

On Dual Programs in Co-Logic Programming

Hirohisa Seki^(✉)

Department of Computer Science, Nagoya Institute of Technology,
Showa-ku, Nagoya 466-8555, Japan
`seki@nitech.ac.jp`

Abstract. Co-logic programming is an extension of the conventional logic programming language, by allowing each predicate to be annotated as either inductive or coinductive. To define its procedural semantics as well as an alternating fixpoint semantics, the *stratification restriction*, a condition on predicate dependency in programs, has been imposed on co-logic programs (co-LPs). In this paper, we first consider *dual* programs in co-logic programming: Given a program P , its dual program P^* is a program such that it defines the “complement” of P , i.e., for any ground atom $p(\bar{t})$, it computes its negation $\neg p(\bar{t})$. When we consider co-LPs with negation, we show that the stratification restriction becomes too restrictive in general, and that the Horn μ -calculus by Charatonik et al. can be used as an extension of co-logic programming for handling “non-stratified” co-LPs. We then consider some applications of non-stratified co-LPs to Answer Set Programming (ASP) and the well-founded semantics (WFS). In particular, we give new iterated fixpoint characterizations of answer sets as well as the WFS via dual programs. We also discuss some applications of non-stratified co-LPs to program transformation such as partial deduction, and a proof procedure for the WFS.

1 Introduction

Co-logic programming, proposed by Gupta et al. [11] and Simon et al. [26, 27], is an extension of logic programming, where each predicate in definite programs is annotated as either *inductive* or *coinductive*. To define its semantics, the *stratification restriction*, a condition on predicate dependency in co-logic programs (co-LPs), is assumed, and the declarative semantics by an alternating fixpoint semantics has been given: the least fixpoints for inductive predicates and the greatest fixpoints for coinductive predicates. A top-down procedural semantics, *co-SLD resolution*, has also been proposed, and recent SWI Prolog [28] has added support for coinduction.

As a result, co-logic programming provides a powerful computational framework, where many interesting applications such as modelling ω -automata [8], model checking [6], non-monotonic reasoning and SAT solvers can be easily

H. Seki — This work was partially supported by JSPS Grant-in-Aid for Scientific Research (C) 15K00305.

expressed and computed (see, e.g., [12]). Recently, there has been reported some work [16, 17] on applying co-LP techniques to Answer Set Programming using *dual programs*.

In this paper, we first consider *dual programs* in co-logic programming: Given a program P , its dual program P^* defines the “complement” of P , i.e., for any ground atom $p(\bar{t})$, $SEM(P) \models \neg p(\bar{t})$ iff $SEM(P^*) \models not_p(\bar{t})$, where not_p is a new predicate symbol and $SEM(P)$ is a semantics of P . The notion of dual programs has been studied in the literature; among others, Sato and Tamaki [22] has introduced a technique for program transformation called the *negation technique*. It has also been utilized in partial evaluation (or *partial deduction*), program transformation, implementation of proof procedures (see, e.g., [1, 4, 20, 21, 24]).

Considering dual programs of co-LPs with negation requires us to handle “non-stratified” co-LPs, and we show that the Horn μ -calculus by Charatonik et al. [5] can be used as an extension of co-logic programming. We then consider some applications of non-stratified co-LPs to Answer Set Programming (ASP) [10] and the well-founded semantics (WFS). In particular, we give new iterated fixpoint characterizations of answer sets as well as the WFS through dual programs. To the best of our knowledge, this is the first result of giving fixpoint characterizations of answer sets/WFS using dual programs. Some applications of non-stratified co-LPs to program transformation such as partial deduction, and a proof procedure for the WFS are also discussed.

The organization of this paper is as follows. In Sect. 2, we summarise some preliminary definitions on co-LPs and dual programs. In Sect. 3, we explain non-stratified co-logic programs and the Horn μ -calculus. In Sect. 4, we consider non-stratified co-LPs in the well-founded semantics. Finally, we discuss about the related work and give a summary of this work in Sect. 5.¹

Throughout this paper, we assume that the reader is familiar with the basic concepts of logic programming, which are found in [3, 15].

2 Preliminaries

In this section, we first recall some basic definitions and notations concerning co-logic programs (co-LPs). The details and more examples are found in [11, 26, 27]. Then, we also explain some preliminaries on deriving dual programs by negation elimination.

A *co-logic program* (co-LP) is a definite program, where predicate symbols are annotated as either inductive or coinductive. There is one restriction on co-LP, referred to as the *stratification restriction*: Inductive and coinductive predicates are not allowed to be mutually recursive. An example which violates the stratification restriction is $\{p \leftarrow q; q \leftarrow p\}$, where p is inductive, while q is coinductive.

When a co-LP P satisfies the stratification restriction, it is possible to decompose the set \mathcal{P} of all predicates in P into a collection (called a *stratification*) of

¹ Due to space constraints, we omit most proofs and some details, which will appear in the full paper.

mutually disjoint sets $\mathcal{P}_0, \dots, \mathcal{P}_r$ ($0 \leq r$), called *strata*, so that, for every clause $p(\tilde{x}_0) \leftarrow p_1(\tilde{x}_1), \dots, p_n(\tilde{x}_n)$ in P , we have that $\sigma(p) \geq \sigma(p_i)$ if p and p_i have the same inductive/coinductive annotations, and $\sigma(p) > \sigma(p_i)$ otherwise, where $\sigma(q) = i$, if the predicate symbol q belongs to \mathcal{P}_i . σ is called a *stratification function*.

The following is an example of co-LPs due to Simon et al. [26], which shows that co-logic programming can handle infinite terms such as infinite lists or trees like $f(f(\dots))$ as well as finite ones.

