NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Implementing a Logical Semantics of LPOD using ASP

Georgios K. Nikolaou

Supervisors: **Panagiotis Rondogiannis,** Professor
Angelos Charalambidis, Assistant Professor

ATHENS

JULY 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση μιας λογικής σημασιολογίας των LPOD με τη χρήση ASP

Γεώργιος Κ. Νικολάου

Επιβλέποντες: **Παναγιώτης Ροντογιάννης,** Καθηγητής
Άγγελος Χαραλαμπίδης, Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2024

BSc THESIS

Implementing a Logical Semantics of LPOD using ASP

Georgios K. Nikolaou
S.N.: 1115202000153

SUPERVISORS: **Panagiotis Rondogiannis,** Professor
Angelos Charalambidis, Assistant Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση μιας λογικής σημασιολογίας των LPOD με τη χρήση ASP

Γεώργιος Κ. Νικολάου
Α.Μ.: 1115202000153

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Παναγιώτης Ροντογιάννης,** Καθηγητής
Άγγελος Χαραλαμπίδης, Επίκουρος Καθηγητής

ABSTRACT

Logic Programs with Ordered Disjunction (LPODs) extend classical logic programs by expressing preferences among alternatives in rule heads. Intuitively, the ordered disjunction of A and B, written as $A \times B$, means "A *is preferred, but if* A *is not possible, then at least* B". The original LPOD semantics can produce counterintuitive solutions in some cases. In [1], a new logical semantics for LPODs is proposed, introducing an additional truth value to indicate failure to satisfy preferences. We implement this new semantics by translating LPODs into standard Answer Set Programs (ASP), allowing us to use answer set solvers to find the most preferred answer sets. We show that our implementation's performance is comparable to existing systems.

SUBJECT AREA: Programming Language Semantics

KEYWORDS: Logic Programs with Ordered Disjunction, Answer Set Programming, many-valued semantics

ΠΕΡΙΛΗΨΗ

Τα Λογικά Προγράμματα με Διατεταγμένη Διάζευξη (LPODs) επεκτείνουν τα κλασικά λογικά προγράμματα με την έκφραση προτιμήσεων μεταξύ εναλλακτικών στις κεφαλές των κανόνων. Διαισθητικά, η διατεταγμένη διάζευξη του A με το B, που συμβολίζεται $A \times B$, σημαίνει "*το A είναι προτιμότερο, αλλά αν το A δεν είναι δυνατό, τότε τουλάχιστον B*". Η αρχική σημασιολογία των LPODs παράγει μη διαισθητικές λύσεις σε μερικές περιπτώσεις. Στο [1], προτείνεται μια νέα λογική σημασιολογία για τα LPODs, εισάγοντας μια επιπρόσθετη τιμή αλήθειας που δηλώνει την αποτυχία ικανοποίησης των προτιμήσεων. Υλοποιούμε αυτή τη νέα σημασιολογία μεταφράζοντας τα LPODs σε Answer Set Programs (ASP), επιτρέποντάς μας να χρησιμοποιήσουμε answer set solvers για να βρούμε τα προτιμότερα answer sets. Δείχνουμε ότι η απόδοση της υλοποίησής μας μπορεί να συγκριθεί με υπάρχοντα συστήματα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σημασιολογία Γλωσσών Προγραμματισμού

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Λογικά Προγράμματα με Διατεταγμένη Διάζευξη, Answer Set Programming, πολύτιμη σημασιολογία

CONTENTS

LIST OF TABLES

1. INTRODUCTION

Logic Programs with Ordered Disjunction (LPODs) extend classical logic programs [2] by expressing preferences among alternatives in rule heads. Specifically, they allow rule heads to be formulas of the form $C_1 \times \ldots \times C_n$, where $\times$ is a propositional connective called “ordered disjunction” and C_i’s are literals. Intuitively, $C_1 \times \ldots \times C_n$ means “*C_1 is preferred; but if C_1 is not possible, then C_2 suffices; ... if all $C_1, \ldots, C_{n-1}$ are not possible, then C_n suffices*”. LPODs were introduced in [3].

The original semantics [3, 4] rely on a two-phase procedure, starting with generating the candidate answer sets of a LPOD and then filtering them in order to obtain the most preferred ones. The filtering step involved computing the *degree of satisfaction* for each of its ordered disjunctive rules. However, there are cases where following this semantics could lead to undesirable results.

In [1], a new many-valued logical semantics for LPODs is introduced. It is purely model-theoretic and does not rely on degrees of satisfaction. Instead, it incorporates an additional truth value to select the most preferred models based on a preferential ordering, ensuring that top preferences fail only if they are impossible to satisfy. This approach overcomes the shortcomings of the traditional LPOD semantics. However, there has been no implementation of this semantics so far.

In this thesis we implement this many-valued semantics for LPODs. We show that the new semantics can be translated into standard Answer Set Programs (ASP). Even though the answer sets of the translated logic program are two-valued, the additional truth value can be computed by introducing extra predicates. We utilize this as we experiment with four translations, each based on an existing LPOD system that implements the traditional semantics.

We implemented a transpiler[1], written in Haskell, that automatically generates the ASP encodings for a given LPOD. We use *clingo* and *asprin* as the ASP solvers to compute and identify the most preferred answer sets. We evaluated the performance of our implementations of this new semantics, comparing them both against each other and against other existing LPOD systems implementing the original semantics. The experimental results indicate that the performance of our implementations is comparable to the existing systems, therefore the new LPOD semantics can indeed be implemented efficiently.

Chapter 2 provides the necessary background on the syntax and semantics of LPODs. Chapter 3 reviews related work. Chapter 4 presents known ways to compute LPOD answer sets. Chapter 5 demonstrates a way for the ASP encodings to generate the F^* values for three-valued answer sets and introduces the four translations from LPOD to ASP. Chapter 6 presents the experiments and compares our implementations with each other as well as with other existing systems. Chapter 7 concludes by discussing potential directions for future research.

[1] `https://github.com/grnkl/lpod2asp`

2. LOGIC PROGRAMS WITH ORDERED DISJUNCTION

2.1 Syntax and Original Semantics

Logic Programs with Ordered Disjunction (LPODs) extend classical logic programs [2] by expressing preferences among alternatives in rule heads.

Definition 1. A (propositional) LPOD is a set of rules of the form:

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k$$

where the C_i , A_j , and B_l are ground literals.

Intuitively, $C_1 \times \ldots \times C_n$ means "C_1 is preferred; but if C_1 is not possible, then C_2 suffices; ... if all $C_1, \ldots, C_{n-1}$ are not possible, then C_n suffices".

We will call a rule *regular* if it has only one head atom, otherwise (if $n > 1$), we will call it *ordered disjunctive*.

An interpretation of an LPOD is a set of literals. An interpretation I is called *consistent* if there does not exist any atom A such that both A and $\neg A$ belong to I.

Definition 2. An interpretation M is a model of an LPOD P if and only if for every rule

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k$$

if $\{A_1, \ldots, A_m\} \subseteq M$ and $\{B_1, \ldots, B_k\} \cap M = \emptyset$ then there exists $C_i \in M$.

Brewka [3] introduced a two-phase procedure to obtain the preferred answer sets of an LPOD. In the first phase, the *candidate* answer sets of the LPOD are produced. In the second phase, the *most preferred* ones are extracted by filtering the candidate answer sets.

Definitions 3, 4 and 5 are associated with the first step, i.e. producing the answer sets of the LPOD.

Definition 3. For an LPOD rule $C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k$, its *$k$-th option* ($k \in \{1, \ldots, n\}$) is defined as:

$$C_i \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k,\ \text{not}\ C_1, \ldots,\ \text{not}\ C_{i-1}.$$

A *split program* of an LPOD P is obtained from P by replacing each ordered disjunctive rule in P by one of its options.

Example 1. The LPOD program (from [3])

$$a \times b \ \leftarrow\ not\ c, \qquad\qquad b \times c \ \leftarrow\ not\ d$$

has the following split programs:

$$\begin{array}{ll} a \leftarrow\ not\ c & \qquad a \leftarrow\ not\ c \\ b \leftarrow\ not\ d & \qquad c \leftarrow\ not\ d,\ not\ b \\ & \\ b \leftarrow\ not\ c,\ not\ a & \qquad b \leftarrow\ not\ c,\ not\ a \\ b \leftarrow\ not\ d & \qquad c \leftarrow\ not\ d,\ not\ b \end{array}$$

Definition 4. Let P be an LPOD. The $\times$-reduct of a rule R of P of the form:

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m, \ not \ B_1, \ldots, \ not \ B_k$$

with respect to a set of literals I, is denoted by $R^I_\times$ and is defined as follows:

$$R^I_\times = \{Ci \leftarrow A_1, \ldots, A_m \mid Ci \in I \text{ and } I \cap \{C_1, \ldots, C_{i-1}, B_1, \ldots, B_k\} = \emptyset\}$$

The $\times$-reduct of P with respect to I is denoted by $P^I_\times$ and is the union of the reducts $R^I_\times$ for all R in P.

Definition 5. A set M of literals is an answer set of an LPOD P if M is a consistent model of P and M is the least model of $P^M_\times$.

