Logic Programming

Operation Definitions

Michael Genesereth
Computer Science Department
Stanford University
Queries and View Definitions

\[
\text{goal}(X,Y) \; :\; \text{p}(X,Y) \; \& \; \neg \text{q}(Y)
\]

\[
\text{r}(X,Y) \; :\; \text{p}(X,Y) \; \& \; \neg \text{q}(Y)
\]

\[
\text{s}(X,Y) \; :\; \text{r}(X,Y) \; \& \; \text{r}(Y,Z)
\]

Updates and Operation Definitions

\[
\text{p}(X) \; \& \; \neg \text{q}(X) \implies \neg \text{p}(X) \; \& \; \text{q}(X)
\]

\[
\text{flip}(X) \; :: \; \text{p}(X) \; \& \; \neg \text{q}(X) \implies \neg \text{p}(X) \; \& \; \text{q}(X)
\]

\[
\text{invert}(X,Y) \; :: \; \text{flip}(X) \; \& \; \text{flip}(Y)
\]
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

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

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

**Unsafe Operation Rule**

\[
\text{click}(X) :: \\
p(X,Y) \land \neg q(Z) ==> \\
\neg p(X,Y) ==> \\
\neg p(X,Y) \land q(W) \land \text{click}(Y)
\]
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**

\[ s(X) :− p(X) \& q(X) \]
\[ s(X) \& \sim r(X) \Rightarrow \sim p(X) \& r(X) \]

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

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

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

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

Inactive Instance

\[ s(b) \land \neg r(b) \Rightarrow \neg p(b) \land r(b) \]
Rule set

\[ s(X) \texttt{ :- } p(X) \land q(X) \]
\[ u(X) \texttt{ :: } s(X) \land \neg r(X) \implies \neg p(X) \land r(X) \]

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

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

Active Instances

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

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.
Dataset:

\{\text{edge}(a,b), \text{edge}(b,d), \text{edge}(b,e)\}\n
Operation Definition:

\text{copy}(X,Y) ::= \text{edge}(X,Z) \implies \text{edge}(Y,Z)\n
Action: \text{copy}(b,c)\n
Positive Updates: \{\text{edge}(c,d), \text{edge}(c,e)\}\n
Negative Updates: {}\n
Result:

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

Operation Definition:
invert(Y) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)

Action: invert(c)

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

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

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) \implies \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.

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), 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.

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)\}  
After:  \{p(b), q(a)\}
Sierra
Lambda

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

Query

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

- Goal: \( \text{goal}(a, c) \)
  \( \text{goal}(b, d) \)
  \( \text{goal}(c, e) \)

Transform

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

- Expand on update: On
- Run on clock tick: Off
Lambda
p(a, b)
p(b, c)
p(c, d)
p(d, e)

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

100 result(s)  ✓  Autorefresh

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

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

Expand ➤  ✓  Expand on update
Execute ➤  Run on clock tick

-p(a, b)
p(b, a)
-p(b, c)
p(c, b)
-p(c, d)
p(d, c)
-p(d, e)
p(e, d)
Lambda

\begin{align*}
p(a, b) \\
p(b, c) \\
p(c, d) \\
p(d, e)
\end{align*}

Query

Pattern: \textit{goal}(X, Z)

Query: \textit{p}(X, Y) \& \textit{p}(Y, Z)

Goal:
- \textit{goal}(a, c)
- \textit{goal}(b, d)
- \textit{goal}(c, e)

Transform

Condition: \textit{p}(X, Y)

Conclusion: \neg \textit{p}(X, Y) \& \textit{p}(Y, X)

- \textit{p}(a, b)
- \textit{p}(b, a)
- \neg \textit{p}(b, c)
- \textit{p}(c, b)
- \neg \textit{p}(c, d)
- \textit{p}(d, c)
- \neg \textit{p}(d, e)
- \textit{p}(e, d)