Example 1 [26]. Suppose that predicates *member* and *drop* are annotated as inductive, while predicate *comember* is annotated as coinductive.

$$\begin{aligned} \text{member}(H, [H|_]) &\leftarrow & \text{drop}(H, [H|T], T) &\leftarrow \\ \text{member}(H, [_|T]) &\leftarrow \text{member}(H, T) & \text{drop}(H, [_|T], T_1) &\leftarrow \text{drop}(H, T, T_1) \\ \text{comember}(X, L) &\leftarrow \text{drop}(X, L, L_1), \text{comember}(X, L_1) \end{aligned}$$

The definition of *member* is a conventional one; its meaning is defined in terms of the least fixpoint, since it is an inductive predicate. So, the prefix ending in the desired element H must be finite. The same applies to predicate *drop*.

On the other hand, predicate *comember* is coinductive, whose meaning is defined in terms of the greatest fixpoint. Therefore, it is true if and only if the desired element X occurs an infinite number of times in the list L . Hence it is false when the element does not occur in the list or when the element only occurs a finite number of times in the list.

For example, the query $X = 1, L = [0, 1|L]$, *comember*(X, L) is true, while the query $X = 1, L = [0, 1, 0, 1]$, *comember*(X, L) is false. Note that $L = [0, 1|L]$ represents an infinite list L consisting of 0 s and 1 s. \square

A meta-interpreter for co-logic programming has been developed and available [14], and recent SWI-Prolog (version 6.5.1) has also offered a module for supporting coinduction.²

The declarative semantics of a co-logic program is a stratified interleaving of the least fixpoint semantics and the greatest fixpoint semantics. To handle infinite terms, we consider the *complete* (or *infinitary*) Herbrand base [13, 15], denoted by HB_P^* , where P is a program.³

Let P be a co-logic program with a stratification $\mathcal{P}_0, \dots, \mathcal{P}_r$ ($0 \leq r$). Let Π_i ($0 \leq i \leq r$) be the set of clauses whose head predicates are in \mathcal{P}_i . Then, $P = \Pi_0 \cup \dots \cup \Pi_r$. Let Π (resp., S) be a set of clauses (resp., ground atoms). Similarly to the “immediate consequence operator” T_P in the literature, our operator $T_{\Pi, S}$ assigns to every set I of ground atoms a new set $T_{\Pi, S}(I)$ of ground atoms as

² <http://www.swi-prolog.org/pldoc/doc/swi/library/coinduction.pl>.

³ In the following sections, we will restrict ourselves to propositional programs for the ease of exposition, thus the “standard” Herbrand base HB_P will suffice. In this section, however, we explain some of the general basics of co-LPs for readers unfamiliar with them.

$$T_{\Pi,S}(I) = \{A \in HB_{\Pi}^* \mid \text{there is a ground instance of some clause in } \Pi \\ A \leftarrow B_1, \dots, B_n, n \geq 0, \text{ such that, for every } 1 \leq i \leq n, \\ \text{either } B_i \in I \text{ or } B_i \in S\}.$$

In the above, the atoms in S are treated as facts. S is intended to be a set of atoms whose predicate symbols are in lower strata than those in the current stratum Π . We consider $T_{\Pi,S}$ to be the operator defined on the set of all subsets of HB_{Π}^* , ordered by standard inclusion. Then, $T_{\Pi,S}$ admits a least and a greatest fixpoint denoted by $lfp(T_{\Pi,S})$ and $gfp(T_{\Pi,S})$, respectively.

Finally, the model $M(P)$ of a co-logic program $P = \Pi_0 \cup \dots \cup \Pi_r$ is defined inductively as follows: Let $M(\Pi_{-1}) = \emptyset$. For $k \geq 0$, $M(\Pi_k) = lfp(T_{\Pi_k, M_{k-1}})$ if \mathcal{P}_i is inductive; $gfp(T_{\Pi_k, M_{k-1}})$ if \mathcal{P}_i is coinductive, where M_{k-1} is the model of lower strata than Π_k , i.e., $M_{k-1} = \bigcup_{i=-1}^{k-1} M(\Pi_i)$.

Then, the *model* of P is $M(P) = \bigcup_{i=0}^r M(\Pi_i)$, the union of all models $M(\Pi_i)$.

Example 2. Let $P_0 = \{p \leftarrow q; q \leftarrow q; r \leftarrow r\}$ be a set of clauses. In the traditional logic programming, the meaning of P_0 is given in terms of the least fixpoint semantics, $lfp(T_{P_0}) = \emptyset$.

In co-logic programming, on the other hand, assume that p and r are inductive predicates, while q is a coinductive predicate. Then, since P_0 satisfies the stratification restriction, its meaning is defined in co-logic programming, i.e., $M(P_0) = \{p, q\}$. \square

Dual Programs in Co-Logic Programming. Our approach to handling negation in (co-)LPs is based on *negation elimination* (NE for short), a familiar program transformation technique [22], tailored to co-logic programs [24]. Given a (co-)LP P , the NE transformation derives from P a set P^* of definite clauses, called the *dual program* of P , by replacing negative literals $\neg p(\bar{t})$ by $not_p(\bar{t})$, where not_p is a newly introduced predicate symbol.

In the following, we explain NE for a program P such that a clause in P might contain negative literals in its body for later use, and we assume that P is a propositional program for the ease of exposition, although it is applicable to programs without existential variables.⁴ NE consists of the following two steps:

- (step 1) for each clause in P , we replace each occurrence $\neg p$ of negative literals (if any) by not_p , where not_p is a new predicate not appearing elsewhere;
- (step 2) for each predicate p , let $comp(p)$ be its completed definition in P . We then derive the definition of not_p from $comp(p)$ as follows:

- (i) [Definition Derivation] Suppose that $comp(p)$ is of the form $p \leftrightarrow B_1 \vee \dots \vee B_n$. Then, negating both sides of $comp(p)$, and replacing every negative occurrence $\neg p$ by not_p , we obtain $not_p \leftrightarrow \neg(B_1 \vee \dots \vee B_n)$.

