Datasets

\[ p(a, b) \]
\[ p(b, c) \]
\[ p(c, d) \]
Views

g(a, c)
g(b, d)

View

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

\[ p(a, b) \quad p(b, c) \quad p(c, d) \]

\[ p(b, a) \quad p(c, b) \quad p(d, c) \]

\[ g(a, c) \quad g(b, d) \]

\[ g(c, a) \quad g(d, b) \]

**View**

**Operation**

\[ t=1 \]

\[ t=2 \]
View Definitions

\[ r(X, Y) :- p(X, Y) \& \neg q(Y) \]
\[ s(X, Y) :- r(X, Y) \& r(Y, Z) \]

Operation Definitions

\[ \text{flip}(X) :: p(X) \& \neg q(X) \implies \neg p(X) \& q(X) \]
\[ \text{flop}(X) :: r(X, Y) \implies \text{flip}(X) \& \text{flop}(Y) \]
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**
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.
\[ c(a) :: p(a, b) \land q(a) \implies \neg q(a) \land c(b) \]

\[ \begin{align*}
\text{head} & \quad \text{conditions} \quad \text{effects} \\
\text{(action)} & \quad \text{(ordinary literals)} \quad \text{(base literals or actions)}
\end{align*} \]
Variables

\[ c(X) :: p(X,Y) \land q(X) \Rightarrow \neg q(X) \land c(Y) \]
A operation rule is **safe** if and only if every variable in every literal on the right hand side appears 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**

\[
\text{c}(X) ::
\]

\[
p(X,Y) \land \neg q(X) \Rightarrow
\]

\[
\neg p(X,Y) \land q(X) \land c(Y)
\]

**Unsafe Operation Rule**

\[
\text{c}(X) ::
\]

\[
p(X,Y) \land \neg q(Z) \Rightarrow
\]

\[
\neg p(X,Y) \land q(W) \land c(Y)
\]
Degenerate Rule

\[ c(X) :: \text{true} \implies \neg p(X) \land q(X) \]

Shorthand

\[ c(X) :: \neg p(X) \land q(X) \]
An *operation definition* is a finite collection of operation rules with the same operation in the head.

**Example**

\[
\begin{align*}
c(X) &:: p(X) \land q(X) \\
c(X) &:: \neg r(X) \implies \neg p(X) \land r(X)
\end{align*}
\]

A *dynamic logic program* is a collection of view definitions and operation definitions.
Semantics
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 $\Omega$ with dataset $\Delta$ and a set $\Gamma$ of actions, an instance of an operation rule in $\Omega$ is active if and only if (1) the head of the rule is in $\Gamma$ and (2) the conditions of the rule are all true in $\Delta$. Otherwise, the instance is inactive.
**Data:** \( p(a), p(b), p(c), q(a), q(b), q(c), r(b) \)

**Rule:**
\[
u(X) :: p(X) \land q(X) \land \neg r(X) \implies \neg p(X) \land r(X)
\]

**Action:** \( u(a) \)

**Active Instance:**
\[
u(a) :: p(a) \land q(a) \land \neg r(a) \implies \neg p(a) \land r(a)
\]

**Inactive Instances:**
\[
u(b) :: p(b) \land q(b) \land \neg r(b) \implies \neg p(b) \land r(b)
u(c) :: p(c) \land q(c) \land \neg r(c) \implies \neg p(c) \land r(c)
\]
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.

