)
u(X) :: p(X) & q(X) ==> r(X)
```

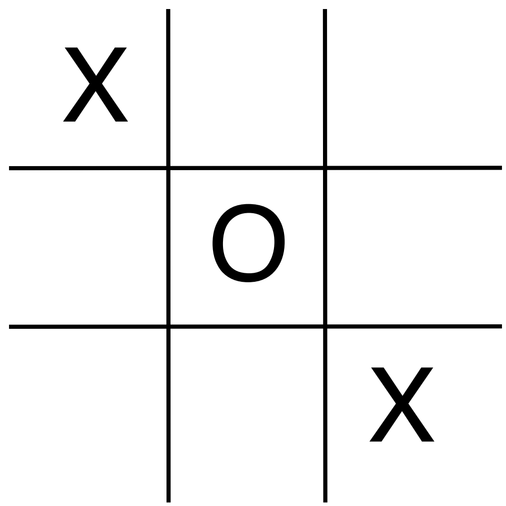
Action: `u(a)`

Negative effects: {`r(a)`}
Positive effects: {`r(a)`}

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

Game Description

States



Game-Specific Vocabulary

Objects:

`x, o` - *roles*
`1, 2, 3` - *indices of rows and columns*
`b` - *blank*

Relations:

`cell`(*index*,*index*,*mark*)
`control`(*role*)
`row`(*index*,*mark*)
`column`(*index*,*mark*)
`diagonal`(*mark*)
`line`(*mark*)
`open`

Operation:

`mark`(*index*,*index*)

Game-Independent Vocabulary

Relation Constants:

role(*role*) - means that role is a role
base(*factoid*)
action(*action*)
init(*factoid*)
control(*role*)
legal(*action*)
goal(*role,number*)
terminal

Object Constants:

0, 1, 2, 3, … , 100 – *numbers*

GDL Descriptions

A **game description** in GDL is a dynamic logic program (1) that defines the *game-independent relations* in terms of the game's *game-specific vocabulary* and (2) that defines the *operations* of the game.

Independent of state:

`role`, `base`, `action`, `init`

State-dependent:

`control`, `legal`, `goal`, `terminal`

Operations

Roles

```
role(x)
role(o)
```

Base Relations and Input Operations

Objects:

`x, o` - *roles*
`1, 2, 3` - *indices of rows and columns*
`b` - *blank*

Relations:

→ `cell`(*index*,*index*,*mark*)
→ `control`(*role*)
`row`(*index*,*mark*)
`column`(*index*,*mark*)
`diagonal`(*mark*)
`line`(*mark*)
`open`

Operation:

→ `mark`(*index*,*index*)

Base Propositions

```
base(cell(X,Y,W)) :-
  index(X) &
  index(Y) &
  filler(W)

base(control(W)) :-
  role(W)

index(1)
index(2)
index(3)

filler(x)
filler(o)
filler(b)
```

```
cell(1,1,x)
cell(1,2,o)
cell(1,3,b)
cell(2,1,x)
cell(2,1,o)
cell(2,1,b)
    ...
cell(3,2,x)
cell(3,2,o)
cell(3,2,b)
cell(3,3,x)
cell(3,3,o)
cell(3,3,b)
control(x)
control(o)
```

Input Actions

```
action(mark(X,Y)) :-
  index(X) &
  index(Y)
```

```
mark(1,1)
mark(1,2)
mark(1,3)
mark(2,1)
mark(2,2)
mark(2,3)
mark(3,1)
mark(3,2)
mark(3,3)
```

Initial State

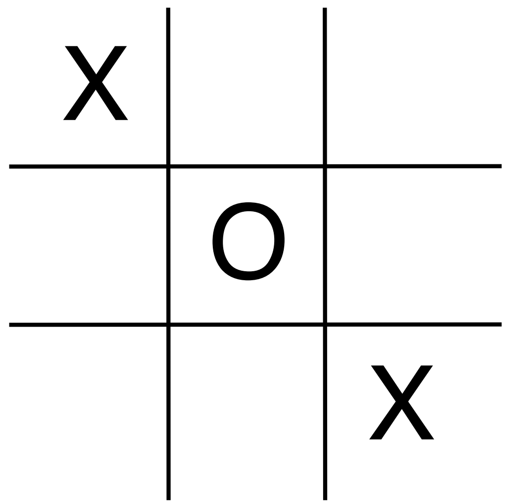
```
init(cell(1,1,b))                  cell(1,1,b)
init(cell(1,2,b))                  cell(1,2,b)
init(cell(1,3,b))                  cell(1,3,b)
init(cell(2,1,b))                  cell(2,1,b)
init(cell(2,2,b))                  cell(2,2,b)
init(cell(2,3,b))                  cell(2,3,b)
init(cell(3,1,b))                  cell(3,1,b)
init(cell(3,2,b))                  cell(3,2,b)
init(cell(3,3,b))                  cell(3,3,b)
init(control(x))                   control(x)
```

Legal Moves