Next, transforming the right-hand side in the above to a disjunctive form,

⁴ A variable in a clause is *existential* if it appears in the body of the clause, but not in the head.

using De Morgan’s laws, replacing each occurrence of $\neg\neg q$ by q , and each occurrence of $\neg q$ by not_q , we obtain the completed definition of not_p , i.e., $not_p \leftrightarrow NB_1 \vee \dots \vee NB_{n'}$, where each NB_i is a conjunction of positive literals. Finally, we transform $comp(not_p)$ to a set of clauses: $\{not_p \leftarrow NB_1; \dots; not_p \leftarrow NB_{n'}\}$.

- (ii) [Annotation Inversion] Annotate the derived predicate not_p as “coinductive” (resp. “inductive”) if the annotation of the original predicate p is inductive (resp. coinductive).

Let P be a (definite) co-LP with the stratification restriction σ , and P^* be the set of all clauses obtained by applying the above NE transformation. We define the stratification function σ^* for P^* as follows: $\sigma^*(p) = \sigma(p)$ for all predicates defined in P , and $\sigma^*(not_p) = \sigma(p) + 1$ for all predicates not_p newly introduced in NE. Then, we can show that P^* satisfies the stratification restriction w.r.t. σ^* .

Proposition 1. Correctness of Negation Elimination [24]

Let P be a *definite* co-logic program. If every clause in P has no existential variable, then the procedure of negation elimination gives a complementary co-logic program P^* , i.e., for any ground term \tilde{t} ,

$$M(P) \models \neg p(\tilde{t}) \text{ iff } M(P^*) \models not_p(\tilde{t}).$$

Example 3 (Continued from Example 2). Consider again $P_0 = \{p \leftarrow q; q \leftarrow q; r \leftarrow r\}$ in Example 2. Then, its dual program $P^* = P_0 \cup \{not_p \leftarrow not_q; not_q \leftarrow not_q; not_r \leftarrow not_r\}$.

Recall that p and r are inductive predicates, while q is a coinductive predicate. Therefore, not_p and not_r are coinductive, while not_q is inductive. Thus, $M(P^*) = \{p, q, not_r\}$. We note that $M(P) \models \neg r$ and $M(P^*) \models not_r$. \square

In the following, we restrict ourselves to propositional programs, where the condition of NE (Proposition 1) is always satisfied. When a program has existential variables, NE will be still applicable if a certain condition [22] is satisfied. Another way is to use grounding by `lparse` in `smodels` [19], which allows us to deal with more general classes of programs such as range-restricted programs. Moreover, our applications of dual programs here include AS and abduction, where it is often the case that datalog programs are considered.

3 Non-stratified Co-Logic Programs and Horn μ -calculus

Proposition 1 deals with *definite* co-logic programs, thus satisfying the stratification restriction. However, the stratification restriction becomes too restrictive, when we consider dual programs of (co-)LPs with negation. In this section, we show that the Horn μ -calculus by Charatonik et al. [5] can be used as a framework for handling “non-stratified” co-LPs.

3.1 Dual Programs of Non-Stratified Programs

The notion of dual programs has been studied in logic programming; we have already explained the program transformation technique, called the *negation technique*, by Sato and Tamaki [22] in Sect. 2. Aravindan and Dung [4] have then proposed partial evaluation (or *partial deduction*) of logic programs in the well-founded semantics (WFS) using dual programs. Dual programs have also been used for implementing proof procedures; ABDUAL by Alferes et al. [1] and its successors such as TABDUAL [20, 21] are abductive reasoning systems for the WFS. Marple and Gupta [16] have proposed a proof procedure for answer set programs using their dual programs.

In particular, Aravindan and Dung [4] have shown the following result on partial deduction in the WFS, which is given in the case of propositional programs for the sake of simplicity.

Proposition 2 (Aravindan and Dung). Partial Deduction in the WFS [4]
Let P be a program whose well-founded model is complete⁵ and P^{*-ai} a negative partial deduction of P obtained by replacing selected negative literals $\neg p$ by not_p and adding new definitions for not_p . Then, for every goal $G: \leftarrow A$ which contains no not_p predicate, $WFS(P) \models A$ iff $WFS(P^{*-ai}) \models A$. \square

In the above, when applied to propositional programs, a *negative partial deduction* of P , denoted by P^{*-ai} , is the same as the dual program P^* explained in Sect. 2, except that annotation inversion is not employed. However, the above proposition is not always correct as the following example shows. Given a program P , we denote by $WFS(P) = \langle T; F \rangle$, where T (resp., F) is the set of atoms true (resp., false) in the WFS of P . The truth value of the remaining atoms in $U = H_P \setminus (T \cup F)$ is undefined, where H_P is the Herbrand base of P .

Example 4. Consider the following program $P = \{p \leftarrow \neg q; q \leftarrow q\}$. Since P is a stratified program, P has a unique answer set, i.e., the perfect model, $PERF(P) = \{p\}$, which coincides with $WFS(P) = \langle \{p\}; \{q\} \rangle$ and thus it is complete.

Consider the negative partial deduction $P^{*-ai} = \{p \leftarrow not_q; not_p \leftarrow q; not_q \leftarrow not_q; q \leftarrow q\}$, where all the atoms are annotated as inductive.

Then, $WFS(P^{*-ai}) = \langle \emptyset; \{p, q, not_p, not_q\} \rangle$. In particular, p is thus false.

