Logic Programming

Operation Definitions

Michael Genesereth
Computer Science Department
Stanford University
Queries and View Definitions

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

Updates and Operation Definitions

\[
p(X) \ & \ \neg q(X) \implies \ \neg p(X) \ & \ q(X)
\]
\[
\text{flip}(X) :: p(X) \ & \ \neg q(X) \implies \ \neg p(X) \ & \ q(X) \\
\text{invert}(X,Y) :: \text{flip}(X) \ & \ \text{flip}(Y)
\]
Syntax
Operation constants represent changes.

invert - turn block upside down
stack - put one block on another
move - move a block from one block to another

Same spelling conventions as constructors. Like constructors, each has a specific arity.
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 and separated by commas.

**Examples:**
- invert(a)
- stack(a,b)
- move(a,b,c)

Syntactically, actions are treated as terms.
click(a) :: p(a,b) & q(a) ==> ~q(a) & click(b)

head        conditions        effects
Example

\[\text{click}(X) :: p(X,Y) \land q(X) \Rightarrow \neg q(X) \land \text{click}(Y)\]
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**

\[
\text{click}(X) :: \\
p(X,Y) \land \neg q(X) \implies \\
\neg p(X,Y) \land q(X) \land \text{click}(Y)
\]

**Unsafe Operation Rule**

\[
\text{click}(X) :: \\
p(X,Y) \land \neg q(Z) \implies \\
\neg p(X,Y) \land q(W) \land \text{click}(Y)
\]
Degenerate Rule

click(X) :: true ==> \neg p(X) \& q(X)

Shorthand

click(X) :: \neg p(X) \& q(X)
An operation definition is a finite collection of operation rules.

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

NB: In purposes of analysis, we sometimes talk about infinite operation definitions.
Semantics
Recall that $E(\Omega, \Delta)$ is the extension of dataset $\Delta$ with respect to ruleset $\Omega$.

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 the head of the rule is in $\Gamma$ and the conditions of the rule are all true in $\Delta$. Otherwise, the instance is **inactive**.
The **expansion** of an action set $\Gamma$ with respect to rule set $\Omega$ and dataset $\Delta$ (written $U(\Gamma, \Omega, \Delta)$) is defined as follows.

$\Gamma_0 = \Gamma$

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

$U(\Gamma, \Omega, \Delta)$ is the fixpoint of this series.

The **positive updates** (written $A(\Gamma, \Omega, \Delta)$) are the positive base factoids in $U(\Gamma, \Omega, \Delta)$. The **negative updates** (written $D(\Gamma, \Omega, \Delta)$) are the negative base factoids in $U(\Gamma, \Omega, \Delta)$.
The update $u(\Gamma,\Omega,\Delta)$ of $\Delta$ with respect to $\Omega$ and $\Gamma$ is the set consisting of all factoids in $\Delta$ minus those in $D(\Gamma,\Omega,\Delta)$ plus those in $A(\Gamma,\Omega,\Delta)$.

$$u(\Gamma,\Omega,\Delta) = \Delta - D(\gamma,\Omega,\Delta) \cup A(\gamma,\Omega,\Delta)$$

**Dataset:** $\{p(a), p(b), p(c), q(a), q(b), q(c), r(b)\}$

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

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

**Result:** $\{p(b), q(a), q(b), q(c), r(a), r(b), r(c)\}$
Example

Ruleset

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

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

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

Active Instances

\[ s(a) \land \neg r(a) \implies \neg p(a) \land r(a) \]
\[ s(c) \land \neg r(c) \implies \neg p(c) \land r(c) \]

Inactive Instance

\[ s(b) \land \neg r(b) \implies \neg p(b) \land r(b) \]
Ruleset

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

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

Actions: \{u(a), u(c)\}

Active Instances

\[ u(a) :: s(a) \land \neg r(a) \implies \neg p(a) \land r(a) \]
\[ u(c) :: s(c) \land \neg r(c) \implies \neg p(c) \land r(c) \]

Negative Update: \{p(a), p(c)\}
Positive Update: \{r(a), r(c)\}
Dataset:
edge(a,b)
edge(b,d)
edge(b,e)

Operations:
copy(X,Y) - copy outgoing links from X to Y.
invert(X) - reverse the incoming arcs to X.
insert(X,Y) - add arc from X to Y and all nodes succs.
Example

Dataset:

\{\text{edge}(a,b), \text{edge}(b,d), \text{edge}(b,e)\}

Operation Definition:

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

Action: \text{copy}(b,c)

Positive Updates: \{\text{edge}(c,d), \text{edge}(c,e)\}

Negative Updates: {}

Result:

\{\text{edge}(a,b), \text{edge}(b,d), \text{edge}(b,e), \text{edge}(c,d), \text{edge}(c,e)\}
Example

Dataset:

\{\text{edge}(a,b), \text{edge}(b,d), \text{edge}(b,d)\}

Operation Definition:

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

Action: \text{invert}(c)

Positive Updates: \{\text{edge}(d,c), \text{edge}(e,c)\}

Negative Updates: \{\text{edge}(c,d), \text{edge}(c,e)\}

Result:

\{\text{edge}(a,b), \text{edge}(b,d), \text{edge}(b,e), \text{edge}(d,c), \text{edge}(e,c)\}
Dataset:
\{\text{edge}(a,b), \text{edge}(b,d), \text{edge}(b,e), \text{edge}(d,c), \text{edge}(e,c)\}

Operation Definition:
\begin{align*}
\text{insert}(X,Y) & :: \text{edge}(X,Y) \\
\text{insert}(X,Y) & :: \text{edge}(Y,Z) \Rightarrow \text{insert}(X,Z)
\end{align*}