In the second phase, the "most preferred" answer sets are produced using the concept of the *the degree of satisfaction of a rule* in an answer set:

Definition 6. Let M be an answer set of an LPOD P. Then, M satisfies the rule:

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m, \ not \ B_1, \ldots, \ not \ B_k$$

- to degree 1 if $A_j \notin M$, for some j, or $B_i \in M$, for some i,
- to degree l, $1 \leq l \leq n$, if all $A_j \in M$, no $B_i \in M$, and $l = \min\{r \mid C_r \in M\}$

When an LPOD P contains m ordered disjunctive rules, the satisfaction degree list of a candidate answer set M of P is $(d_1, \ldots, d_m)$ where d_i is the degree to which M satisfies the i-th ordered disjunctive rule.

Definition 7. For a candidate answer set M of LPOD P, let $M^i(P)$ denote the set of ordered disjunctive rules satisfied by M to degree i. For candidate answer sets M_1 and M_2 of P, four preference relations have been introduced:

1. **Cardinality-Preferred:** M_1 is *cardinality-preferred* to M_2 if there is a $i \geq 1$ such that $|M^i_1(P)| > |M^i_2(P)|$, and $|M^j_1(P)| = |M^j_2(P)|$ for all $j < i$. [4]
2. **Inclusion-Preferred:** M_1 is *inclusion-preferred* to M_2 if there is a $i \geq 1$ such that $M^i_2(P) \subset M^i_1(P)$, and $M^j_1(P) = M^j_2(P)$ for all $j < i$. [3]
3. **Pareto-Preferred:** M_1 is *Pareto-preferred* to M_2 if there is a rule that is satisfied to a lower degree in M_1 than in M_2, and there is no rule that is satisfied to a lower degree in M_2 than in M_1. [4]
4. **Penalty-Sum-Preferred:** M_1 is *penalty-sum-preferred* to M_2 if the sum of the satisfaction degrees of all rules is smaller in M_1 than in M_2. [5]

Example 2. Consider the "cars" program:

```
mercedes × bmw.
gas_mercedes × diesel_mercedes ← mercedes.
¬gas_mercedes.
```

which reads "*I prefer to buy a Mercedes than a BMW. If a Mercedes is available, I prefer a gas model to a diesel one. A gas model of Mercedes is not available*". This LPOD has two candidate answer sets: $M_1 = \{\texttt{mercedes}, \texttt{diesel\_mercedes}, \neg\texttt{gas\_mercedes}\}$ and $M_2 = \{\texttt{bmw}, \neg\texttt{gas\_mercedes}\}$. M_1 satisfies the first rule with degree 1, the second rule with degree 2, and the third rule with degree 1. M_2 satisfies the first rule with degree 2, the second rule with degree 1 (since its body evaluates to false), and the third rule with degree 1. According to the preference statements in Definition 7, the program's answer sets are either incomparable or both (equally) preferred.

Example 3. (From [4]) Assume that we want to book accommodation for a conference. We prefer a 3-star hotel over a 2-star hotel. We also prefer our hotel to be in walking distance from the conference venue:

$$\texttt{walking} \times \neg\texttt{walking}.$$
$$\texttt{3-stars} \times \texttt{2-stars}.$$

Also assume that the only 3-star hotel (say $hotel_1$) is not in walking distance, and the only 2-star hotel (say $hotel_2$) is in walking distance. The answer sets representing these two options are incomparable according to the four preference criteria, because $hotel_1$ satisfies the first rule to degree 2 and the second rule to degree 1, while $hotel_2$ satisfies the first rule to degree 1 and the second rule to degree 2.

Now assume we learn that there is also a 4-star hotel that, however, is not an option due to financial reasons. We make the following (seemingly "innocent") change to the program:

$$\texttt{walking} \times \neg\texttt{walking}.$$
$$\texttt{4-stars} \times \texttt{3-stars} \times \texttt{2-stars}.$$
$$\neg\texttt{4-stars}.$$

In the new program, $hotel_1$ satisfies the first rule to degree 2 and the second rule to degree 2, while $hotel_2$ satisfies the first rule to degree 1 and the second rule to degree 3. Now, $hotel_2$ is preferred to $hotel_1$, according to the cardinality-preference and inclusion-preference criteria. On the other hand, under the Pareto-preference and the penalty-sum-preference criteria, the answer sets remain incomparable.

This example is used in [4] to demonstrate how the inclusion-preference criterion is affected by the presence of unsatisfiable better options. We realize that using the "degree of satisfaction of rules" semantics, an apparently insignificant modification to the program can have a significant impact on the final preferred answer set.

2.2 Redefining the Answer Sets of LPODs using Three-Valued Semantics

In [1], a third truth value F^* is introduced to define the semantics of LPODs. Intuitively, F^* means "impossible to make true," while F means "false by default." In the ordered disjunction $C_1 \times C_2$, we accept C_2 only if C_1 is impossible to make true, meaning that attempting to make C_1 true would cause the interpretation to become inconsistent.

Example 4. Consider the program:

$$\texttt{wine} \times \texttt{beer}.$$
$$\neg\texttt{wine}.$$

Under traditional semantics, the most preferred (and only) answer set of this program is $\{\texttt{beer}, \neg\texttt{wine}\}$, but according to the three-valued approach, the most preferred answer set is $\{(\texttt{wine}, F^*), (\texttt{beer}, T), (\neg\texttt{wine}, T)\}$. Notice that $\{(\texttt{wine}, F), (\texttt{beer}, T), (\neg\texttt{wine}, T)\}$ is an inconsistent interpretation of the program, hence $\texttt{wine}$ receives the value F^* (it's not possible to make it equal to T). The semantics of "$\times$" is defined as follows:

Let $u, v \in \{F, F^*, T\}$. Then:

$$u \times v = \begin{cases} v, & \text{if } u = F^* \\ u, & \text{otherwise} \end{cases}$$

The intuitive meaning of this definition is that we return v *only if it is impossible to satisfy* u; otherwise we return u.

The new definition of the answer sets of LPODs presented in [1] is based on a three-valued logic and enables us to figure out the most preferred answer sets using a purely model-theoretic approach.

Definition 8. Let Σ be a nonempty set of propositional literals. The set of well-formed formulas is inductively defined as follows:

- Every element of Σ is a well-formed formula,
- The 0-place connective F^* is a well-formed formula,
- If ϕ_1 and ϕ_2 are well-formed formulas, then $(\phi_1 \wedge \phi_2)$, $(\phi_1 \vee \phi_2)$, $(\textit{ not } \phi_1)$, $(\phi_1 \leftarrow \phi_2)$, and $(\phi_1 \times \phi_2)$, are well-formed formulas.

The meaning of formulas is defined over the set of truth values $\{F, F^*, T\}$ which are ordered as $F < F^* < T$. For v_1, v_2 in $\{F, F^*, T\}$, we write $v_1 \leq v_2$ iff either $v_1 < v_2$ or $v_1 = v_2$.

Definition 9. A (three-valued) interpretation I is a function from Σ to the set $\{F, F^*, T\}$. We can extend I to apply to formulas, as follows:

$$\begin{array}{lcl}
I(F^*) & = & F^* \\
I(\textit{ not } \phi) & = & \begin{cases} T, & \text{if } I(\phi) \leq F^* \\ F, & \text{otherwise} \end{cases} \\
I(\phi \leftarrow \psi) & = & \begin{cases} T, & \text{if } I(\phi) \geq I(\psi) \\ F, & \text{otherwise} \end{cases} \\
I(\phi_1 \wedge \phi_2) & = & \min\{I(\phi_1), I(\phi_2)\} \\
I(\phi_1 \vee \phi_2) & = & \max\{I(\phi_1), I(\phi_2)\} \\
I(\phi_1 \times \phi_2) & = & \begin{cases} I(\phi_2), & \text{if } I(\phi_1) = F^* \\ I(\phi_1), & \text{otherwise} \end{cases}
\end{array}$$

Notice that the meanings of "$\wedge$", "$\vee$" and "$\times$" are associative, therefore writing $I(\phi_1 \vee \cdots \vee \phi_n)$, $I(\phi_1 \wedge \cdots \wedge \phi_n)$, and $I(\phi_1 \times \cdots \times \phi_n)$ (without using parentheses) is unambiguous.

Definition 10. An interpretation I is a *model* of an LPOD P if every rule of P evaluates to T under I. An interpretation I of P is called *consistent* if there do not exist literals A and $\neg A$ in P such that $I(A) = I(\neg A) = T$.

Definition 11. Let P be an LPOD. The $\times$-reduct of a rule R of P of the form:

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k$$

with respect to an interpretation I, is denoted by $R^I_\times$ and is defined as follows:

- If $I(B_i) = T$ for some i, $1 \leq i \leq k$, then $R^I_\times$ is the empty set.
- If $I(B_i) \neq T$ for all i, $1 \leq i \leq k$, then $R^I_\times$ is the set that contains the rules:

$$\begin{array}{lcl} C_1 & \leftarrow & F^*, A_1, \ldots, A_m \\ & \cdots & \\ C_{r-1} & \leftarrow & F^*, A_1, \ldots, A_m \\ C_r & \leftarrow & A_1, \ldots, A_m \end{array}$$

where r is the least index such that $I(C_1) = \cdots = I(C_{r-1}) = F^*$ and either $r = n$ or $I(C_r) \neq F^*$.

The $\times$-reduct of P with respect to I is denoted by $P^I_\times$ and is the union of the reducts $R^I_\times$ for all R in P.

The main difference between Definitions 4 and 11 are the rules $C_i \leftarrow F^*, A_1, \ldots, A_m$ included in the reduct of Definition 11. These rules enable F^* to be produced for C_i when $I(A_1) = \cdots = I(A_m) = T$. Otherwise there would be no way for the value F^* to be produced by the reduct.