On the other hand, the dual program P^* is the same set of clauses as P^{*-ai} with different predicate annotations: not_p and not_q are coinductive, while p and q are inductive. P^* satisfies the stratification restriction. Then, $M(P^*) = \{p, not_q\}$; this means that P^* with its co-LP semantics $M(P^*)$, exactly captures the semantics of P in the WFS. \square

When we consider conventional general (i.e., non-stratified) programs, all predicates are supposed to be annotated as *inductive*. Then, the resulting dual programs do not satisfy the stratification restriction in general. For example,

⁵ Well-founded model of a program P is complete when it classifies all the elements of the Herbrand base as ‘true’ or ‘false’.

let $P = \{p \leftarrow \neg q; q \leftarrow \neg p\}$. Then, its dual program $P^* = \{p \leftarrow \text{not_}q; q \leftarrow \text{not_}p; \text{not_}p \leftarrow q; \text{not_}q \leftarrow p\}$, which does not satisfy the stratification restriction.

3.2 Horn μ -Calculus and Its Fixpoint Semantics

Charatonik et al. [5] have proposed the *Horn μ -calculus*; it is an extension of logic programs by allowing nesting of least and greatest fixpoints, in terms of a priority of each predicate for specifying whether its semantics has to be computed as a least or a greatest fixpoint. They have given to the Horn μ -programs the semantics based on ground proof trees as well as the nested fixpoints semantics.

A *Horn μ -program* (P, Ω) is a set of definite clauses in which every predicate symbol p in P is associated with a non-negative number $\Omega(p)$, called the *priority* of p .

Charatonik et al. [5] give an iterated fixpoint characterization of the semantics $\llbracket(P, \Omega)\rrbracket$ of a program P , which we will use in the following.

First we recall the familiar T_P operator of logic programming (see [15]); for any set A of ground atoms, we define $T_P(A)$ in the standard manner. Next, for all sets A and B of ground atoms, and non-negative priority k , we define $A[k := B]$ to be the set of ground atoms such that $p \in A[k := B]$ if either the priority of p is different from k and $p \in A$ or the priority of p equals k and $p \in B$. We now define the operator T_P^k such that $T_P^k(A) = A[k := T_P(A)]$; The operator T_P^k is thus analogous to T_P except that T_P^k only updates predicates of priority k . Then, for each value of k , we take fixpoints of the operators T_P^k . To do that, for any integer k we define $F_P^k(A)$ as follows. First, for negative integer k , we define $F_P^k(A) = A$.

$$F_P^k(A) = \begin{cases} \nu B. T_P^k(F_P^{k-1}(A[k := B])) & \text{if } k \text{ is even,} \\ \mu B. T_P^k(F_P^{k-1}(A[k := B])) & \text{if } k \text{ is odd,} \end{cases}$$

Then, $\llbracket(P, \Omega)\rrbracket = F_P^n(\emptyset)$, where $n = \max(\Omega(P))$, i.e., n is the maximal priority of any predicate in P .

It is easy to show that the Horn μ -calculus is an extension of co-logic programming. In fact, let P be a co-LP with a stratification σ . Then, we call a priority function Ω *consistent* with σ , if it satisfies the following: (i) $\sigma(p) \leq \sigma(q)$ iff $\Omega(p) \geq \Omega(q)$ for any predicates p and q , i.e., a predicate in lower stratum has a higher priority, and (ii) $\Omega(p)$ is even (odd) if p is a coinductive (inductive) predicate, respectively. Then, we have the following:

Proposition 3. Let P be a co-logic program with a stratification function σ . Then, $M(P) = F_P^n(\emptyset)$ with priority Ω , where Ω is consistent with σ , and $n = \max \Omega(p)$ for any predicate p in P . \square

Example 5. Consider again $P_0 = \{p \leftarrow q; q \leftarrow q; r \leftarrow r\}$ in Example 2, where p and r are inductive predicates, while q is a coinductive predicate. Then, its stratification function σ is defined as: $\sigma(q) = 0, \sigma(p) = \sigma(r) = 1$.

Consider a priority Ω consistent with σ such that $\Omega(p) = \Omega(r) = 1, \Omega(q) = 2$. Then, we have that $\llbracket (P_0, \Omega) \rrbracket = F_{P_0}^2(\emptyset) = \{p, q\} = M(P_0)$. \square

On the other hand, the Horn μ -calculus is more general than stratified co-LP.

Example 6 (Adapted and Simplified from [5]). Let $P_0 = \{p \leftarrow p; p \leftarrow q; q \leftarrow p\}$ be a set of clauses, where p is an inductive predicate, while q is a coinductive predicate. Since P_0 does not satisfy the stratification restriction, its meaning is not given in co-logic programming.

In the Horn μ -calculus, however, the semantics of P_0 can be determined in terms of priorities assigned to the predicates. Suppose, for example, that the coinductive predicate q has a higher priority than the inductive predicate p . We thus define: $\Omega(p) = 1$ and $\Omega(q) = 2$. Then, $\llbracket (P_0, \Omega) \rrbracket = \{p, q\}$. \square

The framework for unfold/fold transformation of co-logic programs is proposed in [23], where a program is assumed to satisfy the stratification restriction. We note that unfolding does not preserve the meaning of a Horn μ -program in general, as the following example shows.