```
legal(mark(M,N)) :- cell(M,N,b)
```

State:

```
cell(1,1,x)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,o)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,x)
control(o)
```



Legal Moves:

```
mark(1,2)
mark(1,3)
mark(2,1)
mark(2,3)
mark(3,1)
mark(3,2)
```

Physics

```
mark(M,N) ::
   control(Z) ==> ~cell(M,N,b) & cell(M,N,Z)
mark(M,N) ::
   control(x) ==> ~control(x) & control(o)
mark(M,N) ::
   control(o) ==> ~control(o) & control(x)
```

cell(1,1,x)		cell(1,1,x)
cell(1,2,b)		cell(1,2,b)
cell(1,3,b)		cell(1,3,o)
cell(2,1,b)		cell(2,1,b)
cell(2,2,o)	mark(1,3) →	cell(2,2,o)
cell(2,3,b)		cell(2,3,b)
cell(3,1,b)		cell(3,1,b)
cell(3,2,b)		cell(3,2,b)
cell(3,3,x)		cell(3,3,x)
control(o)		control(x)

Supporting Concepts

```
row(M,Z) :- cell(M,1,Z) & cell(M,2,Z) & cell(M,3,Z)
col(M,Z) :- cell(1,N,Z) & cell(2,N,Z) & cell(3,N,Z)
diag(Z) :- cell(1,1,Z) & cell(2,2,Z) & cell(3,3,Z)
diag(Z) :- cell(1,3,Z) & cell(2,2,Z) & cell(3,1,Z)
```

```
line(Z) :- row(M,Z)
line(Z) :- col(M,Z)
line(Z) :- diag(Z)
```

```
open :- cell(X,Y,b)
```

Goals and Termination

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

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


terminal :- line(W)
terminal :- ~open
```

Termination

A game description in GDL *terminates* if and only if every sequence of legal moves from the initial state reaches a terminal state after a finite number of steps.

Playability

A game is playable if and only some player has control in every non-terminal state *and* the player in control has at least one legal move in every non-terminal state.

Note that in chess, if a player cannot move, it is a stalemate. Fortunately, this is a terminal state.

In GGP, we guarantee that every game is playable.

Winnability

A game is strongly winnable if and only if, for some player, there is a sequence of individual moves of that player that leads to a goal state in which that player gets 100 points.

A game is weakly winnable if and only if, for every player, there is some sequence of moves of the players that leads to a a goal state in which that player gets more than 0 points.

In GGP, every game is weakly winnable, and all single player games are strongly winnable.

Obfuscation

What we see:

```
mark(M,N) ::
  control(Z) ==> ~cell(M,N,b) & cell(M,N,Z)
```

What the player sees:

```
welcoul(M,N) ::
  control(Z) ==> ~dupse(M,N,erol) & dupse(M,N,Z)
```

Game Resources

Sierra

Sierra is browser-based IDE (interactive development environment) for Epilog.

