

Logic Programming

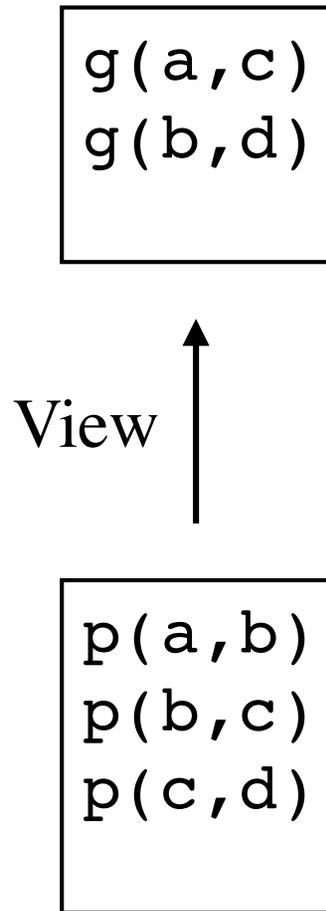
Operation Definitions

Michael Genesereth
Computer Science Department
Stanford University

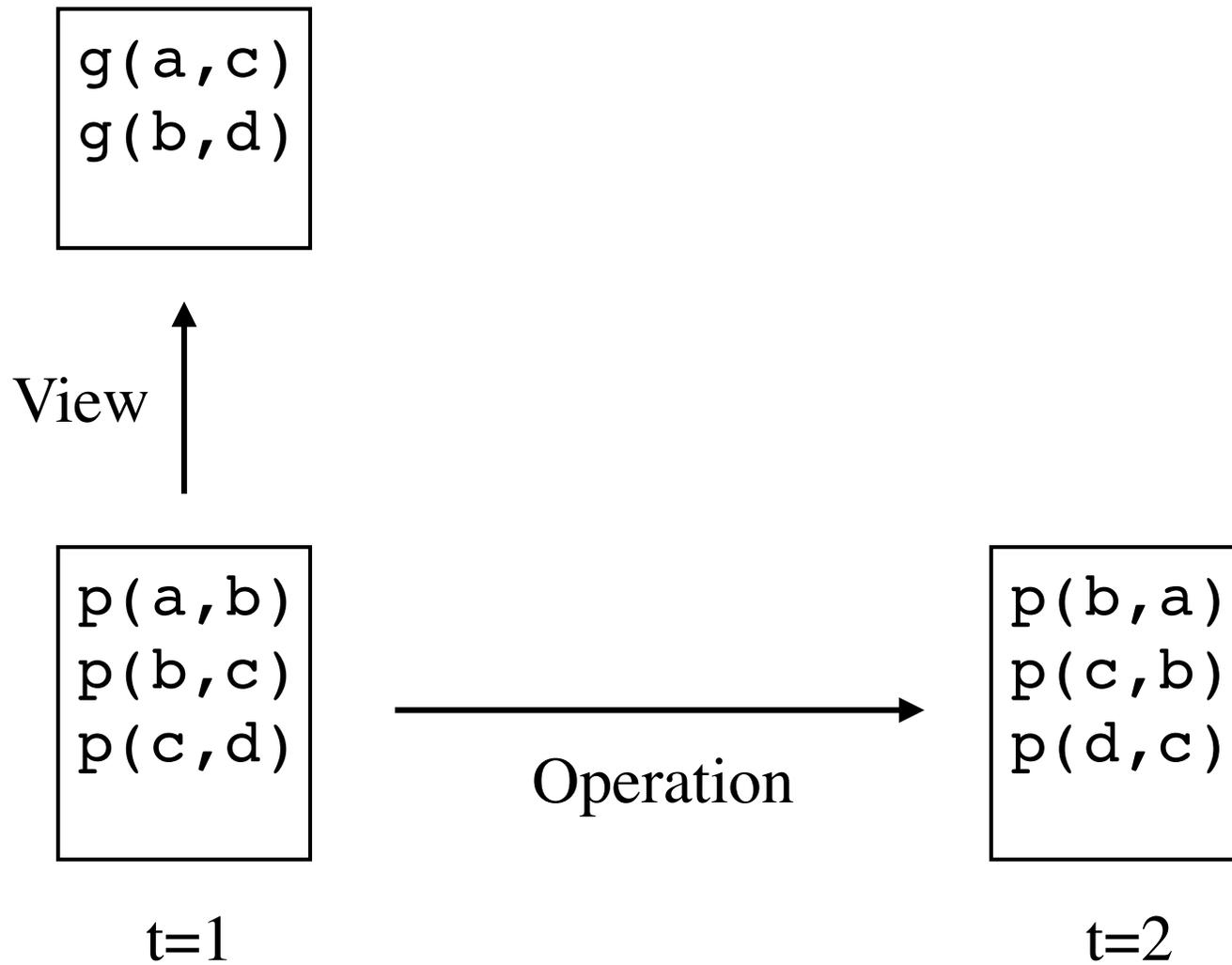
Datasets

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

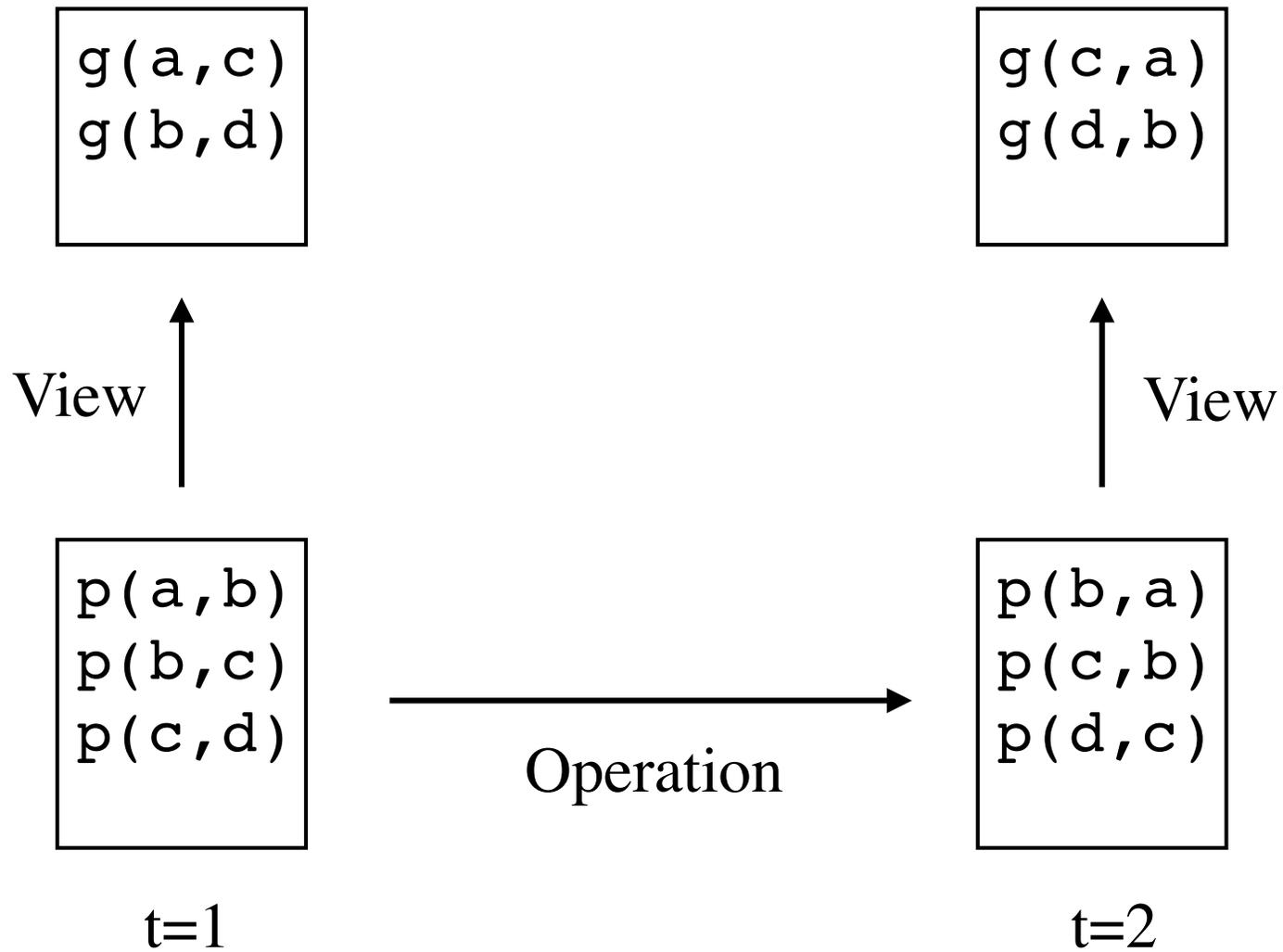
Views



Operations



Operations



Operation Definitions

View Definitions

$$r(X, Y) \text{ :- } p(X, Y) \ \& \ \sim q(Y)$$
$$s(X, Y) \text{ :- } r(X, Y) \ \& \ r(Y, Z)$$

Operation Definitions

$$\text{flip}(X) \text{ :: } p(X) \ \& \ \sim q(X) \implies \sim p(X) \ \& \ q(X)$$
$$\text{flop}(X) \text{ :: } r(X, Y) \implies \text{flip}(X) \ \& \ \text{flop}(Y)$$

Program Sheets

AI Program Requirements Quarters: ■ autumn ■ winter ■ spring

- Take at least 6 courses
 - Take at most 3 courses per quarter

<input type="checkbox"/> CS 124	<input type="checkbox"/> CS 131	<input checked="" type="checkbox"/> CS 157	<input type="checkbox"/> CS 223A
<input type="checkbox"/> CS 224N	<input type="checkbox"/> CS 225A	<input checked="" type="checkbox"/> CS 227B	<input checked="" type="checkbox"/> CS 228
<input type="checkbox"/> CS 229	<input type="checkbox"/> CS 231N	<input type="checkbox"/> CS 238	<input type="checkbox"/> CS 273A
<input type="checkbox"/> CS 273B	<input type="checkbox"/> CS 276	<input type="checkbox"/> CS 279	<input checked="" type="checkbox"/> CS 331B

Focus on AI Topics

Legend:

- Machine Learning (Dark Blue)
- Natural Language Processing (Light Blue)
- Computational Biology (Red)
- Computer Vision (Orange)
- Information Retrieval (Green)
- Logic (Purple)
- Robotics (Pink)

Course	Units	Total: 13	Professor	# Courses
CS 157	3	3		2
CS 227B	3	3		1
CS 228	3	3		1
CS 331B	4	4		

Demonstration

Portico (Symbium)

Not Secure — complaw.stanford.edu

Portico

Use sliders to adjust view. Click and drag to move building. Click Larger, Smaller, Taller, Shorter to adjust size.

