CSE 505 Project Final Report

Junyi Tao
Student ID: 112820617

1 Introduction

This project is based on the ICLP 2018 paper "Translating LPOD and CR-Prolog2 into standard answer set programs"[1]. This paper mainly focuses on how to reduce the specific two extensions of Answer Set Programmings (LPOD and CR-Prolog2) to the standard answer set programs while maintaining the similar functions and features as the extended versions. And in this project, we mainly focus on the part of translating LPOD program into the standard answer set programmings.

Answer set programming (ASP) is useful for nonmonotonic reasoning in knowledge representation. In ASP, with a problem that has been modeled, all answer sets for that problems will be outputted. Due to the fact that there may exist several different answers for the same problem, how to identify the likeness or preference of each answer set is needed for some scenarios. For instance, for hotel choice problem, there may exist several valid candidates to choose, which will be output by the standard answer set programming under some restrictions. However, how to choose the hotel among the valid candidate remains as an open problem. To match these needs, several extensions of standard ASP have been proposed in different aspects: LP^{MLN} and P-log extends the answer set programming by expressing the quantitative uncertainty for each answer set; Logic Programs with Ordered Disjunction (LPOD) extends the answer set programming to express a qualitative preference over the different answer sets. In LPOD, the qualitative preference is introduced as this example: considering a rule $A \times B \leftarrow Body$, it means when Body is true, if possible then A but if A is not possible, at least B must be true. A problem in LPOD consist of several such rules and the answer sets of this LPOD problem should satisfy all rules. After finding the answer sets of the problem, the most preferred one can be identified as follows: as the above example, the preference degree of rule $A \times B \leftarrow Body$ is defined as follows: if body is not true, the degree is 1; if body is true and A is true, the degree is 1; if body is true and A is not true and B is true, the degree is 2. There are several methods to choose the preferred one based the preference degrees as discussed in the following section.

After the introduction of several extensions of the ASP, people started thinking is it possible to use the standard ASP to realize these ASP extensions. In this paper, the authors summarized two translation works they have finished, including translating LPOD, and CR-Prolog2 to the standard ASP. In this work, we study how the LPOD to realize the functions that identify the preference of each answer set , the difference of traditional LOPD method and the novel

translation method proposed by the authors, learn about how it can achieve the goal of translation, implement the method to on two examples, check the correctness of this approach. If possible, I will try to think about the ways to improve the method in this paper, including both functions and performances. All the project material will be uploaded on the github page [2].

2 Methodology

2.1 LPOD

Syntax. In LPOD, $\prod$ is $\prod_{reg} \cup \prod_{ord}$ where $\prod_{reg}$ is regular ASP rules as $Head \leftarrow Body$ and the $\prod_{ord}$ consists of rules like $C^1 \times C^2 ... \times C^n \leftarrow Body$. In such rules, C^i are atoms and the number of C^i is at least 2. It means when Body is true, if possible then C^1; if C^1 is not possible then C^2;..., if all of C^1,..., C^{n-1} are not possible then the last one C^n must be possible.

Semantics. In LPOD, if the i-th atoms of the rules is the options, it is equivalent to: $C^i \leftarrow Body, not C^1, ..., not C^{i-1}$.

Let $\prod$ be an LPOD, a split program of $\prod$ is attained by replacing each rule in $\prod_{ord}$ by one of its options. A set S of atoms is a candidate answer set of $\prod$ if it is an answer set of a split program of $\prod$. Note: not all split programs have a corresponding answer set. Continue the simple example in section 1, the rule $A \times B \leftarrow Body$ has two split program: (1)$A \leftarrow Body$, and (2)$B \leftarrow Body, not\ A$.

The degree of a answer set S is defined as: 1) 1, if S does not satisfy Body of the rule; 2) j ($1 <= j <= n$), if S satisfy the Body and $j = min\{k | C^k \in S\}$.

In LPOD, there are four methods to determine the preference of each answer set based on the degree discussed above: Cardinality-Preferred, Inclusion-Preferred, Pareto-Preferred, Penalty-Sum-Preferred. Now we mainly discuss the Penalty-Sum-Preferred methods: consider a LPOD problem consist of n $\prod_{ord}$ rules and for one answer set i, the corresponding preference are $\{i_1, i_2, ..., i_n\}$, just compute the sum of $\{i_1, i_2, ..., i_n\}$ for each set and choose the set with smallest sum of degrees.