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

Rule:

\[ u(X) : \iff p(X) \land q(X) \land \neg r(X) \implies \neg p(X) \land r(X) \]

Action: \( u(a) \)

Active Instance:

\[ u(a) : \iff p(a) \land q(a) \land \neg r(a) \implies \neg p(a) \land r(a) \]

Expansion: \( \neg p(a), r(a) \)

Negative Update: \( p(a) \)

Positive Update: \( r(a) \)
Given a rule set, the **result** of applying an action set to dataset $\Delta$ is the set consisting of all factoids in $\Delta$ *minus* the negative updates *plus* the positive updates.

$$\Delta - \text{negatives} \bigcup \text{positives}$$
Data: $p(a), p(b), p(c), q(a), q(b), q(c), r(b)$

Rule:

$u(X) :: p(X) \& q(X) \& \neg r(X) \Rightarrow \neg p(X) \& r(X)$

Action: $u(a)$

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

Negative Updates: $p(a)$

Positive Updates: $r(a)$

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

Rule:
\[
u(X) :: p(X) \land q(X) \land \neg r(X) \implies \neg p(X) \land r(X)
\]

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

Active Instances:
\[
u(a) :: p(a) \land q(a) \land \neg r(a) \implies \neg p(a) \land r(a)
u(c) :: p(c) \land q(c) \land \neg r(c) \implies \neg p(c) \land r(c)
\]

Inactive Instance:
\[
u(b) :: p(b) \land q(b) \land \neg r(b) \implies \neg p(b) \land r(b)
\]
Simultaneous Actions

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

Rule:
\[ u(X) ::= p(X) \land q(X) \land \neg r(X) \implies \neg p(X) \land r(X) \]

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

Expansion: \( \neg p(a), \neg p(c), r(a), r(c) \)

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

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

Result: \( p(b), q(a), q(b), q(c), r(a), r(b), r(c) \)
Given a rule set $\Omega$ and a dataset $\Delta$ a set $\Gamma$ of actions, consider the following series.

$\Gamma_0 = \Gamma$

$\Gamma_{n+1}$ = the set of all effects of $\Gamma$ in any active rule instance

The **expansion**$^*$ of $\Gamma$ with respect to $\Omega$ and $\Delta$ 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
Data: $p(a), p(b), p(c), q(a), q(b), q(c), r(b)$

Rule:

$$u(X) :: p(X) \land q(X) \implies \neg p(X) \land r(X) \land u(c)$$

Action: $u(a)$

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

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

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

Result: $p(b), q(a), q(b), q(c), r(a), r(b), r(c)$
function interchange ()
    {x = y;
     y = x}

[x, y]
[3, 4]

interchange()

[x, y]
[4, 4]

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

\[
\text{tick :: } p(X) \implies \neg p(X) \land q(X) \\
\text{tick :: } q(X) \implies \neg q(X) \land p(X)
\]

Before: \{p(a), q(b)\}
After: \{p(b), q(a)\}
Blocks World
Blocks World
Describing States

\[ \text{clear}(a) \]
\[ \text{on}(a,b) \]
\[ \text{on}(b,c) \]
\[ \text{on}(d,e) \]
\[ \ldots \]

\[ \text{clear}(a) \]
\[ \text{table}(a) \]
\[ \text{clear}(b) \]
\[ \text{on}(b,c) \]
\[ \text{on}(d,e) \]
\[ \ldots \]
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:

\( u(X,Y) ::
\)
\[ \text{clear}(X) \land \text{on}(X,Y) \]
\[ \implies \neg \text{on}(X,Y) \land \text{table}(X) \land \text{clear}(Y) \]

\( s(X,Y) ::
\)
\[ \text{table}(X) \land \text{clear}(X) \land \text{clear}(Y) \]
\[ \implies \neg \text{table}(X) \land \neg \text{clear}(Y) \land \text{on}(X,Y) \]
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:
   \( u(X,Y) ::
       \text{clear}(X) \& \text{on}(X,Y)
       \Rightarrow \neg \text{on}(X,Y) \& \text{table}(X) \& \text{clear}(Y) \)
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).
Symbols: $c_{11}, c_{12}, \ldots$

Unary Predicates:
  on - cell is live
  cell - true of cells

Binary Predicates:
  neighbor - cells are neighbors
Any live cell with fewer than two live neighbors dies.