Example 7. Let $P_0 = \{p \leftarrow q; q \leftarrow p\}$ be a set of clauses, where $\Omega(p) = 1$ (i.e., p is an inductive predicate), while $\Omega(q) = 2$ (i.e., q is a coinductive predicate). Then, $\llbracket (P_0, \Omega) \rrbracket = \{p, q\}$. Note that P_0 does not satisfy the stratification restriction.

$$\begin{array}{ccc} P_0 : p \leftarrow \mathbf{q} & \xrightarrow{\text{unfolding}} & P_1 : p \leftarrow p \\ q \leftarrow p & & q \leftarrow p \\ \\ P_0 : p \leftarrow q & \xrightarrow{\text{unfolding}} & P_2 : p \leftarrow q \\ q \leftarrow \mathbf{p} & & q \leftarrow q \end{array}$$

In the above, the atoms in bold letters are the ones on which unfolding is applied. Then, $M(P_1) = \emptyset$, while $M(P_2) = \{p, q\}$. Therefore, when P_0 does not satisfy the stratification restriction, a simple application of unfolding will yield programs with different meanings. \square

We will consider unfolding in the Horn μ -calculus in the following section.

Answer Set Programs and Horn μ -programs. We are now in a position to give the relationship between answer sets and Horn μ -programs. Let M be a set of atoms and M^* a set of atoms including ones of the form $\text{not_}p$ for some atom p . In the following, we denote by $M \equiv M^*$ if, for any ground atom p , $p \in M$ iff $p \in M^*$, and $p \notin M$ iff $\text{not_}p \in M^*$.

Example 8. Consider again the following program $P = \{p \leftarrow \neg q; q \leftarrow \neg p\}$ and its dual program $P^* = \{p \leftarrow \text{not_}q; q \leftarrow \text{not_}p; \text{not_}p \leftarrow q; \text{not_}q \leftarrow p\}$. P is a non-stratified program, and it has two answer sets $M_1 = \{p\}$ and $M_2 = \{q\}$.

On the other hand, P^* does not satisfy the stratification restriction. First, we consider a Horn μ -program (P^*, Ω_1) , where we define the priority Ω_1 as

$\Omega_1(not_q) = 2$ and $\Omega_1(not_p) = 0$, while $\Omega_1(p) = \Omega_1(q) = 1$. Then, $\llbracket(P, \Omega_1)\rrbracket = \{p, not_q\}$, thus $M_1 \equiv \llbracket(P, \Omega_1)\rrbracket$.

On the other hand, we consider a Horn μ -program (P^*, Ω_2) in a symmetric fashion; we define the priority Ω_2 as $\Omega_2(not_p) = 2$ and $\Omega_2(not_q) = 0$, while $\Omega_2(p) = \Omega_2(q) = 1$ as before. Then, $\llbracket(P, \Omega_2)\rrbracket = \{q, not_p\}$, thus $M_2 \equiv \llbracket(P, \Omega_2)\rrbracket$. \square

In general, we can show the following relationship between the answer sets of P and the fixpoints of $F_{P^*}^2(\emptyset)$.

Proposition 4. AS is a fixpoint of a dual program

Let P be a logic program and $AS(P)$ the set of its answer sets. If $M \in AS(P)$, then there exists a priority Ω such that $M \equiv F_{P^*}^2(\emptyset)$.

In particular, when P is a stratified program, $PERF(P) \equiv F_{P^*}^2(\emptyset) = M(P^*)$, where priority Ω is defined to be consistent with the stratification function of P . \square

Example 4 is a special case of the above proposition.

Unfolding Dual Programs. Next, we consider unfolding of dual programs.

Example 9. Consider again the dual program P^* in Example 8 and recall that the priority Ω_1 is defined as $\Omega_1(not_q) = 2$ and $\Omega_1(not_p) = 0$, while $\Omega_1(p) = \Omega_1(q) = 1$. Then, $\llbracket(P, \Omega_1)\rrbracket = \{p, not_q\}$. We consider the following two cases of applying unfolding to P^* , where the atoms in bold letters are the ones on which unfolding are applied.

(i)

$$\begin{array}{ccc} P^* : p \leftarrow not_q & \xrightarrow{\text{unfolding}} & P_1^* : p \leftarrow not_q \\ q \leftarrow \mathbf{not_p} & & q \leftarrow q \\ not_p \leftarrow q & & not_p \leftarrow q \\ not_q \leftarrow p & & not_q \leftarrow p \end{array}$$

Then, $\llbracket(P_1^*, \Omega_1)\rrbracket = \{p, not_q\}$, thus unfolding preserves the semantics. We note that, in the unfolded clause, $\Omega_1(q) \geq \Omega_1(not_p)$.

(ii)

$$\begin{array}{ccc} P^* : p \leftarrow \mathbf{not_q} & \xrightarrow{\text{unfolding}} & P_2^* : p \leftarrow p \\ q \leftarrow not_p & & q \leftarrow not_p \\ not_p \leftarrow q & & not_p \leftarrow q \\ not_q \leftarrow p & & not_q \leftarrow p \end{array}$$

Then, $\llbracket(P_2^*, \Omega_1)\rrbracket = \emptyset$, thus unfolding does not preserve the semantics. We note that, in the unfolded clause, $\Omega_1(p) < \Omega_1(not_q)$. \square

The following proposition explains the above applications of unfolding.

Proposition 5. Unfolding of Horn μ -programs

Let (P, Ω) be a Horn μ -program and P' a program derived by applying unfolding to P . Let $p \leftarrow q_1, \dots, q_n$ ($n > 1$) be the unfolded clause in P and q_i ($1 \leq i \leq n$) the atom upon which unfolding is applied. If $\Omega(p) \geq \Omega(q_i)$, then the semantics of (P, Ω) is preserved, i.e., $\llbracket (P, \Omega) \rrbracket = \llbracket (P', \Omega) \rrbracket$. \square

Aravindan and Dung [4] also studied unfolding of dual programs in the well-founded semantics, where unfolding is defined as usual, i.e., no condition is imposed on applying unfolding. In contrast, unfolding in our framework requires the above-mentioned condition for its application, since each atom in a Horn μ -program is assigned its priority and the semantics (P, Ω) is defined based on the priorities of atoms.