Definition 12. Let P be an LPOD and M an interpretation of P. We say that M is a (three-valued) *answer set* of P if M is consistent and it is the $\leq$-least model of $P^M_\times$.

The preference relation over the answer sets of LPODs is defined next. Intuitively, the most preferred answer sets of LPODs are those that are more likely to satisfy the top choices in ordered disjunctions. Minimizing with respect to F^* values will accomplish this.

Definition 13. Let P be an LPOD and let M_1, M_2 be answer sets of P. Let M_1^* and M_2^* be the sets of literals in M_1 and M_2 respectively that have the value F^*. We say that M_1 is preferred to M_2, written $M_1 \sqsubset M_2$, if $M_1^* \subset M_2^*$.

Definition 14. An answer set of an LPOD P is called *most preferred* if it is minimal among all the answer sets of P with respect to the $\sqsubset$ relation.

Let's reconsider some of the previous examples under the new semantics.

Example 5. The "cars" program from Example 2 has two answer sets, namely:

$$\begin{array}{lcl} M_1 & = & \{(\texttt{mercedes}, T), (\texttt{bmw}, F), (\texttt{gas\_mercedes}, F^*), \\ & & (\texttt{diesel\_mercedes}, T), (\neg\texttt{gas\_mercedes}, T)\} \\ M_2 & = & \{(\texttt{mercedes}, F^*), (\texttt{bmw}, T), (\texttt{gas\_mercedes}, F^*), \\ & & (\texttt{diesel\_mercedes}, F^*), (\neg\texttt{gas\_mercedes}, T)\} \end{array}$$

According to the $\sqsubset$ ordering, the most preferred answer set is M_1.

Example 6. Consider the "hotels" program from Example 3. Assuming that there are no 3-star hotels in walking distance and and that there are no 2-star hotels outside of walking distance, we get two incomparable answer sets:

$$\begin{aligned}M_1 &= \{(\texttt{walking}, T), (\neg\texttt{walking}, F), (\texttt{3-stars}, F^*), (\texttt{2-stars}, T)\}\\ M_2 &= \{(\texttt{walking}, F^*), (\neg\texttt{walking}, T), (\texttt{3-stars}, T), (\texttt{2-stars}, F)\}\end{aligned}$$

Consider now the modified program containing the unsatisfiable better option of a 4-star hotel). Under the same assumption as above, we get the answer sets:

$$\begin{aligned}M_1' &= \{(\texttt{walking}, T), (\neg\texttt{walking}, F), (\texttt{3-stars}, F^*), (\texttt{2-stars}, T),\\ &\quad (\texttt{4-stars}, F^*), (\neg\texttt{4-stars}, T)\}\\ M_2' &= \{(\texttt{walking}, F^*), (\neg\texttt{walking}, T), (\texttt{3-stars}, T), (\texttt{2-stars}, F),\\ &\quad (\texttt{4-stars}, F^*), (\neg\texttt{4-stars}, T)\}\end{aligned}$$

Under the new approach, these two answer sets are also incomparable.

Additionally, a result is given in [1] that illustrates the relationship between the answer sets of LPODs produced using the original and the new approach.

Definition 15. Let I be a three-valued interpretation of LPOD P. We define *collapse*(I) to be the set of literals L in P such that $I(L) = T$.

Lemma 1. Let P be an LPOD and M be a three-valued answer set of P. Then, *collapse*(M) is an answer set of P according to Definition 5.

Lemma 2. Let N be an answer set of P according to Definition 5. There exists a unique three-valued interpretation M such that $N =$ *collapse*(M) and M is a three-valued answer set of P.

It has been shown that this interpretation M can be reconstructed from N and P. Let $\mathcal{F}$ be the set containing the propositional atoms that have the value F^*. Then, M can be defined as

$$M(A) = \begin{cases} F & A \notin N \text{ and } A \notin \mathcal{F} \\ F^* & A \notin N \text{ and } A \in \mathcal{F} \\ T & A \in N \end{cases}$$

The set $\mathcal{F}$ can be defined as the limit of the sequence $\{\mathcal{F}^i\}_{i<\omega}$ where:

$$\begin{aligned}\mathcal{F}^0 &= \emptyset\\ \mathcal{F}^{n+1} &= \{C_j \mid (C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k) \in P\\ &\qquad \{A_1, \ldots, A_m\} \subseteq N \cup \mathcal{F}^n, \{C_1, \ldots, C_j, B_1, \ldots, B_k\} \cap N = \emptyset\}\\ \mathcal{F} &= \cup_{n<\omega} \mathcal{F}^n\end{aligned}$$

Therefore, there is a bijection between the answer sets produced by the approach proposed in [1] and the original ones. The only difference between a three-valued answer set and the corresponding two-valued one is that some literals of the former have an F^* value instead of an F. These F^* values are the ones that enable us to determine the most preferred answer sets.

3. RELATED WORK

Brewka et al. in [6] demonstrated how LPODs can be implemented using answer set solvers for non-disjunctive programs. Their system *psmodels* involves the execution of two programs, a generator of candidate answer sets and a tester program that decides whether a given candidate answer set is maximally preferred or uses the generator to produce an more preferred one, in case it isn't. Similar to their generator program, our third translation encodes all split programs into a single ASP.

Cabalar implements the translation of LPODs into regular logic programs in [7] using *DLV* as a backend. This implementation also uses a generator and a tester program, but the encoding of generating candidate answer sets is different. The encoding is similar to the one in our final translation.

Lee and Yang presented a translation from LPOD to ASP in [8] using assumption programs. The reduction from LPOD to ASP is a one-pass approach where preferred answer sets are computed by calling an answer set solver once, generating all candidate answer sets, and then applying preference criteria in a single execution, unlike other implementations, which require multiple calls to the answer set solver. Our first translation, similar to theirs, uses a weak constraint to ensure all candidate answer sets are generated by the answer set solver. Since this approach increases the complexity of the computation, we present an alternative second implementation which, while still utilizing assumption programs, uses *asprin* in order to identify the most preferred answer sets.

Our final implementation is most similar to Lee and Yang's *lpod2asprin* [9], which uses Cabalar's method for generating candidate answer sets as well as *asprin* for calculating the satisfaction degrees in each answer set in order to find the most preferred ones. We also use Cabalar's method in our final implementation along with *asprin*, applying the subset preference criterion to the sets of atoms that take the F^* value in each candidate answer set.

4. COMPUTING LPOD ANSWER SETS

In this chapter we explore different approaches in generating candidate answer sets for LPODs, each based on existing ideas.

The first approach uses the concept of *assumption programs*, as introduced by Lee and Yang in [8]. The next approach adopts the concept of *split programs*, used by Brewka in [3]. The final approach utilizes Cabalar's method [7].

4.1 Generating LPOD Answer Sets using Assumption Programs

Consider an LPOD with m ordered disjunctive rules. For each rule i ($i \in \{1, \ldots, m\}$)

$$C_i^1 \times \cdots \times C_i^{n_i} \leftarrow A_i^1, \ldots, A_i^{m_i},\ not\ B_i^1, \ldots,\ not\ B_i^{k_i}$$

its x_i*-th assumption*, denoted by $O_i(x_i)$, is defined as the set of ASP rules

$$\begin{array}{ll}
body_i \leftarrow A_i^1, \ldots, A_i^{m_i},\ not\ B_i^1, \ldots,\ not\ B_i^{k_i} & \\
\quad \bot \leftarrow x = 0,\ body_i & \\
\quad \bot \leftarrow x > 0,\ not\ body_i & \\
\quad C_i^j \leftarrow body_i,\ x = j & \text{(for } 1 \le j \le n_i\text{)} \\
\quad \bot \leftarrow body_i, x \neq j,\ not\ C_i^1, \ldots,\ not\ C_i^{j-1}, C_i^j & \text{(for } 1 \le j \le n_i\text{)}
\end{array}$$

where $body_i$ is an auxiliary atom for each ordered disjunctive rule i, that does not appear in P, and $\bot$ denotes an empty rule head.

The first three rules guarantee that the body of rule i is false iff $x = 0$. The next rule represents C_i^j being true when $x = j$ and the last rule ensures that all $C_i^1, \ldots, C_i^{j-1}$ are false under the same assumption ($x = j$).

An assumption program of an LPOD is obtained by replacing each of its ordered disjunctive rules by one of its assumptions. In an LPOD with m ordered disjunctive rules, if each ordered disjunctive rule i is replaced by its x_i*-th assumption*, we call $(x_1, \ldots, x_m)$ the *assumption degree list* of the assumption program.

The following proposition from [8] states that assumption programs can produce the candidate answer sets.

Proposition 1. For any LPOD P and any set S of atoms, S is a candidate answer set of P iff $S \cup \{body_i \mid S$ satisfies the body of ordered disjunctive rule $i\}$ is an answer set of some assumption program of P.