Examples. Considering the following LPOD problem $\prod_1$: $a \times b \leftarrow not\ c$, $b \times c \leftarrow not\ d$.

It has four split programs: (1) $a \leftarrow not\ c$, $b \leftarrow not\ d$; (2) $b \leftarrow not\ c, not\ a$, $b \leftarrow not\ d$; (3) $a \leftarrow not\ c$, $c \leftarrow not\ d, not\ b$; (4) $b \leftarrow not\ c, not\ a$, $c \leftarrow not\ d, not\ b$.

And it has three candidate answer sets: (1) $\{a, b\}$, (2) $\{b\}$, (3) $\{c\}$. And $\{a, b\}$ is the preferred answer set since it satisfy the two rules with degree 1.

2.2 From LPOD to Answer Set Programs

Now we will discuss the methods used in the paper to translate LPOD to Answer Set Programs. The detail of this part can be found in [1]. They named the function as $lpod2asp(\prod)$. The translation of LPOD into ASP is based on assumption

programs to generate the candidate answer sets. Then we will discuss how the assumption programs work.

In general, the translating method works on an opposite direction of the traditional LPOD program. In traditional LPOD program, given a problem with n rules, it firstly find all split programs based on each rule as discussed above, then try to find valid answer sets for each split programs and these should be valid answer sets for this question. Finally, compute the preference degrees for each answer sets and find the most preferred one. In the translating work, given the LPOD problem, it firstly assume all possible sets of preference degrees for each rule, i.e. assumes that the problem has n ordered rules and each rule has n candidate heads, there are $(n+1) \times n$ different preference degree sets (in LPOD, if body is not true, the degree is 1, this work set it to 0 initially for operation and finally reverse it to 1 for such condition). Each preference degree set is $\{p_1, p_2, ..., p_n\}$, where $p_i = 0, 1, ..., n$ denote the preference degree of rule i. The method then will check each set of the preference degrees on each rule to test whether this set is valid or not based on some restrictions discussed below. After finding the valid sets, use the predefined preference degree of each set to find the corresponding value and also find the most preferred answer set.

Let $\prod$ be an LPOD of signature σ where $\prod_{od}$ contains m propositional rules with ordered disjunction: rule 1 is $C_1^1 \times C_1^2 ... \times C_1^{n_1} \leftarrow Body_1$, ... rule m is $C_m^1 \times C_m^2 ... \times C_m^{n_m} \leftarrow Body_m$, where $n_i >= 2$ for $1 =< i =< m$.

Its x-th assumpiton ($x \in \{0, ..., n_i\}$) is defined as the set of ASP rules: $body_i \leftarrow Body_i$ (1), $\perp \leftarrow x = 0, body_i$ (2), $\perp \leftarrow x > 0, not\ body_i$ (3), $C_i^j \leftarrow x = j,\ body_i$ (4), $\perp \leftarrow x \neq j,\ body_i, not\ C_i^1, ..., not\ C_i^{j-1}, C_i^j$ (5), where $body_i$ is a new and distinct atom for rule i, which is used to transfer $Body_i$ to an atom.

The rule (2) (3) ensure that the body is false iff x=0; the rule (4) represent that C_i^x is true under the x-th assumption, and rule (5) ensures that all atoms $C_i^1, ..., C_i^{x-1}$ are false. Combine all five rules together means: if x=0, the Body of rule must be false; whereas $x > 0$ assumes $Body_i$ is true and the x-th atom is the first one which can be true in this rule.

An assumption program of an LPOD $\prod$ is obtained from $\prod$ by replacing each rule in $\prod_{od}$ by one of its assumptions. If each LPOD rule i is replaced by its x_i-th assumption, we call $(x_1, ..., x_m)$ the assumption degree list of the assumption program.

This five restrictions can be used to build the model of each rule in LPOD problem, and at the beginning, it requires to generate all possible degrees sets as the input for testing a specific sets is valid or not. The generating candidates for preference degree sets can be done by this rules: (1)

Examples continue. The assumptions for rule $a \times b \leftarrow not\ c$, denoted by $O_1(X_1)$; and for the rule $b \times c \leftarrow not\ d$, denoted by $O_2(X_2)$ are as follows, where X_1 and X_2 range over $\{0, 1, 2\}$.