Larger Smaller Turn Stop Taller Shorter

Item	Data
Zone	R-1

Standard	Actual	Allowed	Status
Footprint	160000	168000	✓

Item	Min	Max
Home x	200	600

Demonstration

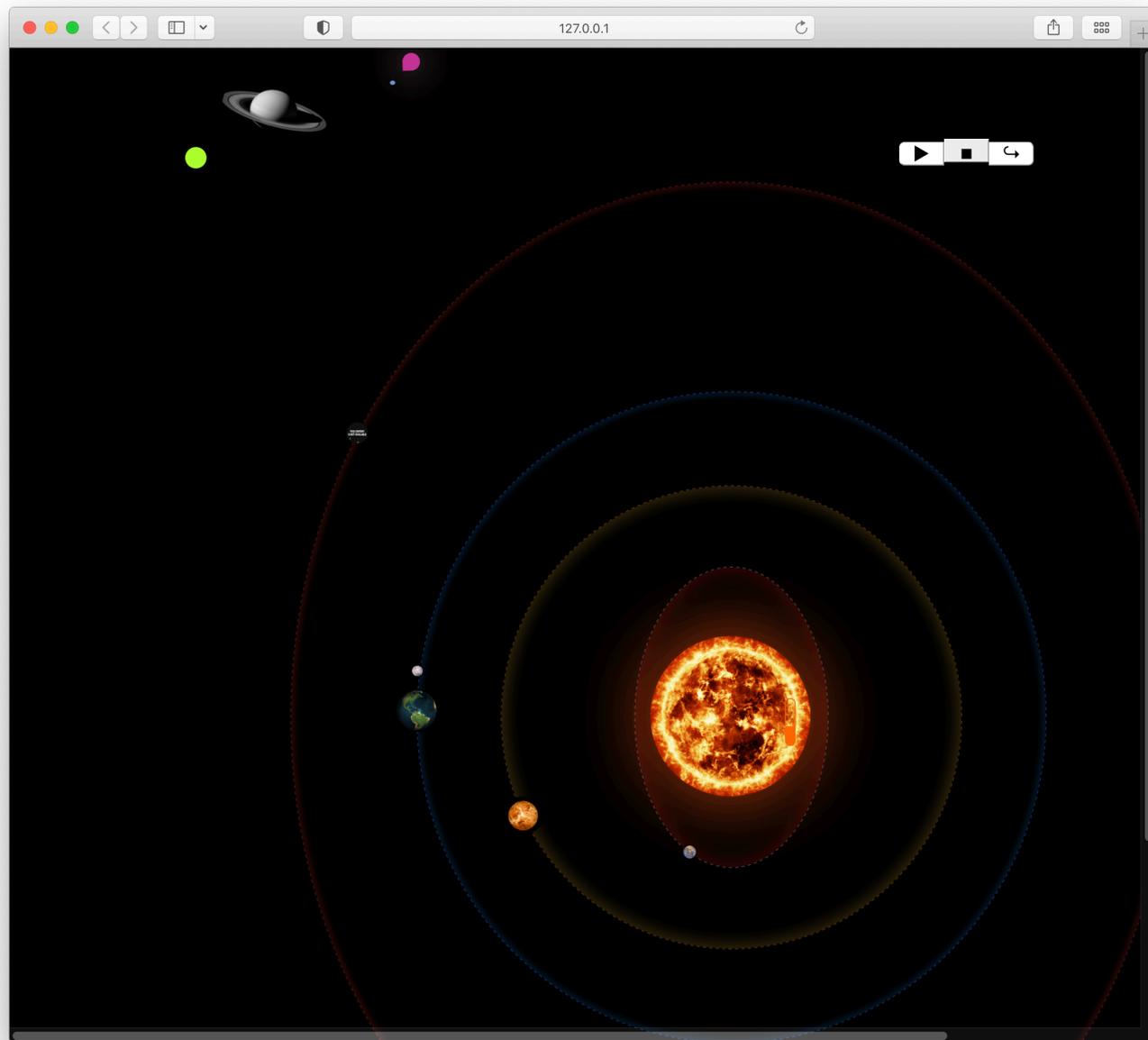
Trifecta

The screenshot shows a web browser window with the URL 127.0.0.1. The game interface includes a 'Restart' button at the top left. Below it, the 'Deck' section displays five cards: a black card with a yellow heart, a red card with a black spade, a grey card with three black diamonds, a purple card with four white spades, a black card with a white heart, and a red card with three white clubs. The 'X' player's hand consists of a red card with four yellow diamonds and a purple card with three yellow diamonds. The 'O' player's hand consists of a grey card with four black clubs. At the bottom, a table shows the current scores for both players.

Player	Score
50	50

Demonstration

Solar System



Demonstration

Syntax

Operation Constants

Operation constants represent operations.

`tick` - tick of the clock

`click` - click a button on a web page

`stack` - place one block on another

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

Same spelling conventions as other constants.

Like constructors, and predicates, each has a specific arity.

`tick/0`

`click/1`

`stack/2`

`mark/3`

Actions

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

In what follows, we denote actions using a syntax similar to that of compound terms, viz. an n -ary operation constant followed by n terms enclosed in parentheses (as appropriate) and separated by commas.

Examples:

```
tick  
click(a)  
stack(a,b)  
mark(x,2,3)
```

Syntactically, actions are treated as terms.

Operation Definition

$$\underbrace{c(a)}_{\text{head}} :: \underbrace{p(a,b) \ \& \ q(a)}_{\text{conditions}} \implies \underbrace{\sim q(a) \ \& \ c(b)}_{\text{effects}}$$

(action) *(ordinary literals)* *(base literals or actions)*

Variables

$c(X) :: p(X, Y) \ \& \ q(X) \implies \sim q(X) \ \& \ c(Y)$

Safety

A operation rule is **safe** if and only if every variable in every literal on the right hand side appears in the head or in a positive literal on the left hand side. Also, every variable in a negative literal on the left hand side appears in a prior positive literal.

Safe Operation Rule

$$\begin{aligned} c(X) &:: \\ p(X, Y) \ \& \ \sim q(X) & \implies \\ \sim p(X, Y) \ \& \ q(X) \ \& \ c(Y) & \end{aligned}$$

Unsafe Operation Rule

$$\begin{aligned} c(X) &:: \\ p(X, Y) \ \& \ \sim q(Z) & \implies \\ \sim p(X, Y) \ \& \ q(W) \ \& \ c(Y) & \end{aligned}$$

Degenerate Rules

Degenerate Rule

$c(X) :: \text{true} \implies \sim p(X) \ \& \ q(X)$

Shorthand

$c(X) :: \sim p(X) \ \& \ q(X)$

Dynamic Logic Programs

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

Example

$$c(X) :: p(X) \ \& \ q(X)$$
$$c(X) :: \sim r(X) \implies \sim p(X) \ \& \ r(X)$$

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

Semantics

Intuition

Given a dynamic logic program, the result of applying an action to a dataset is the dataset that results from

(1) *deleting all of the negative effects* of the action

and then

(2) *adding in all of the positive effects*.

Active and Inactive Rule Instances

Given a ruleset Ω with dataset Δ *and* a set Γ of actions, an *instance* of an operation rule in Ω is **active** if and only if

(1) the head of the rule is in Γ

(2) the conditions of the rule are all true in Δ .