Action: \text{insert}(w,b)

Expansion:
\{\text{insert}(w,b), \text{edge}(w,b), \text{insert}(w,d), \text{insert}(w,e), \text{edge}(w,d), \text{edge}(w,e), \text{insert}(w,c), \text{edge}(w,c)\}

Negative Updates: {} 
Positive Updates:
\{\text{edge}(w,b), \text{edge}(w,d), \text{edge}(w,e), \text{edge}(w,c)\}
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.

\[
\text{if } p(X), \text{ then del } p(X) \text{ and add } q(X) \\
\text{if } q(X), \text{ then del } q(X) \text{ and add } p(X)
\]

Before: \{p(a), q(b)\}
Step 1: \{q(a), q(b)\}
Step 2: \{p(a), p(b)\}
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.

\[
\begin{align*}
\text{if } p(X), & \text{ then del } p(X) \text{ and add } q(X) \\
\text{if } q(X), & \text{ then del } q(X) \text{ and add } p(X)
\end{align*}
\]

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

\[ u(a,b) \]

\[ u(d,e) \]
Symbols: a, b, c, d, e

Unary Predicates:
  clear - blocks with no blocks on top
  table - blocks on the table

Binary Predicates:
  on - pairs of blocks in which first is on the second
  above - pairs in which first block is above the second

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.
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 \( y \).

Transition Rules:
  \( u(X, Y) \ & \ clear(X) \ & \ on(X, Y) \)
  \[ \Rightarrow \ & \ ~on(X, Y) \ & \ table(X) \ & \ clear(Y) \]

  \( s(X, Y) \ & \ table(X) \ & \ clear(X) \ & \ clear(Y) \)
  \[ \Rightarrow \ & \ ~table(X) \ & \ ~clear(Y) \ & \ on(X, Y) \]
Example

Dataset:

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

Transition Rule:

\[ u(X,Y) \& clear(X) \& on(X,Y) \implies \neg on(X,Y) \& table(X) \& clear(Y) \]

Result of \( u(c,a) \):

- clear(a)
- on(a,b)
- table(b)
- clear(c)
- on(d,e)
- table(c)
- clear(d)
- table(e)
Example - Tic Tac Toe
Tic Tac Toe

X

O

X
Non-Differential Rules

\[
\text{mark}(J,K) \land \text{cell}(M,N,R) \implies \neg\text{cell}(M,N,R)
\]

\[
\text{mark}(J,K) \land \text{control}(R) \\
\implies \text{cell}(J,K,R)
\]

\[
\text{mark}(J,K) \land \text{cell}(M,N,R) \land \text{distinct}(R,b) \\
\implies \text{cell}(M,N,R)
\]

\[
\text{mark}(J,K) \land \text{cell}(M,N,R) \land \text{distinct}(J,M) \\
\implies \text{cell}(M,N,b)
\]

\[
\text{mark}(J,K) \land \text{cell}(M,N,R) \land \text{distinct}(K,N) \\
\implies \text{cell}(M,N,b)
\]
The Game of Life
(1) Any live cell with \textit{two or three} live neighbors lives on to the next generation.

(2) Any live cell with \textit{fewer than two} live neighbors dies (as if caused by underpopulation).

(3) Any live cell with \textit{more than three} live neighbors dies (as if by overpopulation).

(4) Any dead cell with \textit{exactly three} live neighbors becomes a live cell (as if by reproduction).
Symbols: c11, c12, ...

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

Binary Predicates:
  neighbor - cells are neighbors
  supports - first cell is alive and a neighbor of the second

Definition of support
  supports(X,Y) :- neighbor(X,Y) & on(X)
Transition Rules

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

$$\text{on}(Y) \land \text{countofall}(X, \text{supports}(X,Y), N) \land \text{leq}(N, 1) \implies \sim\text{on}(Y)$$

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

$$\text{on}(Y) \land \text{countofall}(X, \text{supports}(X,Y), N) \land \text{geq}(N, 4) \implies \sim\text{on}(Y)$$

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

$$\text{cell}(Y) \land \sim\text{on}(Y) \land \text{countofall}(X, \text{supports}(X,Y), 3) \implies \text{on}(Y)$$
http://logicprogramming.stanford.edu/examples/gameoflife.html
http://logicprogramming.stanford.edu/examples/gameoflife.html
Sierra
Lambda

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

Lambda

Query

Pattern: goal(X,Z)
Query: \( p(X,Y) \land p(Y,Z) \)

Transform

Condition: \( p(X,Y) \)
Conclusion: \( \neg p(X,Y) \land p(Y,X) \)

Execute: Expand on update
Lambda

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

Query

Pattern: \( \text{goal}(X,Z) \)
Query: \( p(X,Y) \land p(Y,Z) \)

100 result(s) Autorefresh

Transform

Condition: \( p(X,Y) \)
Conclusion: \( \neg p(X,Y) \land p(Y,X) \)

Expand on update

Execute

\[ \neg p(a,b) \\
\[ p(b,a) \\
\[ \neg p(b,c) \\
\[ p(c,b) \\
\[ \neg p(c,d) \\
\[ p(d,c) \\
\[ \neg p(d,e) \\
\[ p(e,d) \]
p(a,b)
p(b,c)
p(c,d)
p(d,e)

Query
Pattern: goal(X,Z)
Query: p(X,Y) & p(Y,Z)

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

Transform
Condition: p(X,Y)
Conclusion: ~p(X,Y) & p(Y,X)

~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,d)