**REGULAR Logic Program**

\[
penguin(tweety) \leftarrow \\
bird(tweety) \leftarrow \\
\text{flies}(tweety) \leftarrow \text{bird}(tweety), \neg \text{flies}(tweety). \\
\neg \text{flies}(tweety) \leftarrow \text{penguin}(tweety), \neg \text{flies}(tweety). \\
\]

**ORDERED Logic Program**

\[
penguin(tweety) \leftarrow \\
bird(tweety) \leftarrow \\
r1: \text{flies}(tweety) \leftarrow \text{bird}(tweety), \neg \text{flies}(tweety). \\
r2: \neg \text{flies}(tweety) \leftarrow \text{penguin}(tweety), \neg \text{flies}(tweety). \\
\]

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

**Answer Set 2:**
\{penguin(tweety), bird(tweety), \neg flies(tweety)}

**UNIQUE Answer Set:**
\{penguin(tweety), bird(tweety), \neg flies(tweety)}
A logic program $\Pi$ over $\mathcal{A}$ is ordered if $\mathcal{A}$ is partitioned into:

- A set $A$ of regular atoms;
- A set $\mathcal{E}$ of terms used as names for rules;
- A set $\mathcal{F}$ of preference atoms "subsumes" $\mathcal{E}$, where $s,t \in \mathcal{E}$.

Statically Ordered Logic Program $\Pi$: (meta-level preferences)

\[
\Pi = \Pi' \cup \Pi''\]

where:

- $\Pi$ is a logic program over $\mathcal{A}$;
- $\Pi' \subseteq \{ (n(r) \gg (n(r')) \mid r,r' \in \Pi) \}$.

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

Naming function $n: \Pi \rightarrow \mathcal{E}$.
Mapping Function $T$ from Ordered LP, $\Pi$, to Regular LP:

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

The application of a rule, $r$, is $\text{ok}$ 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 $T(\Pi) = \{ \tau(r) | r \in \Pi \}$

\begin{align*}
a1(r): \quad & \text{head}(r) \leftarrow \text{ap}(n(r)). \\
a2(r): \quad & \text{ap}(n(r)) \leftarrow \text{ok}(n(r)), \text{body}(r). \\
b1(r,L): \quad & \text{bl}(n(r)) \leftarrow \text{ok}(n(r)), \text{not } L^+. \\
b2(r,L): \quad & \text{bl}(n(r)) \leftarrow \text{ok}(n(r)), L^-. \\
c1(r): \quad & \text{ok}(n(r)) \leftarrow \text{ok}'(n(r),n(r_1)),...,\text{ok}'(n(r),n(r_k)). \\
c2(r,r'): \quad & \text{ok}'(n(r),n(r')) \leftarrow \text{not } (n(r) << n(r')). \\
c3(r,r'): \quad & \text{ok}'(n(r),n(r')) \leftarrow (n(r) << n(r')), \text{ap}(n(r')). \\
c4(r,r'): \quad & \text{ok}'(n(r),n(r')) \leftarrow (n(r) << n(r')), \text{bl}(n(r')). \\
t(r,r'',r'''): \quad & n(r)<<n(r'') \leftarrow (n(r)<<n(r')),(n(r')<<n(r'')). \\
as(r,r'): \quad & \neg(n(r')<<n(r)) \leftarrow n(r)<<n(r').
\end{align*}
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.
% 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).
Suppose that *normally* John prefers to use the car, but that he prefers the bus when it’s 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*