Saving and loading files

Visualization of datasets
Querying datasets
Transforming datasets

Interpreter (for view definitions, action definitions)
Trace capability (useful for debugging rules)
Analysis tools (error checking and optimizing rules)

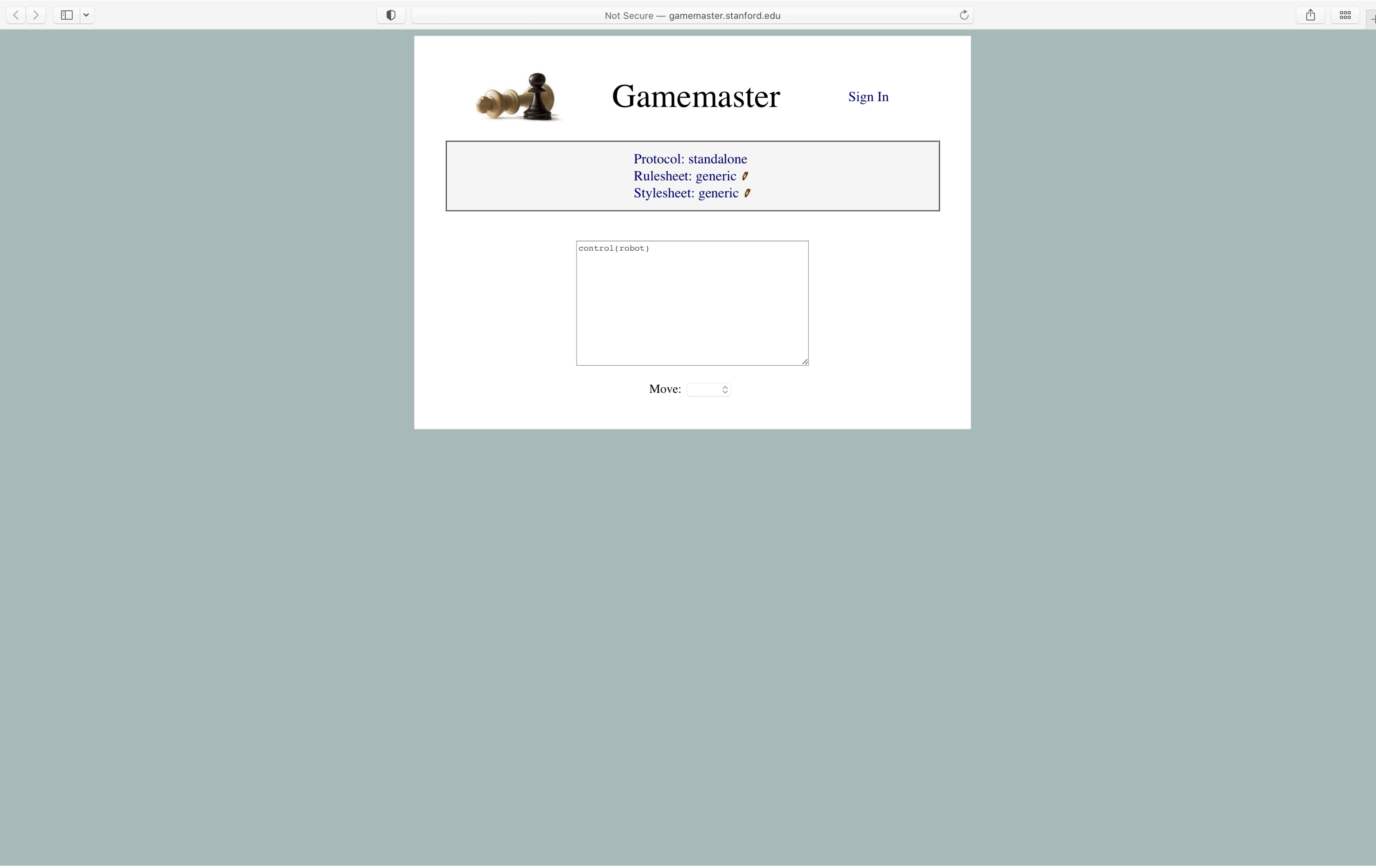
http://epilog.stanford.edu/sierra/sierra.html