Example 1 (Continued) The LPOD

$$a \times b \ \leftarrow \ not\ c, \qquad\qquad b \times c \ \leftarrow \ not\ d$$

has the following assumptions:

$$\begin{array}{rrcl}
O_1(X_1): & body_1 & \leftarrow & not\ c \\
& \bot & \leftarrow & X_1 = 0,\ body_1 \\
& \bot & \leftarrow & X_1 > 0,\ not\ body_1 \\
& a & \leftarrow & body_1,\ X_1 = 1 \\
& b & \leftarrow & body_1,\ X_1 = 2 \\
& \bot & \leftarrow & body_1,\ X_1 \neq 1,\ a \\
& \bot & \leftarrow & body_1,\ X_1 \neq 2,\ not\ a, b
\end{array}
\qquad
\begin{array}{rrcl}
O_2(X_2): & body_2 & \leftarrow & not\ d \\
& \bot & \leftarrow & X_2 = 0,\ body_2 \\
& \bot & \leftarrow & X_2 > 0,\ not\ body_2 \\
& b & \leftarrow & body_2,\ X_2 = 1 \\
& c & \leftarrow & body_2,\ X_2 = 2 \\
& \bot & \leftarrow & body_2,\ X_2 \neq 1,\ b \\
& \bot & \leftarrow & body_2,\ X_2 \neq 2,\ not\ b,\ c
\end{array}$$

where X_1 and X_2 are variables that range over $\{0, 1, 2\}$. Therefore, it has 9 assumption programs. It is easy to verify that assumption programs $O_1(1) \cup O_2(1)$, $O_1(2) \cup O_2(1)$ and $O_1(0) \cup O_2(2)$ produce the answer sets $\{a, b\}$, $\{b\}$ and $\{c\}$ respectively, which are exactly the candidate answer sets of the LPOD.

4.2 Generating LPOD Answer Sets using Split Programs

Recall the definition of *split programs* from Chapter 3. Brewka [3, Definition 3] defines the answer sets of an LPOD as the answer sets obtained from its split programs. Formally:

Proposition 2. Let P be an LPOD. A set of literals A is an answer set of P if it is an answer set of a split program P' of P.

Example 1 (Continued) The LPOD P

$$a \times b \ \leftarrow \ not\ c, \qquad\qquad b \times c \ \leftarrow \ not\ d$$

has four split programs:

$$\begin{array}{ll}
a \leftarrow\ not\ c & a \leftarrow\ not\ c \\
b \leftarrow\ not\ d & c \leftarrow\ not\ d,\ not\ b \\
\\
b \leftarrow\ not\ c,\ not\ a & b \leftarrow\ not\ c,\ not\ a \\
b \leftarrow\ not\ d & c \leftarrow\ not\ d,\ not\ b
\end{array}$$

Each of these split programs has the following answer sets respectively:

$$\begin{array}{ll}
\{a, b\} & \{c\} \\
\\
\{b\} & \{b\}, \{c\}
\end{array}$$

These are the candidate answer sets of the LPOD P.

4.3 Generating LPOD Answer Sets using Cabalar's method

In [7], Cabalar proposed the following translation for LPOD rules $C_1 \times \cdots \times C_n \leftarrow Body$:

$$\begin{array}{rl}
C_i \vee\ not\ C_i \leftarrow Body,\ not\ C_1, \ldots,\ not\ C_{i-1} & \text{for } i \in \{1, \ldots, n\} \\
\bot \leftarrow Body,\ not\ C_1, \ldots,\ not\ C_n &
\end{array}$$

where $Body$ denotes the rules body and $\bot$ denotes an empty rule head.

Lee and Yang in [9] encoded the above translation in ASP as:

$$\begin{aligned}
body &\leftarrow Body \\
\{C_i\} &\leftarrow body,\ not\ C_1, \ldots,\ not\ C_{i-1} && \text{for } i \in \{1, \ldots, n-1\} \\
C_n &\leftarrow body,\ not\ C_1, \ldots,\ not\ C_{n-1}
\end{aligned}$$

where $Body$ denotes the rules body and $body$ is an auxiliary atom not appearing in the LPOD.

Example 1 (Continued) The LPOD P

$$a \times b \ \leftarrow\ not\ c, \qquad\qquad b \times c \ \leftarrow\ not\ d$$

is translated to the following ASP:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
body_1 & \leftarrow & not\ c & body_2 & \leftarrow & not\ d \\
\{a\} & \leftarrow & body_1 & \{b\} & \leftarrow & body_2 \\
b & \leftarrow & body_1,\ not\ a & c & \leftarrow & body_2,\ not\ b
\end{array}$$

that has three candidate answer sets, namely $\{a, b\}, \{b\}, \{c\}$.

Notice that the ASP translations in the above sections produce two-valued answer sets, corresponding to the original LPOD semantics. In order to obtain the three-valued ones, we need additional rules that produce the necessary F^* values.

5. TRANSLATING LPOD INTO ASP UNDER NEW SEMANTICS

The purpose of this chapter is to implement the proposed three-valued semantics for LPODs. In traditional approaches, implementation of LPOD semantics relied on answer set solvers. They involved translating LPODs into standard answer set programs, for answer set solvers to produce the most preferred answer sets. Following a similar approach, we present four implementations, based on the ideas in Chapter 4.

First, we must demonstrate a way for the ASP encodings to generate the F^* values for a three-valued answer set M of a LPOD P. Recall the discussion after Lemma 2 and how the set $\mathcal{F}$ of the atoms that have the F^* value can be defined as the limit of a particular sequence.

This sequence can be encoded using the predicates $isFstar/1$ and $isTFstar/1$. For each atom A in LPOD P, $isFstar(A)$ holds when $M(A) = F^*$ and $isTFstar(A)$ holds when $M(A) = T$ or $M(A) = F^*$. We now define the set of ASP rules that produce the necessary F^* values in the LPOD answer sets.

Definition 16. Let P be an LPOD. We use the following set of rules to produce the F^* values in the answer sets of P:

For each ordered disjunctive rule $C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k$ in P and for $i \in \{1, \ldots, n\}$, the rules:

$$body \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k \tag{5.1}$$

$$isFstar(C_i) \leftarrow isTFstar(body),\ isFstar(C_1), \ldots, isFstar(C_{i-1}),\ not\ C_i \tag{5.2}$$

$$isFstar(body) \leftarrow not\ body,\ isTFstar(A_1), \ldots, isTFstar(A_m),\ not\ B_1, \ldots,\ not\ B_k \tag{5.3}$$

where $body$ is an auxiliary atom not appearing in P.

For each regular rule $C \leftarrow A_1, \ldots, A_m,\ not\ B_1, \ldots,\ not\ B_k$ in P, the rule:

$$isFstar(C) \leftarrow not\ C,\ isTFstar(A_1), \ldots, isTFstar(A_m),\ not\ B_1, \ldots,\ not\ B_k \tag{5.4}$$

For each atom A in P as well as the auxiliary $body$ atoms, the rules:

$$isTFstar(A) \leftarrow isFstar(A) \tag{5.5}$$

$$isTFstar(A) \leftarrow A \tag{5.6}$$

Rules (5.2) produce the F^* value for the i-th atom in the rule's head if it cannot be satisfied and all previous head atoms also take the F^* value. Rule (5.3) produces the F^* value for the auxiliary $body$ atom if the rule's body is not satisfied but none of the body literals evaluate to F (at least one evaluates to F^* and the others may take the T value as well). Rule (5.4) produces the F^* value for regular rule heads and it is similar to rule (5.3). Rules (5.5) and (5.6) define the $isTFstar$ predicate.

We can now present our four implementations of the three-valued semantics. The first two implementations adopt the concept of *assumption programs*. The third implementation uses split programs and the final one utilizes Cabalar's method for generating answer sets.

5.1 Translating LPOD into ASP using Assumption Programs and Weak Constraints

Our first implementation is based on Lee and Yang's [8] translation of LPODs into ASP using assumption programs and a weak constraint. The purpose of the weak constraint is to produce all candidate answer sets in a single execution. The ASP encoding contains rules to compare the answer sets and identify the most preferred ones.

Definition 17. Let P be an LPOD with m ordered disjunctive rules. The translation of P contains the following rules:

1. Two rules for declaring and computing all assumption programs:

$$\{ap(X_1, \dots, X_m) : X_1 = 0..n_1, \dots, X_m = 0..n_m\}. \tag{5.7}$$

$$:\sim ap(X_1, \dots, X_m).\ [-1, (X_1, \dots, X_m)] \tag{5.8}$$

where $X_1, \dots, X_m$ are variables and $(X_1, \dots, X_m)$ represents the assumption degree list of the corresponding assumption program. Rule (5.8) is a weak constraint which ensures that all assumption programs will be considered, therefore all candidate answer sets will be computed in order to extract the most preferred ones.

2. For each regular rule $C \leftarrow Body$ in P, the rule:

$$C(X_1, \dots, X_m) \leftarrow ap(X_1, \dots, X_m), Body(X_1, \dots, X_m). \tag{5.9}$$

where $Body$ denotes the rule's body and $Body(X_1, \dots, X_m)$ is obtained by replacing each atom A in $Body$ with $A(X_1, \dots, X_m)$.