Otherwise, the instance is **inactive**.

Example

Data: $p(a), p(b), p(c), q(a), q(b), q(c), r(b)$

Rule:

$$u(X) :: p(X) \ \& \ q(X) \ \& \ \sim r(X) \implies \sim p(X) \ \& \ r(X)$$

Action: $u(a)$

Active Instance:

$$u(a) :: p(a) \ \& \ q(a) \ \& \ \sim r(a) \implies \sim p(a) \ \& \ r(a)$$

Inactive Instances:

$$u(b) :: p(b) \ \& \ q(b) \ \& \ \sim r(b) \implies \sim p(b) \ \& \ r(b)$$

$$u(c) :: p(c) \ \& \ q(c) \ \& \ \sim r(c) \implies \sim p(c) \ \& \ r(c)$$

Expansion

The **expansion*** of an action set with respect to a rule set is the set of all effects in *any* active instance of *any* operation definition rule.

The **positive updates** of an action with respect to a rule set are the positive literals in the expansion.

The **negative updates** of an action with respect to a rule set are the negative literals in the expansion.

**Simple version*

Example

Data: $p(a)$, $p(b)$, $p(c)$, $q(a)$, $q(b)$, $q(c)$, $r(b)$

Rule:

$u(X) :: p(X) \ \& \ q(X) \ \& \ \sim r(X) \ ==> \ \sim p(X) \ \& \ r(X)$

Action: $u(a)$

Active Instance:

$u(a) :: p(a) \ \& \ q(a) \ \& \ \sim r(a) \ ==> \ \sim p(a) \ \& \ r(a)$

Expansion: $\sim p(a)$, $r(a)$

Negative Update: $p(a)$

Positive Update: $r(a)$

Result

Given a rule set, the **result** of applying an action set to dataset Δ is the set consisting of all factoids in Δ *minus* the negative updates *plus* the positive updates.

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

Example

Data: $p(a)$, $p(b)$, $p(c)$, $q(a)$, $q(b)$, $q(c)$, $r(b)$

Rule:

$u(X) :: p(X) \ \& \ q(X) \ \& \ \sim r(X) \implies \sim p(X) \ \& \ r(X)$

Action: $u(a)$

Negative Updates: $p(a)$

Positive Updates: $r(a)$

Result: $p(b)$, $p(c)$, $q(a)$, $q(b)$, $q(c)$, $r(a)$, $r(b)$

Multiple Rules

Dataset: $p(a)$, $p(b)$, $p(c)$, $q(a)$, $q(b)$, $q(c)$, $r(b)$

Rule:

$u(X) :: p(X) \ \& \ q(X) \ \& \ \sim r(X) \ ==> \ \sim p(X)$

$u(X) :: p(X) \ \& \ q(X) \ \& \ \sim r(X) \ ==> \ r(X)$

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:

$u(X) :: p(X) \ \& \ q(X) \ ==> \sim r(X)$

$u(X) :: p(X) \ \& \ q(X) \ ==> r(X)$

Action: $u(a)$

Negative effects: $r(a)$

Positive effects: $r(a)$

Result: $p(a), p(b), p(c), q(a), q(b), q(c), r(a)$

Simultaneous Actions

Data: $p(a), p(b), p(c), q(a), q(b), q(c), r(b)$

Rule:

$u(X) :: p(X) \ \& \ q(X) \ \& \ \sim r(X) \implies \sim p(X) \ \& \ r(X)$

Actions: $u(a), u(b), u(c)$

Active Instances:

$u(a) :: p(a) \ \& \ q(a) \ \& \ \sim r(a) \implies \sim p(a) \ \& \ r(a)$

$u(c) :: p(c) \ \& \ q(c) \ \& \ \sim r(c) \implies \sim p(c) \ \& \ r(c)$

Inactive Instance:

$u(b) :: p(b) \ \& \ q(b) \ \& \ \sim r(b) \implies \sim p(b) \ \& \ r(b)$

Simultaneous Actions

Data: $p(a)$, $p(b)$, $p(c)$, $q(a)$, $q(b)$, $q(c)$, $r(b)$

Rule:

$u(X) :: p(X) \ \& \ q(X) \ \& \ \sim r(X) \implies \sim p(X) \ \& \ r(X)$

Actions: $u(a)$, $u(b)$, $u(c)$

Expansion: $\sim p(a)$, $\sim p(c)$, $r(a)$, $r(c)$

Negative Updates: $p(a)$, $p(c)$

Positive Updates: $r(a)$, $r(c)$

Result: $p(b)$, $q(a)$, $q(b)$, $q(c)$, $r(a)$, $r(b)$, $r(c)$

Interchange

```
function interchange ()  
  {x = y;  
   y = x}
```

```
[x, y]  
[3, 4]
```

```
interchange()
```

```
[x, y]  
[4, 4]
```

```
function interchange ()  
  {var z = x;  
   x = y;  
   y = z}
```

Interchange

```
interchange ::
```

```
  val(x,X) & val(y,Y) ==>  
    ~val(x,X) & ~val(y,Y) &  
    val(x,Y) & val(y,X)
```

```
val(x,3)
```

```
val(y,4)
```

```
Execute: interchange
```

```
val(x,4)
```

```
val(y,3)
```

Production Systems

A **production system** is a set of condition-action rules. On each step in the execution of a production system, an active rule is chosen and the actions are performed. The cycle then repeats on the new state.

```
if p(X), then del p(X) and add q(X)
if q(X), then del q(X) and add p(X)
```

Before: $\{p(a), q(b)\}$

Step 1: $\{q(a), q(b)\}$

Step 2: $\{p(a), q(b)\}$ *or* $\{p(b), q(a)\}$

When do we stop?

