Core syntax translation from L4 to ASP for an ASP based expert system

September 21, 2021

1 Introduction

The purpose of this document is to give a translation from a restricted rule syntax (such as a subset of L4) to corresponding sets of ASP rules, needed for a preliminary prototype of the ASP expert system work. Given a single input rule in the syntax specified below, there is a single corresponding ASP rule for the main rule based reasoning functionality. (section 3) Furthermore corresponding to that input rule, there is a set of corresponding ASP rules for the question generation and optimization functionality,(section 4) and there is a third set of corresponding ASP rules for the justification graph generation functionality. (section 5) Then there is also some fixed ASP code needed for the whole system to work. (denoted as 'supporting code') Finally there is some ASP code that is specific to the PDPA use-case. (section 6)

2 Assumed input rule syntax

We assume that the input rule syntax from a source such as L4 is compatible with the syntax for prolog-like rules. More specifically we assume that each source rule has exactly the following form (or can be put into the following form):

```
pre_con_1(V1),pre_con_2(V2)...,pre_con_n(Vn) -> post_con(V).
```

We further make the folllowing assumptions:
1. Each pre-condition $pre\_con_i(V_i)$ is atomic and so is the post-condition $post\_con(V)$.
2. V_i is the set of variables occuring in the i^{th} pre-condition $pre\_con_i(V_i)$ and V is the set of variables occuring in the post condition $post\_con(V)$. We assume that $V_1 \cup V_2 \cup ... \cup V_n = V$.
3. Each variable occurring in either a pre-condition or the post condition is universally quantified over.
4. Each input rule of the form above carries with it an integer rule id. possibly originating in the legislation.

3 Main input rule translation

Given an input rule

```
pre_con_1(V1),pre_con_2(V2)...,pre_con_n(Vn) -> post_con(V).
```

obeying all the properties in (2), say this rule has integer rule id. n, then we write the following rule in our ASP program:

```
according_to(n,post_con(V)):-legally_holds(pre_con(V1)),...,legally_holds(pre_con(Vn)).
```

We repeat this for each input rule. Then we add the following ASP rules that encode defeasibility:

3.1 Supporting code

```
defeated(R2,C2):-overrides(R1,R2),according_to(R2,C2),legally_enforces(R1,C1),opposes(C1,C2).

opposes(C1,C2):-opposes(C2,C1).

legally_enforces(R,C):-according_to(R,C),not defeated(R,C).

legally_holds(C):-legally_enforces(R,C).

:-opposes(C1,C2),legally_holds(C1),legally_holds(C2).
```

4 ASP code for question generation and optimization

Given an input rule

```
pre_con_1(V1),pre_con_2(V2)...,pre_con_n(Vn) -> post_con(V).
```

obeying all the properties in (2), we add the following set of ASP rules to our ASP program:

```
explains(pre_con_1(V1),post_con(V),N+1):-query(post_con(V),N).
explains(pre_con_2(V2),post_con(V),N+1):-query(post_con(V),N).
                                .
                                .
                                .

explains(pre_con_n(Vn),post_con(V),N+1):-query(post_con(V),N).
```

We repeat this for each input rule. We then add the following bit of ASP code:

4.1 Supporting code

```
query(C,0):-generate_q(C).
query(C1,0):-generate_q(C),opposes(C,C1).

query(X,N):-explains(X,Y,N),q_level(N).
q_level(N+1):-query(X,N).



not_leaf(X):-explains(X,Y,N),explains(Z,X,N+1).
abducible(leaf,X,N):-explains(X,Y,N),not not_leaf(X).
abducible(nleaf,X,N):-explains(X,Y,N),not_leaf(X).



ask(X,N):-abducible(leaf,X,N).
ask(X,N):-abducible(nleaf,X,N),N>M,min_q_level(M).


max_q_level(N):-not q_level(N+1),q_level(N),N>0.

%Strong Constraint-data breach notifiable
:-generate_q(C),not legally_holds(C),const(N),not opp_const(N+1).

%Strong Constraint-data breach NOT notifiable
:-generate_q(C),legally_holds(C),opp_const(N),not const(N+1).
```

```
% Integrate with main program

{choose(X,N-M)}:-ask(X,M),max_q_level(N).
legally_holds(X):-choose(X,K).
ask_user(X):-choose(X,M).
legally_holds(X):-user_input(pos,X).
:-user_input(neg,X),legally_holds(X).

:~choose(X,M).[1@M,X,M]
```