3. For each ordered disjunctive rule $C_i^1 \times \ldots \times C_i^{n_i} \leftarrow Body_i$ in P, the rules:

$$body_i(X_1, \dots, X_m) \leftarrow ap(X_1, \dots, X_m), Body_i(X_1, \dots, X_m). \tag{5.10}$$

$$\bot \leftarrow ap(X_1, \dots, X_m), X_i = 0, body_i(X_1, \dots, X_m). \tag{5.11}$$

$$\bot \leftarrow ap(X_1, \dots, X_m), X_i > 0, not\ body_i(X_1, \dots, X_m). \tag{5.12}$$

And for $1 \leq r \leq n_i$, the rules:

$$C_i^r(X_1, \dots, X_m) \leftarrow body_i(X_1, \dots, X_m), X_i = r. \tag{5.13}$$

$$\begin{aligned}\bot \leftarrow\ & body_i(X_1, \dots, X_m), X_i \neq r, \\ & not\ C_i^1(X_1, \dots, X_m), \dots,\ not\ C_i^{r-1}(X_1, \dots, X_m), C_i^r(X_1, \dots, X_m).\end{aligned} \tag{5.14}$$

4. For each ordered disjunctive rule $C_i^1 \times \ldots \times C_i^{n_i} \leftarrow A_i^1, \dots, A_i^{a_i},\ not\ B_i^1, \dots, not\ B_i^{b_i}$ in P and for $j \in \{1, \dots, n\}$, the rules:

$$\begin{aligned}isFstar(C_i^j(X_1, \dots, X_m)) \leftarrow\ & ap(X_1, \dots, X_m), isTFstar(body_i(X_1, \dots, X_m)), \\ & isFstar(C_i^1(X_1, \dots, X_m)), \dots, isFstar(C_i^{j-1}(X_1, \dots, X_m)),\ not\ C_i^j(X_1, \dots, X_m).\end{aligned} \tag{5.15}$$

$$\begin{aligned}isFstar(body_i(X_1, \dots, X_m)) \leftarrow\ & ap(X_1, \dots, X_m), not\ body_i(X_1, \dots, X_m), \\ & isTFstar(A_i^1(X_1, \dots, X_m)), \dots, isTFstar(A_i^{a_i}(X_1, \dots, X_m)), \\ & not\ B_i^1(X_1, \dots, X_m), \dots,\ not\ B_i^{b_i}(X_1, \dots, X_m).\end{aligned} \tag{5.16}$$

5. For each regular rule $C \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b$ in P, the rule:

$$\begin{aligned} isFstar(C(X_1,\ldots,X_m)) \leftarrow\ & ap(X_1,\ldots,X_m), not\ C(X_1,\ldots,X_m),\\ & isTFstar(A_1(X_1,\ldots,X_m)),\ldots,isTFstar(A_a(X_1,\ldots,X_m)),\\ & not\ B_1(X_1,\ldots,X_m),\ldots,\ not\ B_b(X_1,\ldots,X_m). \end{aligned} \tag{5.17}$$

6. For each atom A in P as well as the auxiliary $body_i$ atoms, the rules:

$$isTFstar(A(X_1,\ldots,X_m)) \leftarrow isFstar(A(X_1,\ldots,X_m)). \tag{5.18}$$

$$isTFstar(A(X_1,\ldots,X_m)) \leftarrow A(X_1,\ldots,X_m). \tag{5.19}$$

Rules (5.10)-(5.14) for each ordered disjunctive rule in the LPOD correspond to its assumption (and are also found in [8]). The rules (5.15)-(5.19) from Definition 16 produce the necessary F^* values.

Up to this point, the translation can produce the (three-valued) candidate answer sets of the LPOD. In order to be able to compare them, we need rules that define the preference relation of Definition 13. Namely, we need rules that produce the M^* set of each candidate answer set M and also define the subset relation between these M^* sets:

7. For all atoms A that appear in rule heads:

$$isFstar(X_1,...,X_m,A) \leftarrow isFstar(A(X_1,...,X_m)). \tag{5.20}$$

As well as:

$$\begin{aligned} count\_intersection((X_1,\ldots,X_m),(Y_1,\ldots,Y_m),Cnt) \leftarrow\ & ap(X_1,\ldots,X_m),\\ ap(Y_1,\ldots,Y_m), Cnt = \#count\{A : isFstar(X_1,\ldots,X_m,A), & isFstar(Y_1,\ldots,Y_m,A)\}. \end{aligned} \tag{5.21}$$

$$count\_set((X_1,\ldots,X_m),C) \leftarrow ap(X_1,\ldots,X_m), C = \#count\{A : isFstar(X_1,\ldots,X_m,A)\}. \tag{5.22}$$

$$\begin{aligned} subset(S_1,S_2) \leftarrow\ & count\_intersection(S_1,S_2,C_I), count\_set(S_1,C_1),\\ & count\_set(S_2,C_2), C_I = C_1, C_1 < C_2. \end{aligned} \tag{5.23}$$

$$pAS(X_1,\ldots,X_m) \leftarrow ap(X_1,\ldots,X_m), \{subset(P,(X_1,\ldots,X_m))\}\ 0. \tag{5.24}$$

Rules (5.20)-(5.23) define the $\sqsubset$ relation utilizing the following equivalence: "$A \subseteq B \iff |A| = |A \cap B|$". Rules (5.21) and (5.22) are used to count the number of F^* atoms in the intersection of two answer sets and an answer set respectively. Rule (5.23) basically requires $|M_1^*| = |M_1^* \cap M_2^*|$ and $|M_1^*| < |M_2^*|$ for answer sets M_1, M_2 in order for "$M_1 \sqsubset M_2$" to hold. Rule (5.24) defines the answer set corresponding to the assumption program with assumption degree list $(X_1,\ldots,X_m)$ to be the *most preferred* one if it is minimal among all candidate answer sets with respect to the $\sqsubset$ relation.

Example 1 (Continued) The LPOD

$$a \times b \ \leftarrow\ not\ c, \qquad\qquad b \times c \ \leftarrow\ not\ d$$

is translated into the following ASP (in the input language of *clingo*):

```
{ap(X1,X2): X1 = 0..2, X2 = 0..2}.
#minimize {-1,(X1,X2):ap(X1,X2)}.

% a * b <- not c.
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).

isFstar(a(X1,X2)) :- ap(X1,X2), isTFstar(body_1(X1,X2)), not a(X1,X2).
isFstar(b(X1,X2)) :- ap(X1,X2), isTFstar(body_1(X1,X2)), isFstar(a(X1,X2)),
    not b(X1,X2).
isFstar(body_1(X1,X2)) :- ap(X1,X2), not body_1(X1,X2), not c(X1,X2).

% b * c <- not d.
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).

isFstar(b(X1,X2)) :- ap(X1,X2), isTFstar(body_2(X1,X2)), not b(X1,X2).
isFstar(c(X1,X2)) :- ap(X1,X2), isTFstar(body_2(X1,X2)), isFstar(b(X1,X2)),
    not c(X1,X2).
isFstar(body_2(X1,X2)) :- ap(X1,X2), not body_2(X1,X2), not d(X1,X2).

% isTFstar rules
isTFstar(a(X1,X2)) :- isFstar(a(X1,X2)).
isTFstar(a(X1,X2)) :- a(X1,X2).
isTFstar(b(X1,X2)) :- isFstar(b(X1,X2)).
isTFstar(b(X1,X2)) :- b(X1,X2).
isTFstar(c(X1,X2)) :- isFstar(c(X1,X2)).
isTFstar(c(X1,X2)) :- c(X1,X2).
isTFstar(d(X1,X2)) :- isFstar(d(X1,X2)).
isTFstar(d(X1,X2)) :- d(X1,X2).

isTFstar(body_1(X1,X2)) :- isFstar(body_1(X1,X2)).
isTFstar(body_1(X1,X2)) :- body_1(X1,X2).
isTFstar(body_2(X1,X2)) :- isFstar(body_2(X1,X2)).
isTFstar(body_2(X1,X2)) :- body_2(X1,X2).

% make M* sets
isFstar(X1,X2, a) :- isFstar(a(X1,X2)).
isFstar(X1,X2, b) :- isFstar(b(X1,X2)).
isFstar(X1,X2, c) :- isFstar(c(X1,X2)).
```

```
% preference rules
count_intersection((X1,X2),(Y1,Y2), Cnt) :- ap(X1,X2), ap(Y1,Y2),
    Cnt = #count{A : isFstar(X1,X2,A), isFstar(Y1,Y2,A)}.
count_set((X1,X2), Cnt) :- ap(X1,X2), Cnt = #count{A : isFstar(X1,X2,A)}.
subset(S1,S2) :- count_intersection(S1,S2,CI), count_set(S1,C1), count_set(S2,C2),
    CI = C1, C1 < C2.
pAS(X1,X2) :- ap(X1,X2), {subset(P,(X1,X2))}0.
```

The optimal answer set of the translation contains the following atoms:

```
ap(0,2) body_2(0,2) isTFstar(body_2(0,2)) c(0,2) isTFstar(c(0,2)) isFstar(b(0,2))
isTFstar(b(0,2))

ap(1,1) body_1(1,1) isTFstar(body_1(1,1)) body_2(1,1) isTFstar(body_2(1,1)) a(1,1)
isTFstar(a(1,1)) b(1,1) isTFstar(b(1,1))

ap(2,1) body_1(2,1) isTFstar(body_1(2,1)) body_2(2,1) isTFstar(body_2(2,1))
isFstar(a(2,1)) isTFstar(a(2,1)) b(2,1) isTFstar(b(2,1))

isFstar(0,2,b) isFstar(2,1,a)

count_set((0,2),1) count_set((1,1),0) count_set((2,1),1)

count_intersection((0,2),(0,2),1) count_intersection((1,1),(0,2),0)
count_intersection((2,1),(0,2),0) count_intersection((0,2),(1,1),0)
count_intersection((1,1),(1,1),0) count_intersection((2,1),(1,1),0)
count_intersection((0,2),(2,1),0) count_intersection((1,1),(2,1),0)
count_intersection((2,1),(2,1),1)

subset((1,1),(0,2)) subset((1,1),(2,1))

pAS(1,1)
```

Assumption programs with degree lists $(0,2)$, $(1,1)$, $(2,1)$ correspond to the answer sets $\{(a,F),(b,F^*),(c,T),(d,F)\}$, $\{(a,T),(b,T),(c,F),(d,F)\}$, $\{(a,F^*),(b,T),(c,F),(d,F)\}$ respectively, which are exactly the candidate answer sets of the LPOD. The auxiliary $body$ atoms are produced, as well as atoms related to the preference relation between candidate answer sets. The presence of the $pAS(1,1)$ atom suggests that the answer set corresponding to the assumption program $(1,1)$, namely $\{(a,T),(b,T),(c,F),(d,F)\}$, is the most preferred answer set of the LPOD.