$O_1(X_1)$: $body_1 \leftarrow not\ c$; $\perp \leftarrow X_1 = 0, body_1$; $\perp \leftarrow X_1 > 0, not\ body_1$; $a \leftarrow body_1, X_1 = 1$; $b \leftarrow body_1, X_1 = 2$; $\perp \leftarrow X_1 \neq 1,\ body_1, a$; $\perp \leftarrow X_1 \neq 2,\ body_1, not\ a, b$.

$O_1(X_2)$: $body_1 \leftarrow not\ d$; $\perp \leftarrow X_2 = 0, body_2$; $\perp \leftarrow X_2 > 0, not\ body_2$; $b \leftarrow body_2, X_1 = 1$; $c \leftarrow body_2, X_2 = 2$; $\perp \leftarrow X_2 \neq 1,\ body_2, b$; $\perp \leftarrow X_2 \neq 2,\ body_2, not\ b, c$.

And this is the answer set program code which can be run on clingo for this example (this code is provided by the authors of this paper).

```
{ap(X1,X2): X1=0..2, X2=0..2}.    :~ ap(X1,X2). [-1,X1,X2]
%a*b<-notc.
body_1(X1,X2) :- ap(X1,X2), not c(X1,X2).
:- ap(X1,X2), X1=0, body_1(X1,X2).
:- ap(X1,X2), X1>0, not body_1(X1,X2).
a(X1,X2) :- body_1(X1,X2), X1=1.
b(X1,X2) :- body_1(X1,X2), X1=2.
:- body_1(X1,X2), X1!=1, a(X1,X2).
:- body_1(X1,X2), X1!=2, not a(X1,X2), b(X1,X2).

%b*c<-notd.
body_2(X1,X2):- ap(X1,X2), not d(X1,X2).
:- ap(X1,X2), X2=0, body_2(X1,X2).
:- ap(X1,X2), X2>0, not body_2(X1,X2).
b(X1,X2) :- body_2(X1,X2), X2=1.
c(X1,X2) :- body_2(X1,X2), X2=2.
:- body_2(X1,X2), X2!=1, b(X1,X2).
:- body_2(X1,X2), X2!=2, not b(X1,X2), c(X1,X2).

%%%% 3 %%%%
1{degree(ap(X1,X2), D1, D2): D1=1..2, D2=1..2}1 :- ap(X1,X2).
:- degree(ap(X1,X2), D1, D2), X1=0, D1!=1.
:- degree(ap(X1,X2), D1, D2), X1>0, D1!=X1.
:- degree(ap(X1,X2), D1, D2), X2=0, D2!=1.
:- degree(ap(X1,X2), D1, D2), X2>0, D2!=X2.


sum(P,N) :- degree(P,D1,D2), N=D1+D2.
prf(P1,P2) :- sum(P1,N1), sum(P2,N2), N1<N2.
pAS(X1,X2) :- ap(X1,X2), {prf(P, ap(X1,X2))}0.
```

After finding the valid preference degree sets for the LPOD problem, it needs to compute the preference degrees and find the most preferred one. There are 4 typical ways to evaluate the preference degree, and we just discuss the Penalty-Sum-Preferred method: compute the sum of preference degrees of each set and choose the smallest one. This work can complete by the last three rules of the code shown above.

3 The implementation of LPOD reduction method

In this section, we will discuss our work to realize the LPOD reduction method on standard answer set programming. We will show a little more complex LPOD models than the author do.

3.1 The implementation example

The example we implement is about how to spend a weekend: Assume that you have three candidate choice of place to go: beach, cinema and library. Normally you prefer the cinema over the library, and prefer library over the beach, unless it is hot (the order of cinema and library will not change). Meanwhile, assume the current weather is not summer. As a consequence, the probability of the weather is cool is larger than the weather is hot. Then we can use the LPOD program to construct the problems with the following rules: (1)$cinema \times library \times beach \leftarrow cool$; (2)$beach \times cinema \times library \leftarrow hot$; (3)$cool \times hot \leftarrow not\ summer$. Then we can build the translating model as follows:

Implementation Example. The assumptions for rule $cinema \times library \times beach \leftarrow cool$, denoted by $O_1(X_1)$; and for the rule $beach \times cinema \times library \leftarrow hot$, denoted by $O_2(X_2)$ are as follows; and for the rule $cool \times hot \leftarrow not\ summer$, denoted by $O_3(X_3)$ are as follows, where X_1 and X_2 range over $\{0, 1, 2, 3\}$ and X_3 range over $\{0, 1, 2\}$.