5 ASP code for justification graph generation

Given an input rule

```
pre_con_1(V1),pre_con_2(V2)...,pre_con_n(Vn) -> post_con(V).
```

obeying all the properties in (2), with integer rule id n we add the following set of ASP rules to our ASP program:

```
caused_by(pos,legally_holds(pre_con_1(V1)),according_to(n,post_con(V)),N+1)
:-according_to(post_con(V)),legally_holds(pre_con_1(V1)),...,legally_holds(pre_con_n(Vn)),
justify(according_to(post_con(V),N).
                .
                .
                .
caused_by(pos,legally_holds(pre_con_n(Vn)),according_to(n,post_con(V)),N+1)
:-according_to(post_con(V)),legally_holds(pre_con_1(V1)),...,legally_holds(pre_con_n(Vn)),
justify(according_to(post_con(V),N).
```

We repeat this for each input rule and then add the following bit of ASP code

5.1 Supporting code

```
caused_by(pos,overrides(R1,R2),defeated(R2,C2),N+1):-defeated(R2,C2),overrides(R1,R2),
according_to(R2,C2),legally_enforces(R1,C1),opposes(C1,C2),justify(defeated(R2,C2),N).

caused_by(pos,according_to(R2,C2),defeated(R2,C2),N+1):-defeated(R2,C2),overrides(R1,R2),
according_to(R2,C2),legally_enforces(R1,C1),opposes(C1,C2),justify(defeated(R2,C2),N).

caused_by(pos,legally_enforces(R1,C1),defeated(R2,C2),N+1):-defeated(R2,C2),overrides(R1,R2),
according_to(R2,C2),legally_enforces(R1,C1),opposes(C1,C2),justify(defeated(R2,C2),N).

caused_by(pos,opposes(C1,C2),defeated(R2,C2),N+1):-defeated(R2,C2),overrides(R1,R2),
according_to(R2,C2),legally_enforces(R1,C1),opposes(C1,C2),justify(defeated(R2,C2),N).


caused_by(pos,according_to(R,C),legally_enforces(R,C),N+1):-legally_enforces(R,C),according_to(R,C),
not defeated(R,C),justify(legally_enforces(R,C),N).

caused_by(neg,defeated(R,C),legally_enforces(R,C),N+1):-legally_enforces(R,C),according_to(R,C),
not defeated(R,C),justify(legally_enforces(R,C),N).


caused_by(pos,legally_enforces(R,C),legally_holds(C),N+1):-legally_holds(C),legally_enforces(R,C),
not user_input(pos,C), justify(legally_holds(C),N).
```

```
caused_by(pos,user_input(pos,C),legally_holds(C),N+1):-legally_holds(C),
user_input(pos,C), justify(legally_holds(C),N).

justify(X,N):-caused_by(pos,X,Y,N),graph_level(N+1), not user_input(pos,X).
directedEdge(Sgn,X,Y):-caused_by(Sgn,X,Y,M).

graph_level(0..N):-max_graph_level(N).

justify(X,0):-gen_graph(X).

#show ask_user/1.
#show directedEdge/3.
```

6 Specific ASP code for PDPA use case

Here is some code that is specific to the PDPA use-case.

```
data_event(hospital_data,1).
generate_q(is_notifiable_data_breach(E,T)):-data_event(E,T).
min_q_level(0).
max_graph_level(6).

overrides(30,27).
opposes(is_notifiable_data_breach(E,T),not_is_notifiable_data_breach(E,T)):-data_event(E,T).


%gen_graph(legally_holds(is_notifiable_data_breach(hospital_data,1))).
%gen_graph(defeated(R,C)):-defeated(R,C).


% Toggle between the integrity constraints enforcing notifiable vs not notifiable data breach
const(0).
%opp_const(1).
%const(2).


% User inputed facts
%user_input(neg,is_personal_data(hospital_data,1)).
```