5.2 Translating LPOD into ASP using Assumption Programs and asprin

As Lee and Yang point out in [8], using a translation that computes all assumption programs makes the size of the resulting program exponential to the number of ordered disjunctive rules. And since assumption programs are independent of one another, we could consider computing them separately. They also mention *asprin* [10] as a potential tool for expressing and computing preferences over answer sets.

Rule (5.7) of the translation generates a set of $ap(x_1,\ldots,x_m)$ atoms, each of which corresponds to a distinct assumption program characterized by its assumption degree list

$(x_1, \ldots, x_m)$. Rule (5.8) is a weak constraint that maximizes the number of ap/m atoms by adding a negative penalty of -1 for each ap atom in the translation program's answer set, i.e. for each assumption program considered in the translation. Rule (5.7) enables the *clingo* solver to produce several answer sets, but rule (5.8) restricts the optimal answer sets to only one, that is the one that includes all candidate answer sets of the translated LPOD. We realize that this approach undoubtedly leads to many unnecessary computations.

In our next approach, we alter the translation so it generates one candidate answer set at a time and then we utilize *asprin* in order to express the $\sqsubset$ relation, which enables us to distinguish the most preferred answer sets. Specifically, we change rule (5.7) to:

$$\{ap(X_1, \ldots, X_m) : X_1 = 0..n_1, \ldots, X_m = 0..n_m\} = 1. \tag{5.25}$$

and we omit rule (5.8). Rule (5.25) ensures all models the solver produces contain exactly one assumption program and therefore one candidate answer set of the translated LPOD. And since rules (5.20)-(5.24) were used to compare candidate answer sets produced in the same model of the translation, we also don't need them anymore. Instead, since we use *asprin* that compares candidate answer sets across models of the translation, we need new rules that produce the M^* sets for each answer set:

For all atoms that appear in rule heads:

$$mstar(A) \leftarrow isFstar(A(X_1, ..., X_m)). \tag{5.26}$$

The predicate $mstar/1$ representing the set of atoms that get the F^* value in an assumption program without denoting its assumption degree list facilitates the comparison of different candidate answers by *asprin* based on their M^* sets.

Definition 18. Rules (5.25), (5.9)-(5.19) and (5.26) constitute the *base* program of the translation of an LPOD with m ordered disjunctive rules. In order to compare the answer sets, we use the *asprin* rules:

$$\#preference(p, subset)\{mstar(X)\}. \tag{5.27}$$
$$\#optimize(p). \tag{5.28}$$

In rule (5.27) the preference statement p defines a $subset$ preference over the $mstar$ atoms. Rule (5.28) instructs *asprin* to compute answer sets that are optimal with respect to p.

Example 1 (Continued) The LPOD

$$a \times b \;\leftarrow\; not\ c, \qquad\qquad b \times c \;\leftarrow\; not\ d$$

is translated into the following ASP:

```
{ap(X1,X2): X1 = 0..2, X2 = 0..2} = 1.

% a * b <- not c.
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).

isFstar(a(X1,X2)) :- ap(X1,X2), isTFstar(body_1(X1,X2)), not a(X1,X2).
isFstar(b(X1,X2)) :- ap(X1,X2), isTFstar(body_1(X1,X2)), isFstar(a(X1,X2)),
    not b(X1,X2).
isFstar(body_1(X1,X2)) :- ap(X1,X2), not body_1(X1,X2), not c(X1,X2).

% b * c <- not d.
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).

isFstar(b(X1,X2)) :- ap(X1,X2), isTFstar(body_2(X1,X2)), not b(X1,X2).
isFstar(c(X1,X2)) :- ap(X1,X2), isTFstar(body_2(X1,X2)), isFstar(b(X1,X2)),
    not c(X1,X2).
isFstar(body_2(X1,X2)) :- ap(X1,X2), not body_2(X1,X2), not d(X1,X2).

% isTFstar rules
isTFstar(a(X1,X2)) :- isFstar(a(X1,X2)).
isTFstar(a(X1,X2)) :- a(X1,X2).
isTFstar(b(X1,X2)) :- isFstar(b(X1,X2)).
isTFstar(b(X1,X2)) :- b(X1,X2).
isTFstar(c(X1,X2)) :- isFstar(c(X1,X2)).
isTFstar(c(X1,X2)) :- c(X1,X2).
isTFstar(d(X1,X2)) :- isFstar(d(X1,X2)).
isTFstar(d(X1,X2)) :- d(X1,X2).
isTFstar(body_1(X1,X2)) :- isFstar(body_1(X1,X2)).
isTFstar(body_1(X1,X2)) :- body_1(X1,X2).
isTFstar(body_2(X1,X2)) :- isFstar(body_2(X1,X2)).
isTFstar(body_2(X1,X2)) :- body_2(X1,X2).

% make M* set
mstar(a) :- isFstar(a(X1,X2)).
mstar(b) :- isFstar(b(X1,X2)).
mstar(c) :- isFstar(c(X1,X2)).
```

Asprin produces the following two candidate answer sets:

Answer Set 1

```
ap(2,1) body_1(2,1) b(2,1) isTFstar(body_1(2,1)) isTFstar(body_2(2,1))
isTFstar(b(2,1)) isTFstar(a(2,1)) isFstar(a(2,1)) body_2(2,1) mstar(a)
```

Answer Set 2 (declared ”optimum”)

```
ap(1,1) body_1(1,1) a(1,1) b(1,1) isTFstar(body_1(1,1)) isTFstar(b(1,1))
```

```
isTFstar(body_2(1,1)) isTFstar(a(1,1)) body_2(1,1)
```

Notice that not all candidate answer sets of the LPOD are produced by *asprin* (in this case, the answer set corresponding to assumption program $(0, 2)$ is missing). This is due to *asprin*'s process to find the most preferred answer sets. The execution begins by calling *clingo* once, to produce an initial model X and then checks if there is another model Y that is more preferred to X according to the specified preference criterion. If such model Y exists, X is replaced by Y. This process is continued until there is no such Y, therefore the current X is considered one of the "most preferred" models. To find all most "preferred answer" sets, *asprin* repeats the above procedure, each time with a new initial model.

In the program of Example 1, the *clingo* model which includes $ap(0, 2)$ is not produced by *asprin*, since it produces the model including $ap(1, 1)$ first, which is preferred to the former, according to the $\sqsubset$ relation. Notice that when the translation isn't solved with *asprin*, but just with *clingo* (only the base part is considered), then all candidate answer sets are produced. Running just *clingo* on the translation of Example 1, we get:

Answer Set 1

```
body_2(2,1) ap(2,1) body_1(2,1) isTFstar(body_1(2,1)) isFstar(a(2,1)) b(2,1)
isTFstar(body_2(2,1)) mstar(a) isTFstar(b(2,1)) isTFstar(a(2,1))
```

Answer Set 2

```
body_2(1,1) ap(1,1) body_1(1,1) isTFstar(body_1(1,1)) a(1,1) b(1,1)
isTFstar(body_2(1,1)) isTFstar(b(1,1)) isTFstar(a(1,1))
```

Answer Set 3

```
body_2(0,2) ap(0,2) c(0,2) isFstar(b(0,2)) isTFstar(body_2(0,2)) mstar(b)
isTFstar(c(0,2)) isTFstar(b(0,2))
```

Notice that the *clingo* answer set containing $ap(0, 2)$ is produced after the answer set corresponding to the most preferred answer set of the LPOD, according to the $\sqsubset$ relation.

5.3 Translating LPOD into ASP using Split Programs and asprin

In [6], LPODs are implemented using *psmodels*, which is a modification of *smodels* that allows computing preferred answer sets under the LPOD semantics. The implementation involves the execution of two programs, a generator that produces candidate answer sets, as well as a tester that checks whether a given candidate answer set is the most preferred under the specified preference relation, and produces a more preferred answer set if it is not. Brewka et al. in [6] define LPOD answer sets based on the notion of a split program.

Our next translation is based on the approach in [6]. Instead of the ap atoms corresponding to assumption programs, in this translation there exist $op(i, k)$ atoms indicating that the k-th option of the i-th ordered disjunctive rule is used.

In this approach, as with our previous approach, we have a base program to generate candidate answer sets (each corresponding to one of the LPOD's split programs). We use *asprin* again to extract the most preferred ones.

Definition 19. Let P be an LPOD with m ordered disjunctive rules. The translation of P contains the following rules:

1. Every regular rule $C \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b$ in P.

2. For each ordered disjunctive rule $C_i^1 \times \ldots \times C_i^{n_i} \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b$ in P, the rules:

$$\{op(i, X_i) : X_i = 0, \ldots, n_i\} = 1 \tag{5.29}$$
$$body_i \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b \tag{5.30}$$
$$\leftarrow op(i, 0), body_i \tag{5.31}$$
$$\leftarrow op(i, X_i), X_i > 0, not\ body_i \tag{5.32}$$

And for $1 \leq k \leq n_i$, the rules:

$$C_i^k \leftarrow op(i, k), not\ C_i^1, \ldots, not\ C_i^{k-1}, body_i \tag{5.33}$$
$$\leftarrow C_i^k, not\ op(i, k), not\ C_i^1, \ldots, not\ C_i^{k-1}, body_i \tag{5.34}$$

where X_i is a variable and $body_i$ is an auxiliary atom not appearing in P.