4 Dual Programs in the Well-Founded Semantics

Finally, we consider a fixpoint characterization of the well-founded semantics (WFS) via dual programs, which is based on the Horn μ -calculus.

Proposition 6. Fixpoint Characterization of the WFS

Let P be a logic program and v the truth valuation in the well-founded semantics $WFS(P)$. Let P^* be its dual program and Ω be a priority defined as $\Omega(p) = 1$ (resp., $\Omega(not_p) = 2$) for any predicate p in P . Then, we have

- $v(p) = \mathbf{t}$ iff $p \in F_{P^*}^2(\emptyset)$ and $not_p \notin F_{P^*}^2(\emptyset)$,
- $v(p) = \mathbf{f}$ iff $not_p \in F_{P^*}^2(\emptyset)$ and $p \notin F_{P^*}^2(\emptyset)$, and
- $v(p) = \mathbf{u}$ iff $p \in F_{P^*}^2(\emptyset)$ and $not_p \in F_{P^*}^2(\emptyset)$. \square

Example 10. Consider the following non-stratified program $P = \{p \leftarrow q; q \leftarrow \neg p\}$, and its dual program $P^* = \{p \leftarrow q; q \leftarrow not_p; not_p \leftarrow not_q; not_q \leftarrow p\}$. Then, $WFS(P) = \{\emptyset; \emptyset\}$, i.e., the truth value of every atom in P is undefined: $v(a) = \mathbf{u}$ for $a \in \{p, q\}$.

On the other hand, we consider a Horn μ -program (P^*, Ω) , where we define the priority Ω as $\Omega(not_a) = 2$ and $\Omega(a) = 1$ for $a \in \{p, q\}$. Then, $\llbracket (P, \Omega) \rrbracket = \{p, q, not_p, not_q\}$. \square

A Proof Procedure of Dual Programs in the WFS. From Proposition 6, we will propose a simple proof procedure, which is based on the notion of P -derivation of the Horn μ -calculus [5].

The semantics of a Horn μ -program is given in terms of ground derivations [5]. Given a logic program P , r is called a P -derivation if for each node n in r , labeled by some ground atom h , there exists a ground instance $h \leftarrow b_1, \dots, b_m$ ($m \geq 0$) of a clause in P , and n has m children nodes, each of which is labeled by b_i ($0 \leq i \leq m$). When $m = 0$, the node n has no children nodes and is a *success node*. If there are no such clauses in P , then n has no children nodes and is a *failure node*. When the root node of r is labeled by p , r is a P -derivation of p (or p has a P -derivation r).

Given a P -derivation r , let w be an infinite sequence $w_0w_1w_2\dots$ of nodes in r such that w_{i+1} is a child of w_i . Such a sequence w is called an *infinite path*. For an infinite path π in a P -derivation, we denote by $\text{Inf}(\pi)$ the set of all priorities occurring infinitely often on the path π . We say that a path w in r is *accepting* if the largest element of $\text{Inf}(\pi)$ is even. A P -derivation r is *accepting* if every infinite path in r is accepting and every finite path ends with a success node. A P -derivation r is *not accepting* if there exists either an infinite path in r which is not accepting or a finite path ending with a failure node.

From the equivalence of the procedural semantics and the iterated fixpoint semantics of the Horn μ -calculus [5], we have the following characterization of procedural semantics of a dual program P^* in the WFS.

- $v(p) = \mathbf{t}$ iff p (resp. $\text{not_}p$) has an (resp. no) accepting P^* -derivation,
- $v(p) = \mathbf{f}$ iff $\text{not_}p$ (resp. p) has an (resp. no) accepting P^* -derivation, and
- $v(p) = \mathbf{u}$ iff both p and $\text{not_}p$ have accepting P^* -derivations.

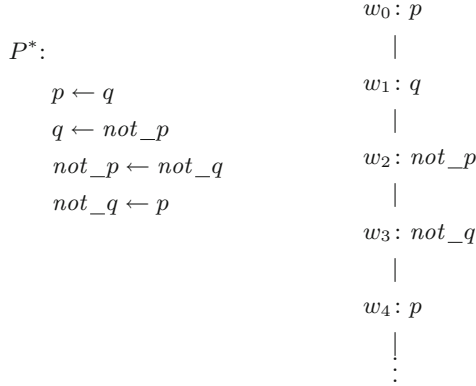


Fig. 1. The P^* -Derivation of p (Example 10)

Since we consider here propositional dual programs in the WFS, we can have a simple procedure for detecting whether $v(p) = \mathbf{u}$ for a ground atom p . To do that, we will make the notion of *(non)-acceptance* more detailed.

Given a P^* -derivation r of an atom p_0 , suppose that there is an infinite path $\pi = w_0w_1w_2\dots$ in r . We denote the label of node w_i ($i \geq 0$) by $w_i.l$. Since dual programs are propositional, there exist some node w_n ($n \geq 0$) and $k \geq 0$ such that w_n and w_{n+k+1} have the same label, i.e., $w_n.l = w_{n+k+1}.l$. Then, π is of the form: $w_0w_1\dots(w_n\dots w_{n+k})^\omega$.

Definition 1. Let r be a P^* -derivation. Then,

- r is called *successful* if every finite path in r ends with a success node and for every infinite path π in r , $\Omega(w_i.l) = 2$ for all i ($n \leq i \leq n+k$).