Rule Checker

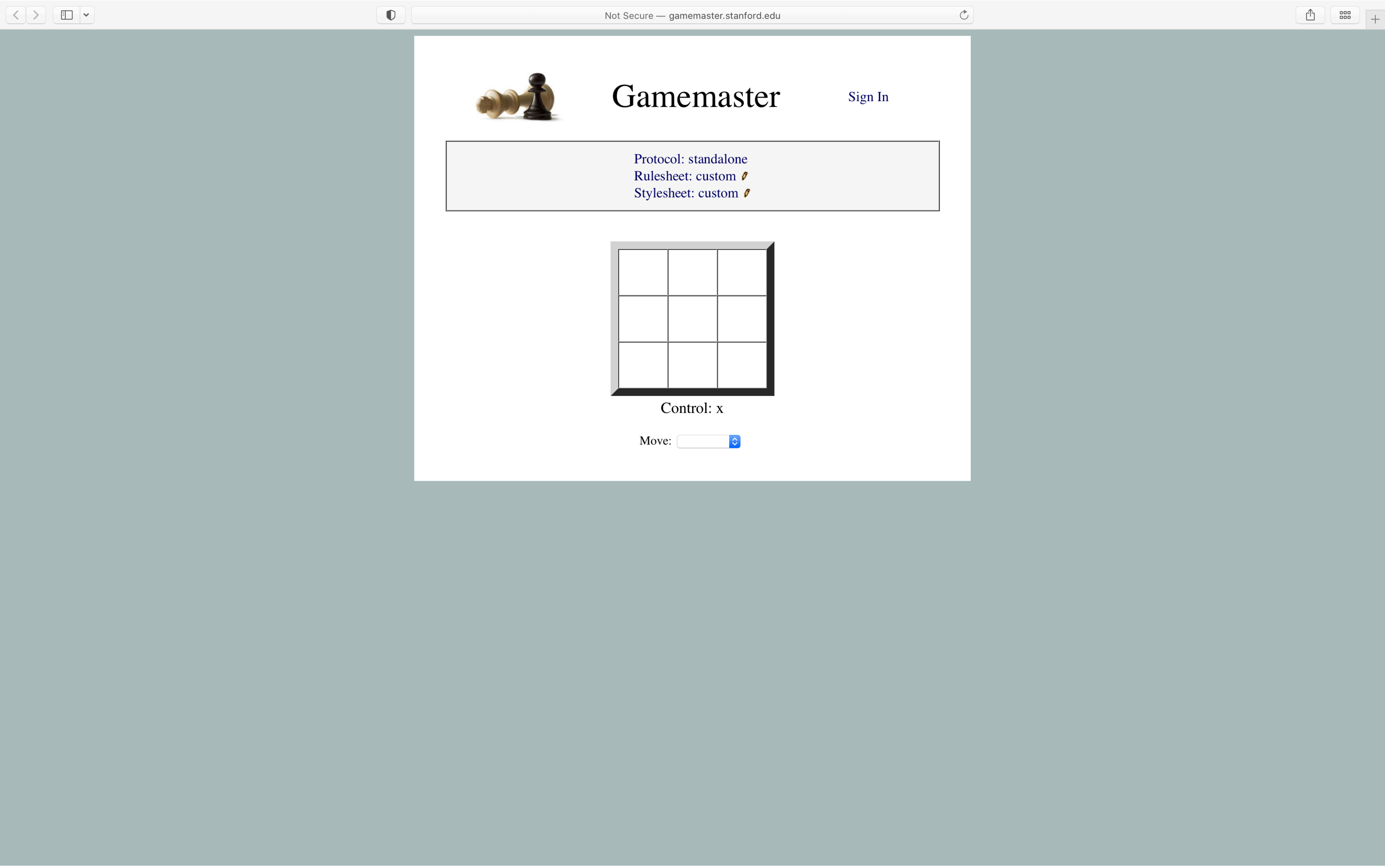
http://gamemaster.stanford.edu/homepage/rulecheckerintro.php