Dynamic Logic Programs

Dynamic logic programs differ from production systems in that all active transition rules “fire” at the same time. (1) All updates are computed *before* any changes are made, and (2) all changes are made simultaneously.

```
tick :: p(X) ==> ~p(X) & q(X)
tick :: q(X) ==> ~q(X) & p(X)
```

Before: {p(a), q(b)}
After: {p(b), q(a)}

Derived Actions

Data: $p(a)$, $p(b)$, $p(c)$, $q(a)$, $q(b)$, $q(c)$, $r(b)$

Rule:

$u(X) :: p(X) \ \& \ q(X) \ ==> \sim p(X) \ \& \ r(X) \ \& \ u(c)$

Input Action: $u(a)$

Derived action: $u(c)$

Expansion: $\sim p(a)$, $\sim p(c)$, $r(a)$, $r(c)$, $u(a)$, $u(c)$

Negative Updates: $\{p(a), p(c)\}$

Positive Updates: $\{r(a), r(c)\}$

Result: $p(b)$, $q(a)$, $q(b)$, $q(c)$, $r(a)$, $r(b)$, $r(c)$

Expansion

Given a rule set Ω and a dataset Δ a set Γ of actions, consider the following series.

$$\Gamma_0 = \Gamma$$

Γ_{n+1} = the set of all effects of Γ in any active rule instance

The **expansion*** of Γ with respect to Ω and Δ is the fixpoint of this series.

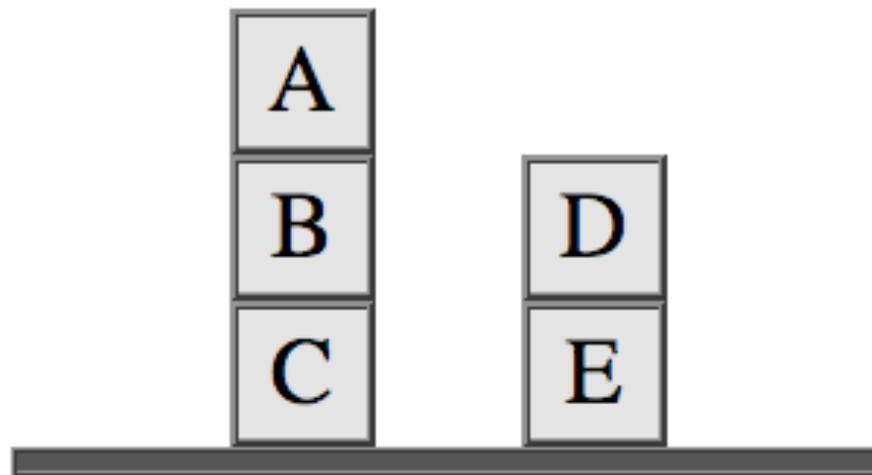
The **positive updates** of an action with respect to a rule set are the positive literals in the full expansion.

The **negative updates** of an action with respect to a rule set are the negative literals in the full expansion.