tick ::
on(Y) & evaluate(countofall(X,neighbor(X,Y)&on(X)),0)
==> ~on(Y)

tick ::
on(Y) & evaluate(countofall(X,neighbor(X,Y)&on(X)),1)
==> ~on(Y)
Overcrowding

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

\[
\text{tick} :: \\
\quad \text{on}(Y) \ & \\
\quad \text{evaluate}(\min(\text{countofall}(X, \text{neighbor}(X,Y) \& \text{on}(X)), 4) , 4) \\
\Rightarrow \neg \text{on}(Y)
\]
Any *dead* cell with *exactly three* live neighbors becomes live.

\[
\text{tick :: cell(Y) \& \sim on(Y) \&
\text{evaluate}(\text{countofall}(X, \text{neighbor}(X,Y) \& on(X)),3)
\Rightarrow on(Y)}
\]
http://logicprogramming.stanford.edu/examples/gameoflife.html
Tic Tac Toe
States

\[
\begin{array}{ccc}
  & X & \\
X & & O \\
  & X &
\end{array}
\]

- 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(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)
Actions

\[
\text{mark}(M,N) ::
\]
\[
\text{control}(Z) \implies \neg \text{cell}(M,N,b) \land \text{cell}(M,N,Z)
\]
\[
\text{mark}(M,N) ::
\]
\[
\text{control}(x) \implies \neg \text{control}(x) \land \text{control}(o)
\]
\[
\text{mark}(M,N) ::
\]
\[
\text{control}(o) \implies \neg \text{control}(o) \land \text{control}(x)
\]

\[
\begin{array}{c}
\text{cell}(1,1,x) \\
\text{cell}(1,2,b) \\
\text{cell}(1,3,b) \\
\text{cell}(2,1,b) \\
\text{cell}(2,2,o) \\
\text{cell}(2,3,b) \\
\text{cell}(3,1,b) \\
\text{cell}(3,2,b) \\
\text{cell}(3,3,x) \\
\text{control}(o)
\end{array}
\]

\[
\begin{array}{c}
\text{cell}(1,1,x) \\
\text{cell}(1,2,b) \\
\text{cell}(1,3,o) \\
\text{cell}(2,1,b) \\
\text{cell}(2,2,o) \\
\text{cell}(2,3,b) \\
\text{cell}(3,1,b) \\
\text{cell}(3,2,b) \\
\text{cell}(3,3,x) \\
\text{control}(x)
\end{array}
\]

mark(1,3)
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)
goals and termination

\[
\text{win}(x) \, \text{:-} \, \text{line}(x)
\]

\[
\text{win}(o) \, \text{:-} \, \text{line}(o)
\]

\[
\text{terminal} \, \text{:-} \, \text{win}(Z)
\]

\[
\text{terminal} \, \text{:-}
\]

\[
\text{evaluate}(\text{countofall}([M,N], \text{cell}(M,N,b)), 0)
\]
Assignments
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 though Sections 7 and 8 of the documentation and reproduce the examples in the Sierra window you opened earlier. Once you have done this, experiment on your own. Try different data and different actions.
Pelican Hunters

http://logicprogramming.stanford.edu/assignments/pelican_overview.html
### Schedule

<table>
<thead>
<tr>
<th>Course</th>
<th>Room</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs151</td>
<td>✿</td>
<td>✿</td>
</tr>
<tr>
<td>cs157</td>
<td>✿</td>
<td>✿</td>
</tr>
<tr>
<td>cs161</td>
<td>✿</td>
<td>✿</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schedule</th>
<th>g100</th>
<th>g200</th>
<th>g300</th>
</tr>
</thead>
<tbody>
<tr>
<td>morning</td>
<td>✿</td>
<td>✿</td>
<td>✿</td>
</tr>
<tr>
<td>afternoon</td>
<td>✿</td>
<td>✿</td>
<td>✿</td>
</tr>
<tr>
<td>evening</td>
<td>✿</td>
<td>✿</td>
<td>✿</td>
</tr>
</tbody>
</table>