$O_1(X_1)$: $body_1 \leftarrow cool$; $\perp \leftarrow X_1 = 0, body_1$; $\perp \leftarrow X_1 > 0, not\ body_1$; $cinema \leftarrow body_1, X_1 = 1$; $library \leftarrow body_1, X_1 = 2$, $beach \leftarrow body_1, X_1 = 3$; $\perp \leftarrow X_1 \neq 1,\ body_1, cinema$; $\perp \leftarrow X_1 \neq 2,\ body_1, not\ cinema, library$, $\perp \leftarrow X_1 \neq 3,\ body_1, not\ cinema, not\ library, beach$.

$O_2(X_2)$: $body_2 \leftarrow hot$; $\perp \leftarrow X_2 = 0, body_2$; $\perp \leftarrow X_2 > 0, not\ body_2$; $beach \leftarrow body_2, X_1 = 1$; $cinema \leftarrow body_2, X_2 = 2$; $library \leftarrow body_2, X_2 = 3$; $\perp \leftarrow X_2 \neq 1,\ body_2, beach$; $\perp \leftarrow X_2 \neq 2,\ body_2, not\ beach, cinema$, $\perp \leftarrow X_2 \neq 3,\ body_2, not\ beach, not\ cinema, library$.

$O_3(X_3)$: $body_3 \leftarrow not\ summer$; $\perp \leftarrow X_3 = 0, body_3$; $\perp \leftarrow X_2 > 0, not\ body_3$; $cool \leftarrow body_3, X_1 = 1$; $hot \leftarrow body_3, X_2 = 2$; $\perp \leftarrow X_2 \neq 1,\ body_3, cool$; $\perp \leftarrow X_2 \neq 2,\ body_3, not\ cool, hot$.

And the code of my implementation is as follows:

```
%Implement the following LPOD program into ASP on Clingo: 1) cinema  beach<-not
%hot;2)beach  cinema<-hot 3) cool hot<-not summer.
%generate all possible preference degree sets
{ap(X1,X2,X4): X1=0..3, X2=0..3,X4=0..2}. :~ ap(X1,X2,X4). [-1, X1, X2,X4]


% cinema  library beach <-cool.
body_1(X1,X2,X4) :- ap(X1,X2,X4), cool(X1,X2,X4).
:- ap(X1,X2,X4), X1=0, body_1(X1,X2,X4).
:- ap(X1,X2,X4), X1>0, not body_1(X1,X2,X4).
```

```
cinema(X1,X2,X4):- body_1(X1,X2,X4), X1=1.
library(X1,X2,X4):- body_1(X1,X2,X4), X1=2.
beach(X1,X2,X4):- body_1(X1,X2,X4), X1=3.
:- body_1(X1,X2,X4), X1!=1, cinema(X1,X2,X4).
:- body_1(X1,X2,X4), X1!=2, not cinema(X1,X2,X4), library(X1,X2,X4).
:- body_1(X1,X2,X4), X1!=3, not cinema(X1,X2,X4), not library(X1,X2,X4), beach(X1,X2,X4).

% beach  cinema library<-hot.
body_2(X1,X2,X4) :- ap(X1,X2,X4),hot(X1,X2,X4).
:- ap(X1,X2,X4), X2=0, body_2(X1,X2,X4).
:- ap(X1,X2,X4), X2>0, not body_2(X1,X2,X4).
beach(X1,X2,X4):- body_2(X1,X2,X4), X2=1.
cinema(X1,X2,X4):- body_2(X1,X2,X4), X2=2.
library(X1,X2,X4):- body_2(X1,X2,X4), X2=3.
:- body_2(X1,X2,X4), X2!=1, beach(X1,X2,X4).
:- body_2(X1,X2,X4), X2!=2, not beach(X1,X2,X4), cinema(X1,X2,X4).
:- body_2(X1,X2,X4), X2!=3, not beach(X1,X2,X4), not cinema(X1,X2,X4), library(X1,X2,X4).

% cool hot<-not summer.
body_4(X1,X2,X4):- ap(X1,X2,X4),not summer(X1,X2,X4).
:- ap(X1,X2,X4), X4=0, body_4(X1,X2,X4).
:- ap(X1,X2,X4), X4>0, not body_4(X1,X2,X4).
hot(X1,X2,X4):- body_4(X1,X2,X4), X4=2.
cool(X1,X2,X4):- body_4(X1,X2,X4), X4=1.
:- body_4(X1,X2,X4), X4!=1, cool(X1,X2,X4).
:- body_4(X1,X2,X4), X4!=2, not cool(X1,X2,X4), hot(X1,X2,X4).

%This part is used to reset the degree since the initial degree set
%method has one modification than the normal mthod.
1{degree(ap(X1,X2,X4), D1, D2, D4): D1=1..3, D2=1..3, D4=1..2}1 :- ap(X1,X2,X4).
:- degree(ap(X1,X2,X4), D1, D2,D4), X1=0, D1!=1.
:- degree(ap(X1,X2,X4), D1, D2,D4), X1>0, D1!=X1.
:- degree(ap(X1,X2,X4), D1, D2,D4), X2=0, D2!=1.
:- degree(ap(X1,X2,X4), D1, D2,D4), X2>0, D2!=X2.
:- degree(ap(X1,X2,X4), D1, D2,D4), X4=0, D4!=1.
:- degree(ap(X1,X2,X4), D1, D2,D4), X4>0, D4!=X4.

sum(P,N) :- degree(P,D1,D2,D4), N=D1+D2+D4.
prf(P1,P2) :- sum(P1,N1), sum(P2,N2), N1<N2.
pAS(X1,X2,X4) :- ap(X1,X2,X4), {prf(P, ap(X1,X2,X4))}0.
```

