Queries and View Definitions

\begin{align*}
goal(X,Y) & : = p(X,Y) \land \neg q(Y) \\
r(X,Y) & : = p(X,Y) \land \neg q(Y) \\
s(X,Y) & : = r(X,Y) \land r(Y,Z)
\end{align*}

Updates and Operation Definitions

\begin{align*}
p(X) \land \neg q(X) & \implies \neg p(X) \land q(X) \\
\text{flip}(X) & : : p(X) \land \neg q(X) \implies \neg p(X) \land q(X) \\
\text{flop}(X) & : : r(X,Y) \implies \text{flip}(X) \land \text{flop}(Y)
\end{align*}
Syntax
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.
Operation Definition

\[
c(a) :: p(a,b) \land q(a) \implies \neg q(a) \land c(b)
\]

- **head**
- **conditions**
- **effects**

*(action) (ordinary literals) (base literals or actions)*
\[ c(X) :: p(X,Y) \land q(X) \implies \neg q(X) \land c(Y) \]
**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

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

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

\[
\begin{align*}
\text{c}(X) &::: \\
\text{c}(X) &::: \\
\text{p}(X,Y) &\& \text{~q}(X) \implies \\
\text{~p}(X,Y) &\& \text{q}(X) &\& \text{c}(Y)
\end{align*}
\]

**Unsafe Operation Rule**

\[
\begin{align*}
\text{c}(X) &::: \\
\text{p}(X,Y) &\& \text{~q}(Z) \implies \\
\text{~p}(X,Y) &\& \text{q}(W) &\& \text{c}(Y)
\end{align*}
\]
Semantics
Given a rule set, the result of applying an action to a dataset is the dataset that results from *deleting all of the negative effects* of the action from the dataset and *adding in all of the positive effects*. 
Dataset: \{p(a), p(b), p(c), q(a), q(b), q(c), r(b)\}

Ruleset:

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

Actionset: \{u(a), u(b)\}

Extension: \{p(a), p(b), p(c), q(a), q(b), q(c), r(b), s(a), s(b), s(c)\}

Active Instances of operation rule:

\[
\begin{align*}
u(a) & : : s(a) \land \neg r(a) \implies \neg p(a) \land r(a)
\end{align*}
\]

Inactive Instance:

\[
\begin{align*}
u(b) & : : s(b) \land \neg r(b) \implies \neg p(b) \land r(b) \\
u(c) & : : s(a) \land \neg r(c) \implies \neg p(c) \land r(c)
\end{align*}
\]
Dataset: \{p(a), p(b), p(c), q(a), q(b), q(c), r(b)\}

Ruleset:
\[
\begin{align*}
  s(X) & : - p(X) & q(X) \\
  u(X) & : s(X) & \neg r(X) \implies \neg p(X) & r(X)
\end{align*}
\]

Actionset: \{u(a), u(b)\}

Extension: \{p(a), p(b), p(c), q(a), q(b), q(c), r(b), s(a), s(b), s(c)\}

Active Instances of operation rule:
\[
\begin{align*}
  u(a) & : s(a) & \neg r(a) \implies \neg p(a) & r(a)
\end{align*}
\]

Inactive Instance:
\[
\begin{align*}
  u(b) & : s(b) & \neg r(b) \implies \neg p(b) & r(b) \\
  u(c) & : s(a) & \neg r(c) \implies \neg p(c) & r(c)
\end{align*}
\]
Dataset: \{p(a), p(b), p(c), q(a), q(b), q(c), r(b)\}

Ruleset:
\[ s(X) :\!\!\!\!: p(X) \& q(X) \]
\[ u(X) :\!\!\!\!: s(X) \& \neg r(X) \Rightarrow \neg p(X) \& r(X) \]

Actionset: \{u(a), u(b)\}

Active Instances of operation rules:
\[ u(a) :\!\!\!\!: s(a) \& \neg r(a) \Rightarrow \neg p(a) \& r(a) \]

Expansion: \{\neg p(a), r(a)\}
Negative Updates: \{p(a)\}
Positive Updates: \{r(a)\}
Dataset: \{p(a), p(b), p(c), q(a), q(b), q(c), r(b)\}

Ruleset:
- \(s(X) \leftarrow p(X) \land q(X)\)
- \(u(X) :: s(X) \land \neg r(X) \Rightarrow \neg p(X) \land r(X)\)

Actionset: \{u(a), u(b)\}

Negative Updates: \{p(a)\}
Positive Updates: \{r(a)\}

Result: \{p(b), p(c), q(a), q(b), q(c), r(a), r(b)\}
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:** \( \{p(a),q(b)\} \text{ or } \{p(b),q(a)\} \)

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.