3. For each ordered disjunctive rule $C_i^1 \times \ldots \times C_i^{n_i} \leftarrow A_i^1, \ldots, A_i^{a_i},\ not\ B_i^1, \ldots,\ not\ B_i^{b_i}$ in P and for $j \in \{1, \ldots, n_i\}$, the rules:

$$isFstar(C_i^j) \leftarrow isTFstar(body_i),\ isFstar(C_i^1), \ldots, isFstar(C_i^{j-1}),\ not\ C_i^j \tag{5.35}$$
$$isFstar(body_i) \leftarrow not\ body_i,\ isTFstar(A_i^1), \ldots, isTFstar(A_i^{a_i}),\ not\ B_i^1, \ldots,\ not\ B_i^{b_i} \tag{5.36}$$

4. For each regular rule $C \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b$ in P, the rule:

$$isFstar(C) \leftarrow not\ C,\ isTFstar(A_1), \ldots, isTFstar(A_a),\ not\ B_1, \ldots,\ not\ B_b \tag{5.37}$$

5. For each atom A in P as well as the auxiliary $body_i$ atoms, the rules:

$$isTFstar(A) \leftarrow isFstar(A) \tag{5.38}$$
$$isTFstar(A) \leftarrow A \tag{5.39}$$

6. In order to find the most preferred answer sets, the rules:

For all atoms A that appear in rule heads:

$$mstar(A) \leftarrow isFstar(A) \tag{5.40}$$

As well as the *asprin* preference rules (same as before):

$$\#preference(p, subset)\{mstar(X)\}. \tag{5.41}$$
$$\#optimize(p). \tag{5.42}$$

Example 1 (Continued) The LPOD

$$a \times b \ \leftarrow\ not\ c, \qquad\qquad b \times c \ \leftarrow\ not\ d$$

is translated to the following ASP:

```
% a * b <- not c.
{op(1,X1) : X1 = 0..2} = 1.
body_1 :- not c.
:- op(1,0), body_1.
:- op(1,X1), X1 > 0, not body_1.

a :- body_1, op(1,1).
b :- body_1, op(1,2), not a.
:- body_1, not op(1,1), a.
:- body_1, not op(1,2), not a, b.

isFstar(a) :- isTFstar(body_1), not a.
isFstar(b) :- isTFstar(body_1), isFstar(a), not b.
isFstar(body_1) :- not body_1, not c.

% b * c <- not d.
{op(2,X2) : X2 = 0..2} = 1.
body_2 :- not d.
:- op(2,0), body_2.
:- op(2,X2), X2 > 0, not body_2.

b :- body_2, op(2,1).
c :- body_2, op(2,2), not b.
:- body_2, not op(2,1), b.
:- body_2, not op(2,2), not b, c.

isFstar(b) :- isTFstar(body_2), not b.
isFstar(c) :- isTFstar(body_2), isFstar(b), not c.
isFstar(body_2) :- not body_2, not d.

% isTFstar rules
isTFstar(a) :- isFstar(a).
isTFstar(a) :- a.
isTFstar(b) :- isFstar(b).
isTFstar(b) :- b.
isTFstar(c) :- isFstar(c).
isTFstar(c) :- c.
isTFstar(d) :- isFstar(d).
isTFstar(d) :- d.

isTFstar(body_1) :- isFstar(body_1).
isTFstar(body_1) :- body_1.
isTFstar(body_2) :- isFstar(body_2).
isTFstar(body_2) :- body_2.

% make M* set
mstar(a) :- isFstar(a).
mstar(b) :- isFstar(b).
mstar(c) :- isFstar(c).
```

Asprin produces the following two candidate answer sets:

Answer Set 1

```
op(1,0) op(2,2) c isTFstar(body_2) isTFstar(c) isTFstar(b) isFstar(b)
body_2 mstar(b)
```

Answer Set 2 (declared ”optimum”)

```
op(1,1) op(2,1) body_1 a b isTFstar(body_1) isTFstar(body_2) isTFstar(b)
isTFstar(a) body_2
```

Running just *clingo* on the translation of Example 1, we get:

Answer Set 1

```
isTFstar(body_2) body_2 c op(2,2) isFstar(b) mstar(b) isTFstar(c)
isTFstar(b) op(1,0)
```

Answer Set 2

```
isTFstar(body_2) body_2 b op(2,1) body_1 op(1,2) isTFstar(body_1)
isFstar(a) mstar(a) isTFstar(b) isTFstar(a)
```

Answer Set 3

```
isTFstar(body_2) body_2 b op(2,1) body_1 a op(1,1) isTFstar(body_1)
isTFstar(b) isTFstar(a)
```

Notice that in this example, the $mstar$ atoms may seem redundant. However, instructing *asprin* to compute the subset minimal answer sets M with respect to the $isFstar$ predicate wouldn't necessarily be correct, since we could end up with $body_i$ atoms in M^* sets. The potential existence of auxiliary atoms in the M^* sets prevents us from relying on this preference relation.

5.4 Translating LPOD into ASP using Cabalar's method and asprin

Our final approach is based on Lee and Yang's [8] ASP encoding of Cabalar's [7] method of generating candidate asnwer sets. The apparent difference between the previous approaches and this one is the absence of predicates such as *ap* and *op* that declare which head atom in each ordered disjunctive rule satisfies it. Instead of using a choice rule for such predicates in order to generate the candidate answer sets, choice rules are used for generating the true atoms in the heads of ordered disjunctive rules.

Definition 20. Let P be an LPOD with m ordered disjunctive rules. The translation of P contains the following rules:

1. Every regular rule $C \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b$ in P.

2. For each ordered disjunctive rule $C_i^1 \times \ldots \times C_i^{n_i} \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b$ in P, the rules:

$$body_i \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b \tag{5.43}$$

For $1 \leq k \leq n_i - 1$:

$$\{C_i^k\} \leftarrow body_i, not\ C_i^1, \ldots, not\ C_i^{k-1} \tag{5.44}$$

And the rule:

$$C_i^{n_i} \leftarrow body_i, not\ C_i^1, \ldots, not\ C_i^{n_i-1} \tag{5.45}$$

where $body$ is an auxiliary atom not appearing in P.

3. For each ordered disjunctive rule $C_i^1 \times \ldots \times C_i^{n_i} \leftarrow A_i^1, \ldots, A_i^{a_i},\ not\ B_i^1, \ldots,\ not\ B_i^{b_i}$ in P and for $j \in \{1, \ldots, n_i\}$, the rules:

$$isFstar(C_i^j) \leftarrow isTFstar(body_i),\ isFstar(C_i^1), \ldots, isFstar(C_i^{j-1}),\ not\ C_i^j \tag{5.46}$$

$$isFstar(body_i) \leftarrow not\ body_i,\ isTFstar(A_i^1), \ldots, isTFstar(A_i^{a_i}),\ not\ B_i^1, \ldots,\ not\ B_i^{b_i} \tag{5.47}$$

4. For each regular rule $C \leftarrow A_1, \ldots, A_a,\ not\ B_1, \ldots,\ not\ B_b$ in P, the rule:

$$isFstar(C) \leftarrow not\ C,\ isTFstar(A_1), \ldots, isTFstar(A_a),\ not\ B_1, \ldots,\ not\ B_b \tag{5.48}$$

5. For each atom A in P as well as the auxiliary $body$ atoms, the rules:

$$isTFstar(A) \leftarrow isFstar(A) \tag{5.49}$$

$$isTFstar(A) \leftarrow A \tag{5.50}$$

6. In order to find the most preferred answer sets, the rules:

For all atoms A that appear in rule heads:

$$mstar(A) \leftarrow isFstar(A) \tag{5.51}$$

As well as the *asprin* preference rules (same as before):

$$\#preference(p, subset)\{mstar(X)\}. \tag{5.52}$$

$$\#optimize(p). \tag{5.53}$$

Example 1 (Continued) The LPOD

$$a \times b \ \leftarrow\ not\ c, \qquad\qquad b \times c \ \leftarrow\ not\ d$$

is translated to the following ASP:

```
% a * b <- not c.
body_1 :- not c.
{a} :- body_1.
b :- body_1, not a.

isFstar(a) :- isTFstar(body_1), not a.
isFstar(b) :- isTFstar(body_1), isFstar(a), not b.
isFstar(body_1) :- not body_1, not c.

% b * c <- not d.
body_2 :- not d.
{b} :- body_2.
c :- body_2, not b.

isFstar(b) :- isTFstar(body_2), not b.
isFstar(c) :- isTFstar(body_2), isFstar(b), not c.
isFstar(body_2) :- not body_2, not d.

% isTFstar rules
isTFstar(a) :- isFstar(a).
isTFstar(a) :- a.
isTFstar(b) :- isFstar(b).
isTFstar(b) :- b.
isTFstar(c) :- isFstar(c).
isTFstar(c) :- c.
isTFstar(d) :- isFstar(d).
isTFstar(d) :- d.

isTFstar(body_1) :- isFstar(body_1).
isTFstar(body_1) :- body_1.
isTFstar(body_2) :- isFstar(body_2).
isTFstar(body_2) :- body_2.

% make M* set
mstar(a) :- isFstar(a).
mstar(b) :- isFstar(b).
mstar(c) :- isFstar(c).
```

Asprin produces the following two candidate answer sets:

Answer Set 1

```
c isTFstar(body_2) isTFstar(c) isTFstar(b) isFstar(b) body_2 mstar(b)
```

Answer Set 2 (declared ”optimum”)

```
body_1 a b isTFstar(body_1) isTFstar(body_2) isTFstar(b) isTFstar(a) body_2
```

Running just *clingo* on the translation of Example 1, we get:

Answer Set 1

```
isTFstar(body_2) body_2 c isFstar(b) mstar(b) isTFstar(c) isTFstar(b)
```

Answer Set 2

```
isTFstar(body_2) body_2 b body_1 isTFstar(body_1) isFstar(a) mstar(a)
isTFstar(b) isTFstar(a)
```

Answer Set 3

```
isTFstar(body_2) body_2 b body_1 a isTFstar(body_1) isTFstar(b) isTFstar(a)
```

The answer sets are almost identical with the ones of the previous approach, excluding the op atoms. Again, the issue with using $isFstar$ in *asprin*'s preference relation may not seem obvious in this example, but it is indeed existent as long as we consider the auxiliary $body$ atoms.