- r is called *failed* if there exists either a finite path in r ending with a failure node or an infinite path π in r such that $\Omega(w_i.l) = 1$ for all i ($n \leq i \leq n+k$).
- r is called *undefined* if it is neither successful nor failed. In this case, r has an infinite path π such that $\Omega(w_n.l) \neq \Omega(w_m.l)$ for some m ($n \leq m \leq n+k$). \square

Example 11. Figure 1 shows a P^* -derivation of p in Example 10. Recall that $v(p) = \mathbf{u}$. p has a single P^* -derivation r ; it consists of a single infinite path $\pi = (w_0 \dots w_3)^\omega$, where $\Omega(w_0.l) = \Omega(p) = 1$ and $\Omega(w_2.l) = \Omega(\text{not_}p) = 2$. The priorities of labels in the infinite loop are thus alternating. \square

Proposition 7. Procedural Semantics of Dual Programs in the WFS

Let P be a logic program and v the truth valuation in the well-founded semantics $WFS(P)$. Let P^* be its dual program and Ω be a priority defined as $\Omega(p) = 1$ (resp., $\Omega(\text{not_}p) = 2$) for any predicate p in P . Then, we have

- $v(p) = \mathbf{t}$ iff p has a successful P^* -derivation,
- $v(p) = \mathbf{f}$ iff all of P^* -derivations of p are failed, and
- $v(p) = \mathbf{u}$ iff all of P^* -derivations of p are neither successful nor failed. \square

We note some implementation issues on the above procedural semantics. ABDUAL [1] has some problems in handling loops involving negative atoms, referred to as “negative loops over negation” (*NLoN*) [20]. In Example 10, for example, goals $\leftarrow p$ and $\leftarrow \text{not_}p$ succeed unexpectedly in ABDUAL. In TABDUAL [20], such problems have been remedied by introducing some mechanisms for dealing with NLoN. However, its implementation for handling NLoN is dependent on the XSB built-in predicates such as `tnot/1` and `call.tv/2` (see Fig. 2 (below)) together with an auxiliary predicate `over/1`, which would make it difficult to perform program analysis and its possible optimization.

On the other hand, Proposition 7 will give a simple proof procedure for the WFS. For example, predicate `solve/1` in DRA interpreter [14] (see Fig. 2 (above)) has an argument which maintains information about the current path of ancestors (stack). Using this mechanism, it would be simple to check whether a path currently stored on the stack is successful, failed or undefined, since it is enough to examine the labels of atoms in the stack.

5 Related Work and Concluding Remarks

The notion of dual programs has been proposed and utilized in various fields in logic programming. Techniques using dual programs have been proposed for performing partial evaluation (or partial deduction), program transformation and implementation of reasoning systems and proof procedures (see, for example, [1, 4, 16–18, 20–22, 24]). In this paper we have extended to non-stratified (co-)LPs negation elimination with the operation of annotation inversion, which was proposed for definite co-LPs [24].

The main contributions of this paper are the following. (i) We have shown that the Horn μ -calculus can be utilized as a framework for handling non-stratified co-LPs. In fact, the Horn μ -calculus is an extension of co-logic programming (Proposition 3). Gupta et al. [12] have also proposed an extension

```

% solve/1 in DRA interpreter
solve( + sequence of goals, + stack, + coinductive hypotheses, + level):
%% Solve the sequence of goals, maintaining information about the current chain
%% of tabled ancestors (stack) and the chain of coinductive ancestors
%% (coinductive hypotheses). The level is the level of recursion, and is used
%% only for tracing.

% Tbdual Implementation of Ex. 10
% the predicate over(G)/1 is defined as over(G) :- tnot(G).
1. :- table q_ab/1, over/1, not_p/1, p_st/3.
2. q_ab(E) :- tnot p_ab([]), not_p_ab([],E).
3. not_p_ab(I,0) :- call_tv(tnot over(not_p(I)), V),
    (V=undefined, 0=I, undefined;
    inspect(p_st(I,0,[]))).
4. not_p(I) :- p_st(I,0,[]).
5. ... (omitted)

```

Fig. 2. Predicate `solve/1` in DRA interpreter [14] (above) and implementation in TABDUAL (below)

of co-LPs to handle non-stratified co-LPs. To do that, they have introduced *strong/weak inductive* annotations, which play a similar role of priorities in the Horn μ -calculus. However, they have not discussed the relationship of their extension with the Horn μ -calculus, and its declarative semantics is not known either. (ii) We have given new iterated fixpoint characterizations of answer sets as well as the WFS for propositional programs via dual programs. A lot of work has been done on the fixpoint semantics for logic programming (see, e.g., an excellent survey in [9]). Denecker et al. [7], for example, have proposed a fixpoint theory as a uniform framework of major semantics of general logic programs. In contrast, our approach has focused on the use of dual programs, and to the best of our knowledge, this is the first result of giving fixpoint characterizations of answer sets/WFS through dual programs. In [25], the relationship between co-LP and the Horn μ -calculus has been studied from the procedural point of view. (iii) Finally, we have proposed an unfolding rule for Horn μ -programs (Proposition 5) and a procedural semantics of dual programs in the WFS (Proposition 7). The unfolding rule in this paper is more general than that for co-LPs [23] in that the latter is applicable only when the stratification restriction of a given program is satisfied. Furthermore, the proof procedure for the WFS in this paper is much simpler than that in [25] in that the latter requires checking a well-founded ordering among ground atoms, while such checking is replaced here by simply examining whether priorities of labels are alternating or not.

In this paper, we have restricted ourselves to propositional programs. This would be reasonable, since co-logic programming has some computational difficulty [2]. This restriction is due to the condition of NE (Proposition 1), i.e., no existential variables in every clause in a given co-LP. One direction for future work is thus to extend the current framework to handle a more general class of co-LPs. Some approaches have already been mentioned in the end of

Sect. 2. Another approach will be to allow arbitrary first order logic formulas in the body of a clause, as in the work by Denecker et al. [7].

We have proposed the proof procedure for the WFS, and compared it with ABDUAL and TABDUAL (Fig. 2). It will be interesting to extend proof procedure to allow *abducibles* for performing abduction. We also have a plan to implement our proof procedure for the WFS in Proposition 7.