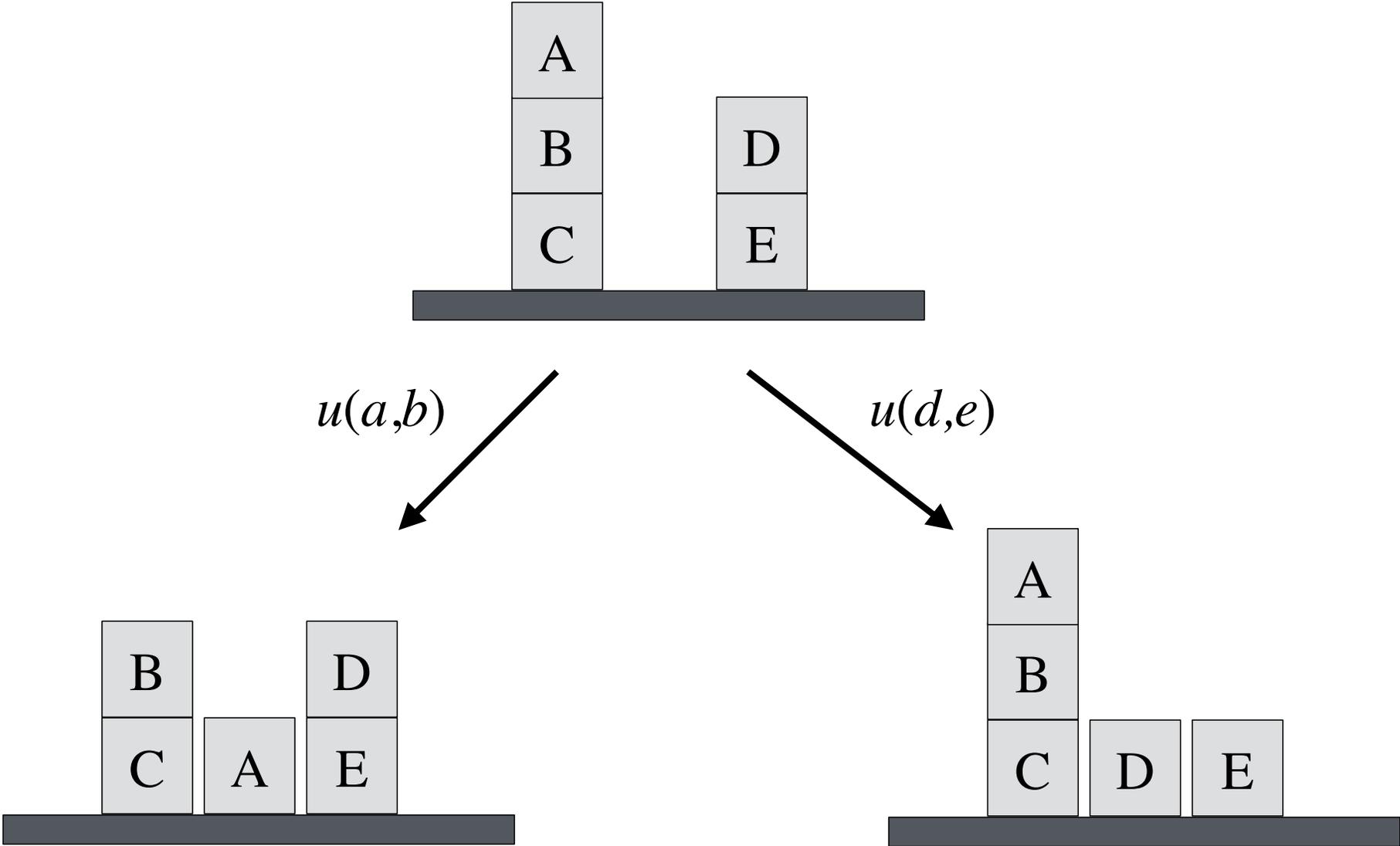
**Exact version*

Blocks World

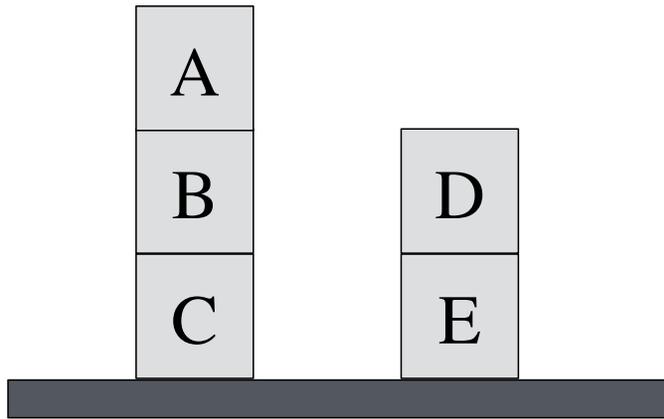
Blocks World



External Actions

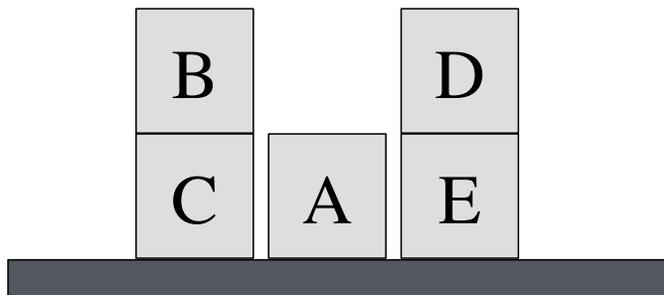


Describing States



```
clear(a)  
on(a,b)  
on(b,c)  
on(d,e)  
...
```

$u(a,b)$



```
clear(a)  
table(a)  
clear(b)  
on(b,c)  
on(d,e)  
...
```

Operation Definitions

Operations:

$u(x, y)$ means that x is moved from y to the table.

$s(x, y)$ means that x is moved from the table to y .

Operation Definitions:

$u(X, Y) ::$

`clear(X) & on(X, Y)`

`==> ~on(X, Y) & table(X) & clear(Y)`

Operation Definitions

Operations:

$u(x, y)$ means that x is moved from y to the table.

$s(x, y)$ means that x is moved from the table to y .

Operation Definitions:

$u(X, Y) ::$

$clear(X) \ \& \ on(X, Y)$

$\implies \sim on(X, Y) \ \& \ table(X) \ \& \ clear(Y)$

$s(X, Y) ::$

$table(X) \ \& \ clear(X) \ \& \ clear(Y)$

$\implies \sim table(X) \ \& \ \sim clear(Y) \ \& \ on(X, Y)$

The Game of Life

Rules of the Game

- (1) Any *live* cell with *two or three* live neighbors lives on to the next generation.
- (2) Any *live* cell with *fewer than two* live neighbors dies (as if caused by underpopulation).
- (3) Any *live* cell with *more than three* live neighbors dies (as if by overpopulation).
- (4) Any *dead* cell with *exactly three* live neighbors becomes a live cell (as if by reproduction).

Vocabulary

Symbols: c_{11} , c_{12} , ...

Unary Predicates:

`on` - cell is live

`cell` - true of cells

Binary Predicates:

`neighbor` - cells are neighbors

Loneliness

Any *live* cell with *fewer than two* live neighbors dies.

```
tick ::
```

```
  on(Y) & countofall(X,neighbor(X,Y)&on(X),0)
```

```
==> ~on(Y)
```

```
tick ::
```

```
  on(Y) & countofall(X,neighbor(X,Y)&on(X),1)
```

```
==> ~on(Y)
```

Overcrowding

Any *live* cell with *more than three* live neighbors dies.

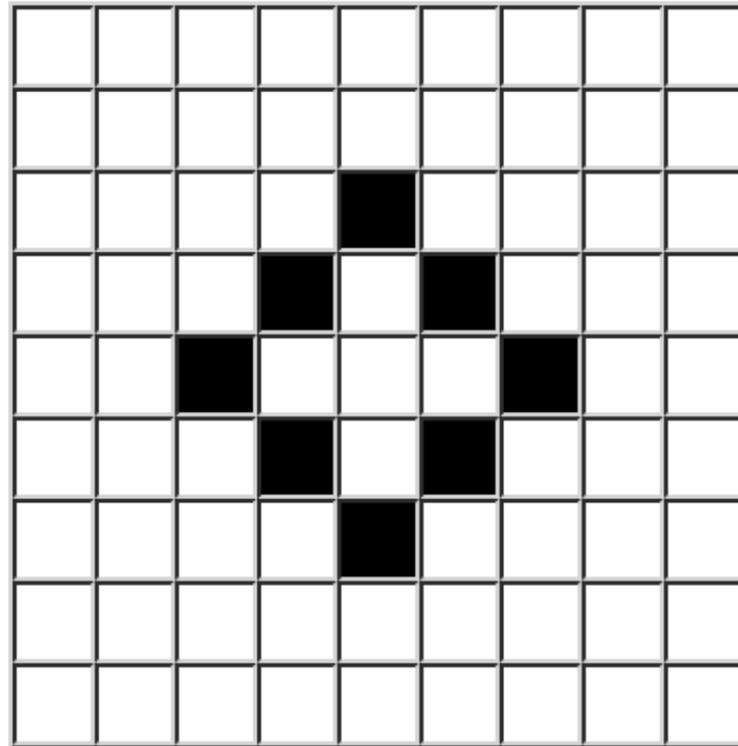
```
tick ::  
  on(Y) &  
  countofall(X, neighbor(X, Y) & on(X), N) &  
  leq(4, N)  
==> ~on(Y)
```