Style Checker

http://gamemaster.stanford.edu/homepage/stylecheckerintro.php



http://gamemaster.stanford.edu/homepage/standaloneopen.php



http://gamemaster.stanford.edu/homepage/standaloneopen.php

Not Secure — gamemaster.stanford.edu

Gamemaster

Sign In

Protocol: localstorage
Strategy: manager
Identifier: manager

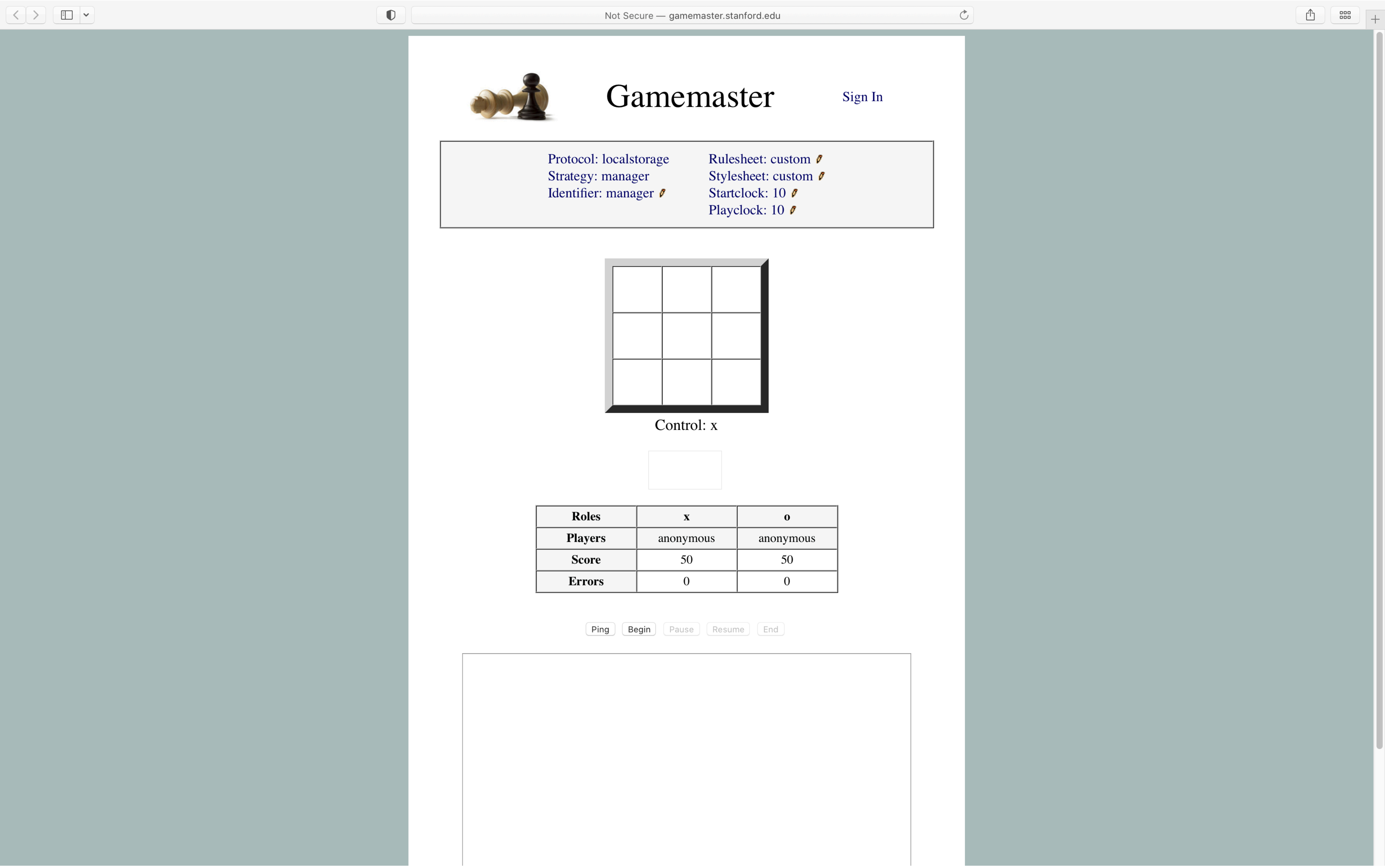
Rulesheet: generic
Stylesheet: generic
Startclock: 10
Playclock: 10

```
control(robot)
```

Roles	robot
Players	anonymous
Score	0
Errors	0

Ping Begin Pause Resume End

http://gamemaster.stanford.edu/homepage/manageropen.php



http://gamemaster.stanford.edu/homepage/manageropen.php


GENERAL
GAME
PLAYING