6. EXPERIMENTS

In this chapter, we present a performance comparison between the aforementioned translation approaches as well as with other systems, that implement the original LPOD semantics.

In order to demonstrate the superiority of the *asprin*-based approaches compared to the one using weak constraints, we conduct several tests using LPODs of differing characteristics, such the total number of rules, the number of ordered disjunctive rules (OD Rules), the number of atoms, and the number of candidate and preferred answer sets. The experiments were conducted on a Linux machine equipped with an 8-core CPU clocked at 3.8 GHz and 16 GB of RAM. For the experiments, we used *asprin* 3.1.1 and *clingo* 5.4.0. The characteristics of the test programs and the execution times (in seconds) of our proposed translations are displayed in the following table:

Table 6.1: Execution time (in seconds) comparison between our proposed translations.

#	Atoms / Rules / OD Rules / Max OD	Answer Sets / Preferred	t_1	t_2	t_3	t_4
P1	8 / 5 / 1 / 2	1 / 1	0.001	0.056	0.052	0.058
P2	23 / 30 / 2 / 2	3 / 1	0.019	0.071	0.062	0.064
P3	108 / 10 / 1 / 2	1 / 1	0.084	0.179	0.114	0.119
P4	22 / 30 / 3 / 3	2 / 1	0.301	0.085	0.063	0.061
P5	35 / 1 / 1 / 34	34 / 1	2.393	0.080	0.105	0.102
P6	94 / 200 / 3 / 2	2 / 1	4.126	4.386	0.768	0.787
P7	10 / 5 / 5 / 2	24 / 1	17.040	0.072	0.063	0.060
P8	11 / 6 / 5 / 2	31 / 5	20.214	0.087	0.072	0.065
P9	11 / 6 / 5 / 2	24 / 2	20.217	0.076	0.065	0.060

The above results highlight the performance difference between the first approach and the others, with the execution times varying considerably. As expected, the most influential parameters are the number of ordered disjunctive rules (m) and the number of candidate and preferred answer sets. Higher values for m lead to a substantially larger translation of the LPOD. Also, *clingo* must compute all candidate answer sets and compare them in a one-pass execution, therefore a LPOD with multiple candidate and preferred answer sets increases the complexity of this computation. On the other hand, the number of atoms in a LPOD seem to influence the first two implementations equally, that is the implementations using assumption programs. The other two implementations, using split programs and Cabalar's method for generating answer sets respectively, appear to have similar performance in the tests of Table 6.1.

In order to effectively compare our last two translations as well as test their performance against existing systems, we conduct new experiments using primarily more computationally intensive test programs, that were randomly generated. These programs vary in number of atoms, total and ordered disjunctive rules, and maximum disjunction lengths. Table 6.2 displays the characteristics of each test. All executions with *psmodels* and *lpods2asprin* were performed under the inclusion preference relation.

Overall, the implementation using Cabalar's method for generating answer sets seems to be the best of our approaches, since it performs better than the one utilizing split programs in the experiments of Table 6.2, which involves more resource-heavy programs.

Table 6.2: Execution time (in seconds) comparison between existing LPOD systems and two of our translations.

#	Atoms / Rules / OD Rules / Max OD	Answer Sets / Preferred	psmodels	lpod2asprin	t_3	t_4
P10	13 / 5 / 2 / 8	1 / 1	0.001	0.060	0.055	0.055
P11	231 / 5 / 2 / 200	1 / 1	0.004	0.344	6.325	6.139
P12	36 / 18 / 18 / 2	512 / 512	0.042	42.878	4.655	2.906
P13	41 / 82 / 61 / 34	9603 / 1	176.415	0.518	0.781	0.590
P14	36 / 27 / 18 / 2	19683 / 512	2.984	78.591	9.906	6.434
P15	39 / 20 / 19 / 2	524287 / 19	96.097	0.812	0.360	0.328
P16	41 / 21 / 20 / 2	1048575 / 20	209.807	1.043	0.413	0.367

For programs with fewer candidate answer sets (such as P10, P11), *psmodels* clearly outperforms both *lpods2asprin* and our implementations, taking only a few milliseconds to finish their execution. The advantage is probably due to the optimizations made within *psmodels* specifically for LPODs, as opposed to the other systems that rely on general-purpose solvers (*asprin*, *clingo*). However, *psmodels* performs worse as the number of candidate answer sets increases, like in the case of P13. This decline is most likely caused by the generator's need for producing multiple suboptimal models, which results in many calls being made to the tester for validating the minimality of each produced model. On the other hand, the performance of *lpods2asprin* and our implementations is not affected, probably due to *asprin*'s multi-shot answer set solving technique.

In several tests, the performance of our implementations and *lpod2asprin* is similar. However, *lpods2asprin* seems to perform better when there are ordered disjunctive rules with lots of options. In P11, for example, the maximum order disjunction involves most of the program's atoms. The extra rules that are required in our implemntations for computing the F^* values seem to be responsible for the performance impact in this case. On the other hand, our implementation performs better on programs that have many candidate and preferred answers sets, such as P12 and P14. We presume that *lpods2asprin*'s performance is affected by the more complex processing required to find out the degrees of satisfaction.

7. CONCLUSION AND FUTURE WORK

This thesis presents four translations of LPODs under the three-valued semantics introduced in [1]. The first two translations use assumption programs based on Lee and Yang's work in [8]. The third translation utilizes split programs, as Brewka et. al did in [6]. The final translation is based on Cabalar's [7] method for generating answer sets and our implementation is similar to the one in [9]. Several experiments were conducted in order to effectively compare our four translations as well as test their performance against existing LPOD systems. The results indicate that our final implementation has comparable performance to existing systems based on traditional semantics, while clearly outperforming them in computationally intensive programs, such as LPODs with numerous candidate and preferred answer sets. Therefore, the three-valued semantics can be implemented without any major additional computational burden.

Potential improvements for our implementations include optimizing the generation of rules that produce the F^* value in the translation process. Since the F^* value can be assigned to certain atoms –those in the heads of ordered disjunctive rules and those in the heads of regular rules whose bodies contain literals that can take the F^* value– the $isTFstar$ rules do not need to be included in the translations for all atoms of the program, but only for those that could take the F* value. Consequently, the $isTFstar(A)$ atoms that appear in rule bodies could be replaced by simply A for all atoms A that cannot take the F^* value.

To address the performance decrease in our implementations for programs with large ordered disjunctions, it would be interesting to explore ways of splitting large ordered disjunctions into smaller ones and examine whether doing so enhances the performance of the implementation.

Furthermore, a key area for future research is the definition and implementation of first-order LPODs. Although the semantics in [1] are for propositional LPODs, we could perhaps define the semantics of first-order LPODs using the ground instantiation of the program. It would also be interesting to see if the implementation could be expanded to handle first-order LPODs, i.e. implementing first-order LPODs using first-order ASP programs.

ABBREVIATIONS - ACRONYMS

LPOD	Logic Program(ming) with Ordered Disjunction
ASP	Answer Set Program(ming)

BIBLIOGRAPHY

[1] A. Charalambidis, P. Rondogiannis, and A. Troumpoukis. A logical characterization of the preferred models of logic programs with ordered disjunction. *Theory and Practice of Logic Programming*, 21(5):629–645, 2021.

[2] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.*, 9(4):365–385, 1991.

[3] G. Brewka. Logic programming with ordered disjunction. *Proceedings of the National Conference on Artificial Intelligence*, 05 2002.

[4] G. Brewka, I. Niemelä, and T. Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20(2):335–357, 2004.

[5] G. Brewka. Preferences in answer set programming. volume 4177, pages 1–10, 11 2005.

[6] G. Brewka, I. Niemelä, and T. Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. volume 2424, pages 444–455, 09 2002.

[7] P. Cabalar. A logical characterisation of ordered disjunction. *AI Commun.*, 24:165–175, 01 2011.

[8] J. Lee and Z. Yang. Translating LPOD and CR-Prolog2 into standard answer set programs. *Theory and Practice of Logic Programming*, 18(3–4):589–606, 2018.

[9] J. Lee and Z. Yang. Computing logic programs with ordered disjunction using asprin. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning*, Principles of Knowledge Representation and Reasoning: Proceedings of the 16th International Conference, KR 2018, pages 57–61. AAAI press, 2018.

[10] G. Brewka, J. Delgrande, and T. Schaub. asprin: Customizing answer set preferences without a headache. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29, 02 2015.