Reproduction

Any *dead* cell with *exactly three* live neighbors becomes live.

```
tick ::  
  cell(Y) & ~on(Y) &  
  countofall(X, neighbor(X, Y) & on(X), 3)  
==> on(Y)
```

Example



Demonstration

Tic Tac Toe

States

X		
	O	
		X

```
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

```
legal(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)
```

X		
	O	
		X

Legal Moves:

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

Actions

```
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(1,2,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)
```

mark(1,3)

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

Supporting Concepts

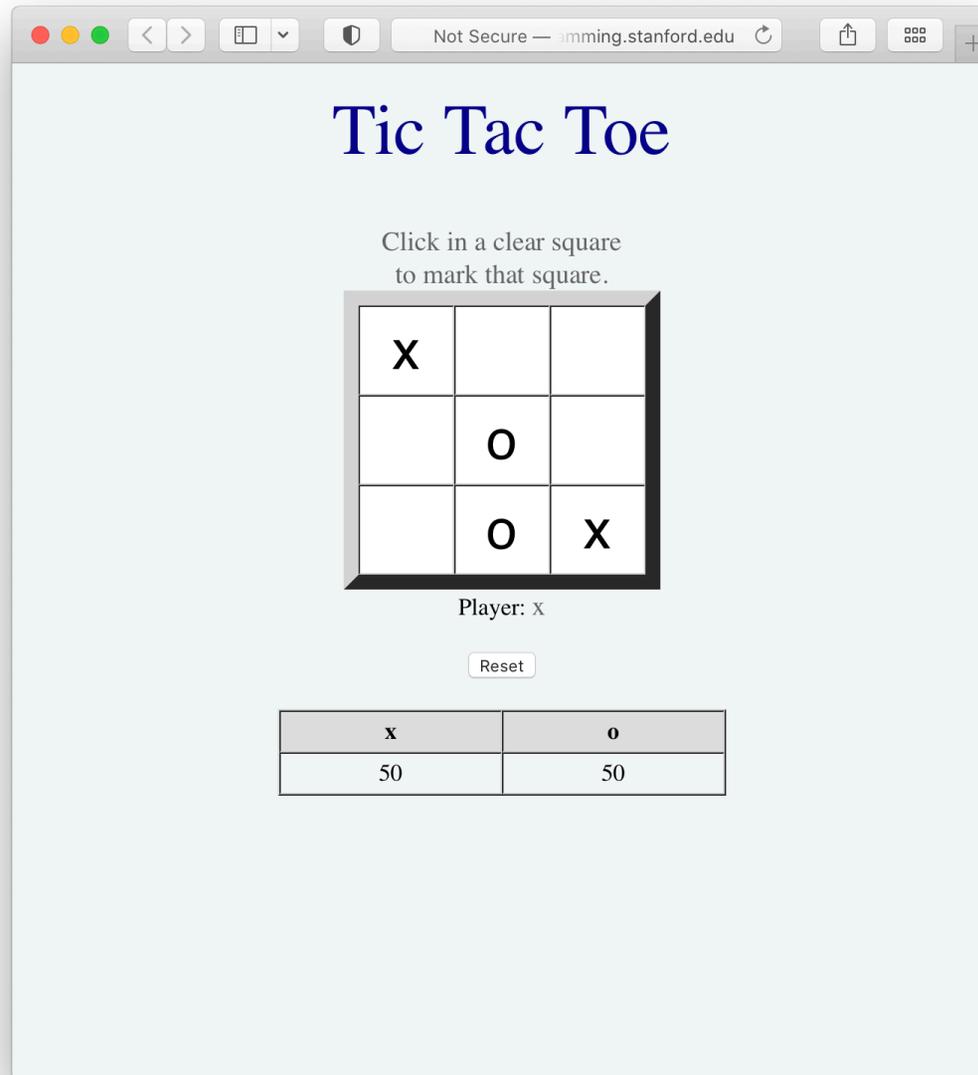
```
row(M,Z) :- cell(M,1,Z) & cell(M,2,Z) & cell(M,3,Z)
col(M,Z) :- cell(1,N,Z) & cell(2,N,Z) & cell(3,N,Z)
diag(Z) :- cell(1,1,Z) & cell(2,2,Z) & cell(3,3,Z)
diag(Z) :- cell(1,3,Z) & cell(2,2,Z) & cell(3,1,Z)

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

win(x) :- line(x)
win(o) :- line(o)

