Logic Programs with Preferences

REGULAR Logic Program

penguin(tweety) ←
bird(tweety) ←
flies(tweety) ← bird(tweety), not ¬flies(tweety).
¬flies(tweety) ← penguin(tweety), not flies(tweety).

Answer Set 1:
{penguin(tweety), bird(tweety), flies(tweety)}

Answer Set 2:
{penguin(tweety), bird(tweety), ¬flies(tweety)}

ORDERED Logic Program

penguin(tweety) ←
bird(tweety) ←
r1: flies(tweety) ← bird(tweety), not ¬flies(tweety).
r2: ¬flies(tweety) ← penguin(tweety), not flies(tweety).

$n_{r1} << n_{r2} \leftarrow$

UNIQUE Answer Set:
{penguin(tweety), bird(tweety), ¬flies(tweety)}

Definitions

A Logic Program Π over $\mathscr{L}$ is *ordered* if $\mathscr{L}$ is partitioned in:

- a set A of regular atoms;
- a set $\aleph$ of terms used as *names* for rules;
- a set $A_{<<}$ of *preference atoms* "s << t" where $s,t \in \aleph$.

The answer sets of an ordered LP are called *preferred* answer sets.

Naming function

$$n: \Pi \rightarrow \aleph$$

Statically Ordered logic program Π: (*meta−level* perferences)

$\Pi=\Pi' \cup \Pi''$, where:

- Π' is a logic program over $\mathscr{L} \setminus A_{<<}$, and
- $\Pi'' \subseteq \{(n(r) << n(r')) \leftarrow \mid r,r' \in \Pi\}$.

Compiling Preferences

Mapping Function T from Ordered LP, Π, to Regular LP:

a rule can be "applied" only if this is compatible with all preferences.

The application of a rule, r, is *ok* w.r.t. a rule, r', with higher preference if:

- r' was *applied*, or
- r' was *blocked*.

For any rule, $r \in \Pi$, $\tau(r)$ is defined as follows, and $T(\Pi)=\{\tau(r) \mid r \in \Pi\}$

a1(r): head(r) $\leftarrow$ ap(n(r)).
a2(r): ap(n(r)) $\leftarrow$ ok(n(r)), body(r).
b1(r,L): bl(n(r)) $\leftarrow$ ok(n(r)), not L^+.
b2(r,L): bl(n(r)) $\leftarrow$ ok(n(r)), L^-.

c1(r): ok(n(r)) $\leftarrow$ ok'(n(r),n(r_1)),...,ok'(n(r),n(r_k)).
c2(r,r'): ok'(n(r),n(r')) $\leftarrow$ not (n(r) << n(r')).
c3(r,r'): ok'(n(r),n(r')) $\leftarrow$ (n(r) << n(r')), ap(n(r')).
c4(r,r'): ok'(n(r),n(r')) $\leftarrow$ (n(r) << n(r')), bl(n(r')).

t(r,r',r''): n(r)<<n(r'') $\leftarrow$ (n(r)<<n(r')), (n(r')<<n(r'')).
as(r,r'): $\neg$(n(r')<<n(r)) $\leftarrow$ n(r)<<n(r').

Example

penguin(tweety) ←
bird(tweety) ←
r1: flies(tweety) ← bird(tweety), not ¬flies(tweety).
r2: ¬flies(tweety) ← penguin(tweety), not flies(tweety).
$n_{r1} << n_{r2}$ ←

flies(tweety) ← ap(n(r1)).
ap(n(r1)) ← ok(n(r1)), bird(tweety), not ¬flies(tweety).
bl(n(r1)) ← ok(n(r1)), not bird(tweety).
bl(n(r1)) ← ok(n(r1)), ¬flies(tweety).

ok(n(r1)) ← ok'(n(r1),n(r2)).
ok'(n(r1),n(r2)) ← not (n(r1)<<n(r2)).
ok'(n(r1),n(r2)) ← (n(r1)<<n(r2)), ap(n(r2)).
ok'(n(r1),n(r2)) ← (n(r1)<<n(r2)), bl(n(r2)).

Going to the Airport (1)

John can go to the airport either by car or by bus:

- two sets of actions: car−related actions and bus−related actions;
- initial state: at−home;
- final state: at−airport.

If it is possible to go to the airport both by car and by bus, John prefers the car.

Going to the Airport (2)

```
% initial state
h(at−home,0).

% goal
goal(T) :− time(T), h(at−airport,T).
:− not goal(lasttime).

occurs(A,T) :−      time(T), car_action(A),
                    going_by_car,
                    not goal(T),
                    not other_action(A,T).

occurs(A,T) :−      time(T), bus_action(A),
                    going_by_bus,
                    not goal(T),
                    not other_action(A,T).

% choice between car & bus
r1:  going_by_car :− not going_by_bus.
r2:  going_by_bus :− not going_by_car.

% preference
n(r2) << n(r1).
```

Going to the Airport (3)

Suppose that *normally* John prefers to use the car, but that he prefers the bus when it's snowing.

```
% initial state
h(at−home,0).

% goal
goal(T) :− time(T), h(at−airport,T).
:− not goal(lasttime).

occurs(A,T) :−      time(T), car_action(A),
                    going_by_car,
                    not goal(T),
                    not other_action(A,T).

occurs(A,T) :−      time(T), bus_action(A),
                    going_by_bus,
                    not goal(T),
                    not other_action(A,T).

% choice between car & bus
r1:  going_by_car  :− not going_by_bus.
r2:  going_by_bus  :− not going_by_car.

% preference
n(r2) << n(r1)      :− not ab.
ab                  :− n(r1) << n(r2).
n(r1) << n(r2)      :− snowing.
```

References

James P. Delgrande, Torsten Schaub, Hans Tompits
Logic Programs with Compiled Preferences

James P. Delgrande, Torsten Schaub, Hans Tompits
plp − A compiler for logic programs with preferences