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

Before: \{p(a), q(b)\}
After: \{p(b), q(a)\}
Blocks World
Blocks World
External Actions

\[ u(a,b) \]

\[ u(d,e) \]
Describing States

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

\[ u(a,b) \]

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

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

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

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

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

\[
\text{tick} :: \\
on(Y) \& \text{evaluate(countofall}(X, \text{neighbor}(X,Y) \& \text{on}(X)), 0) \implies \neg \text{on}(Y)
\]

\[
\text{tick} :: \\
on(Y) \& \text{evaluate(countofall}(X, \text{neighbor}(X,Y) \& \text{on}(X)), 1) \implies \neg \text{on}(Y)
\]
Overcrowding

Any live cell with more than three live neighbors dies.

tick ::
  on(Y) &
  evaluate(min(countofall(X,neighbor(X,Y)&on(X)),4),4) ==>
=> ~on(Y)
Any *dead* cell with *exactly three* live neighbors becomes live.

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

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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

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

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

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

\[ \text{terminal} \text{ :- win(Z)} \]
\[ \text{terminal} \text{ :- evaluate(countofall([M,N],cell(M,N,b)),0)} \]
Lambda
edge(a,b)
edge(b,d)
edge(b,e)

Execute
Action
Expand
Expand on update
Run on clock tick
Execute

Library
copy(X,Y) :: edge(X,Z) => edge(Y,Z)
invert(X) :: edge(X,Y) => ¬edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) => insert(X,Z)
edge(a,b)
edge(b,d)
edge(b,e)

copy(b,c)

data

edge(c,d)
edge(c,e)

copy(X,Y) :: edge(X,Z) => edge(Y,Z)

invert(X) :: edge(X,Y) => ¬edge(X,Y) & edge(Y,X)

insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) => insert(X,Z)

Display a menu
Lambda

edge(a,b)
edge(b,d)
edge(b,e)
edge(c,d)
edge(c,e)

Library

copy(X,Y) :: edge(X,Z) => edge(Y,Z)
invert(X) :: edge(X,Y) => ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) => insert(X,Z)
Lambda

- edge(a, b)
- edge(b, d)
- edge(b, e)
- edge(c, d)
- edge(c, e)

Execute

Action: invert(c)

- edge(c, d)
- edge(d, c)
- ~edge(c, e)
- edge(e, c)

Library

- copy(X, Y) :: edge(X, Z) => edge(Y, Z)
- invert(X) :: edge(X, Y) => ~edge(X, Y) & edge(Y, X)
- insert(X, Y) :: edge(X, Y)
- insert(X, Y) :: edge(Y, Z) => insert(X, Z)
Lambda
edge(a,b)
edge(b,d)
edge(b,e)
edge(d,c)
edge(e,c)

Library

copy(X,Y) :: edge(X,Z) => edge(Y,Z)

invert(X) :: edge(X,Y) => ¬edge(X,Y) & edge(Y,X)

insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) => insert(X,Z)
Lambda

edge(a, b)
edge(b, d)
edge(b, e)
edge(d, c)
edge(e, c)

Execute

Action: insert(w, b)

Expand

Expand on update

Run on clock tick

edge(w, b)
edge(w, d)
edge(w, c)
edge(w, e)

Library

save
revert

\text{copy}(X, Y) :: edge(X, Z) \implies edge(Y, Z)

\text{invert}(X) :: edge(X, Y) \implies \neg\text{edge}(X, Y) \land \text{edge}(Y, X)

\text{insert}(X, Y) :: edge(X, Y)
\text{insert}(X, Y) :: edge(Y, Z) \implies \text{insert}(X, Z)
Lambda

edge(a,b)
edge(b,d)
edge(b,e)
edge(d,c)
edge(e,c)
edge(w,b)
edge(w,d)
edge(w,c)
edge(w,e)

Execute

Action: insert(w,b)
Expand on update
Run on clock tick

Library

`copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)`
`invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)`
`insert(X,Y) :: edge(X,Y)`
`insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)`

Display a menu
Assignments
Reading 4.1 - Dynamics
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 section 7 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.
Assignment - Nineboard Tic Tac Toe

http://complaw.stanford.edu/chapters/nineboard.html
Pelican Hunters

http://complaw.stanford.edu/assignments/pelican_basic.html
Legal Empowerment through Information Technology