Acknowledgement. The author would like to thank anonymous reviewers for their constructive and useful comments on the previous version of the paper. The idea of using co-LP techniques for a proof procedure for the WFS in Sect. 4 came from the discussions with Gopal Gupta at LOPSTR'13 in Madrid.

References

1. Alferes, J.J., Pereira, L.M., Swift, T.: Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Theor. Pract. Log. Program.* **4**, 383–428 (2004)
2. Ancona, D., Dovier, A.: co-LP: Back to the Roots. *Theory and Practice of Logic Programming* 13(4–5) (2013)
3. Apt, K.R.: Introduction to logic programming. In: *Handbook of Theoretical Computer Science*, pp. 493–576. Elsevier (1990)
4. Aravindan, C., Dung, P.M.: Partial deduction of logic programs wrt well-founded semantics. *New Gener. Comput.* **13**(1), 45–74 (1994)
5. Charatonik, W., McAllester, D., Niwinski, D., Podelski, A., Walukiewicz, I.: The Horn Mu-calculus. In: *LICS 1998*, pp. 58–69. IEEE Computer Society (1998)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: *Model Checking*. MIT Press, Cambridge (1999)
7. Denecker, M., Bruynooghe, M., Vennekens, J.: Approximation fixpoint theory and the semantics of logic and answers set programs. In: Erdem, E., Lee, J., Lierler, Y., Pearce, D. (eds.) *Correct Reasoning*. LNCS, vol. 7265, pp. 178–194. Springer, Heidelberg (2012)
8. Farwer, B.: ω -Automata. In: Grädel, E., Thomas, W., Wilke, T. (eds.) *Automata, Logics, and Infinite Games*. LNCS, vol. 2500, pp. 3–21. Springer, Heidelberg (2002)
9. Fitting, M.: Fixpoint semantics for logic programming a survey. *Theoret. Comput. Sci.* **278**(1–2), 25–51 (2002)
10. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pp. 1070–1080. MIT Press (1988)
11. Gupta, G., Bansal, A., Min, R., Simon, L., Mallya, A.: Coinductive logic programming and its applications. In: Dahl, V., Niemelä, I. (eds.) *ICLP 2007*. LNCS, vol. 4670, pp. 27–44. Springer, Heidelberg (2007)
12. Gupta, G., Saeedloei, N., DeVries, B., Min, R., Marple, K., Kluźniak, F.: Infinite computation, co-induction and computational logic. In: Corradini, A., Klin, B., Cirstea, C. (eds.) *CALCO 2011*. LNCS, vol. 6859, pp. 40–54. Springer, Heidelberg (2011)
13. Jaffar, J., Stuckey, P.: Semantics of infinite tree logic programming. *Theoret. Comput. Sci.* **46**, 141–158 (1986)

14. Kluźniak, F.: Meta-interpreter supporting tabling and coinduction (2009). <http://www.utdallas.edu/gupta/meta.html>
15. Lloyd, J.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987). Extended edition
16. Marple, K., Bansal, A., Min, R., Gupta, G.: Goal-directed execution of answer set programs. In: PPDP 2012. pp. 35–44 (2012)
17. Marple, K., Bansal, A., Min, R., Gupta, G.: Dynamic consistency checking in goal-directed answer set programming. *Theory Pract. Log. Program.* **14**(4–5), 415–427 (2014)
18. Min, R.: Predicate Answer Set Programming with Coinduction. Ph.D. thesis, University of Texas at Dallas (2010)
19. Niemelä, I., Simons, A.: Smodels — an implementation of the stable model and well-founded semantics for normal logic programs. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 420–429. Springer, Heidelberg (1997)
20. Pereira, L.M., Saptawijaya, A.: Abductive logic programming with tabled abduction. In: Proceedings of the Seventh International Conference on Software Engineering Advances ICSEA 2012, pp. 548–556 (2012)
21. Saptawijaya, A., Pereira, L.M.: Tabled abduction in logic programs. *Theory Pract. Log. Program.* **13**, 4–5 (2013)
22. Sato, T., Tamaki, H.: Transformational logic program synthesis. In: FGCS 1984 Tokyo, pp. 195–201 (1984)
23. Seki, H.: Proving properties of co-logic programs by unfold/fold transformations. In: Vidal, G. (ed.) LOPSTR 2011. LNCS, vol. 7225, pp. 205–220. Springer, Heidelberg (2012)
24. Seki, H.: Proving properties of co-logic programs with negation by program transformations. In: Albert, E. (ed.) LOPSTR 2012. LNCS, vol. 7844, pp. 213–227. Springer, Heidelberg (2013)
25. Seki, H.: Extending co-logic programs for branching-time model checking. In: Gupta, G., Peña, R. (eds.) LOPSTR 2013, LNCS 8901. LNCS, vol. 8901, pp. 127–144. Springer, Heidelberg (2014)
26. Simon, L., Mallya, A., Bansal, A., Gupta, G.: Coinductive logic programming. In: Etalle, S., Truszczyński, M. (eds.) ICLP 2006. LNCS, vol. 4079, pp. 330–345. Springer, Heidelberg (2006)
27. Simon, L.E.: Extending Logic Programming with Coinduction. Ph.D. thesis. University of Texas at Dallas (2006)
28. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. *Theory Pract. Log. Program.* **12**(1–2), 67–96 (2012)

Logic-Based Program Synthesis and Transformation
25th International Symposium, LOPSTR 2015, Siena,
Italy, July 13-15, 2015. Revised Selected Papers
Falaschi, M. (Ed.)
2015, XVII, 385 p. 88 illus. in color., Softcover
ISBN: 978-3-319-27435-5