After running this program on clingo, we can get the following valid answer sets : ap(1,0,1), ap(2,0,1), ap(3,0,1), ap(0,1,2), ap(0,2,2), ap(0,3,2), and it will find that ap(1,0,1) is the most preferred answer set (the corresponding degree is (1,1,1) since the degree is set to 0 initially for rule in which body is not true), and the choice of place is most likely cinema. After observing the result, we find

that the third rule is always true, since the first two rules depends on the result of third rules: only when the weather is cool or hot has been determined, we can give a plan accordingly. For this part, we will upload this code and a little more complex LPOD programming problem on github project page [2].

3.2 Observations of the implementation

Although this work proposed a novel and effective way to reduce LPOD to the standard answer set programming. After consideration of the work, we think some open problems are remained, and the most important one is that: to set the restrictions of each rule of the LPOD model, we need to hard code each restrictions method for each head candidate. Therefore, once each rule contains a large number of head candidates, we need to set a large number of restrictions accordingly. Consider the example implemented by us, we need to set $\bot \leftarrow X_1 \neq 1,\ body_1, cinema$; $\bot \leftarrow X_1 \neq 2,\ body_1, not\ cinema, library$, $\bot \leftarrow X_1 \neq 3,\ body_1, not\ cinema, not\ library, beach$ for the first rule. Furthermore, if we have 100 candidates, we need to set 100 such rules accordingly. Therefore, We think such work is not suitable for large LPOD programming problems, and a more efficient way for implementation is wanted.

4 Conclusion

In this project, I extend my knowledge of logic programming by learning the extensions of the one of the course topic Answer set programming. After reading the related paper, I have a better understanding of the answer set programming including the limitations of ASP. And then I learned how to reduce the LPOD into the standard ASP program. Compared with the work to implement example of LPOD into ASP, I think understanding the thoughts of the reduction method requires more effort and and is more important. Meanwhile, the ways to design reduction method in this paper also inspire me how to think about a problem in different ways, which I think is most helpful for me. By the way, after the observation of the implementation for the method, I found several open problems and will continue thinking a solution about that.

References

1. Lee, Joohyung, and Zhun Yang. "Translating LPOD and CR-Prolog 2 into standard answer set programs." Theory and Practice of Logic Programming 18.3-4 (2018): 589-606.
2. https://github.com/JaydenTaojy/junyi-505-project