terminal :- win(Z)
terminal :-
    evaluate(countofall([M,N],cell(M,N,b)),0)
```

Example



The screenshot shows a web browser window with the URL "Not Secure — amming.stanford.edu". The page title is "Tic Tac Toe". Below the title, there is a instruction: "Click in a clear square to mark that square." The game board is a 3x3 grid with the following state:

X		
	O	
	O	X

Below the board, it says "Player: x". There is a "Reset" button. At the bottom, there is a score table:

x	o
50	50

Demonstration

Assignments

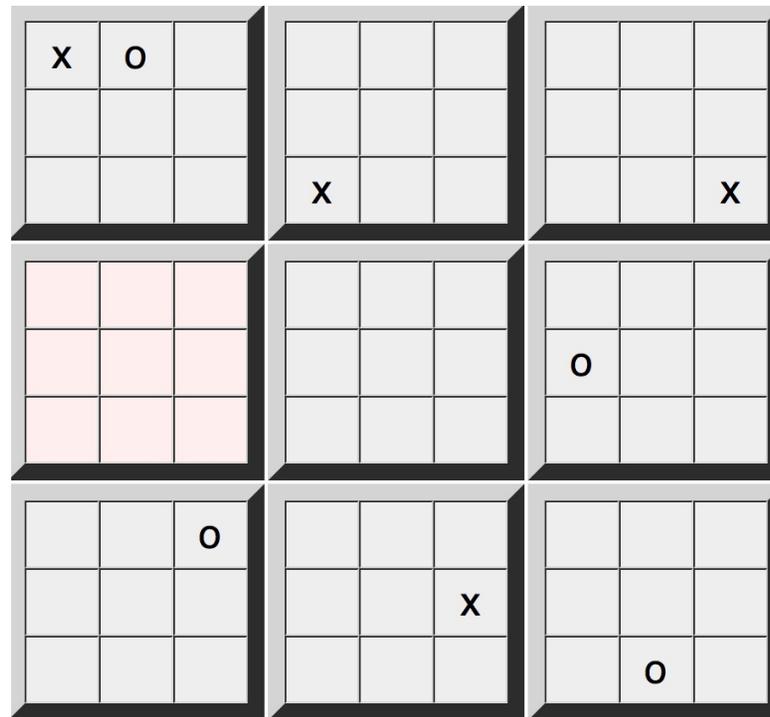
Assignment - Sierra

The goal of this exercise is for you to familiarize yourself with the Sierra capabilities for editing and using action definitions. Go to <http://epilog.stanford.edu> and click on the Sierra link.

In a separate window, open the documentation for Sierra. To access the documentation, go to <http://epilog.stanford.edu>, click on Documentation, and then click on the Sierra item on the resulting drop-down menu.

Read through Section 10 of the documentation and reproduce the examples in the Sierra instance you opened earlier. Once you have done this, experiment on your own. Try different data and different actions.

Assignment - Nineboard Tic Tac Toe



<http://logicprogramming.stanford.edu/assignments/nineboard/editor.html>



Pelican Hunters



0

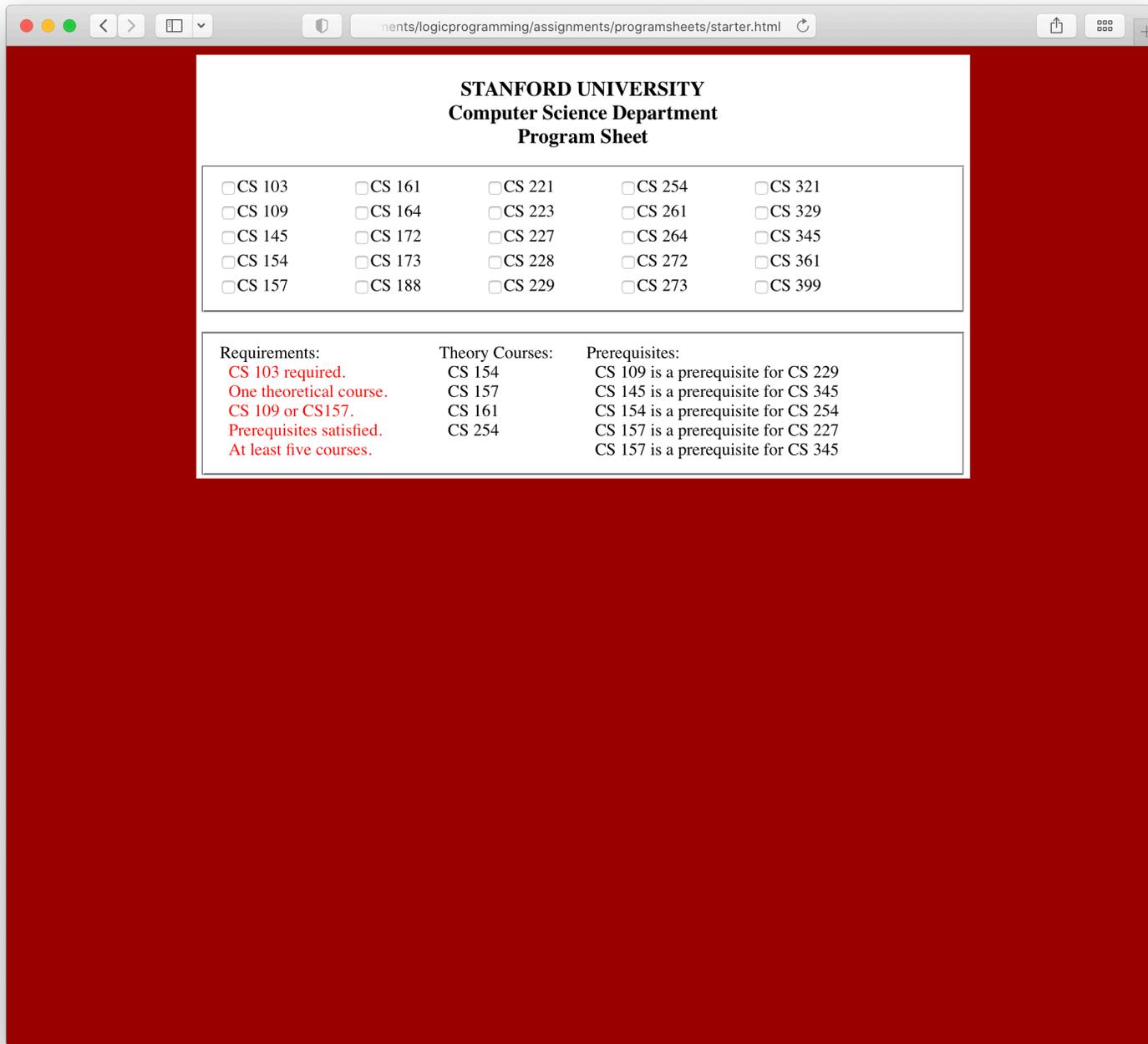
								
								
								
								
								
								

0

0

Player: indy

<http://logicprogramming.stanford.edu/assignments/pelicanhunters/index.html>



<http://logicprogramming.stanford.edu/assignments/programsheets/index.html>

Schedule

Course	Room	Time
cs151	<input type="text"/>	<input type="text"/>
cs157	<input type="text"/>	<input type="text"/>
cs161	<input type="text"/>	<input type="text"/>

Schedule	g100	g200	g300
morning	<input type="text"/>	<input type="text"/>	<input type="text"/>
afternoon	<input type="text"/>	<input type="text"/>	<input type="text"/>
evening	<input type="text"/>	<input type="text"/>	<input type="text"/>

<http://logicprogramming.stanford.edu/assignments/schedule/index.html>

Term Project Proposal

Term Project

