Electronic Notes in Theoretical Computer Science 48 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume48.html` pp. 1 – 25

On the Relations between Disjunctive and Linear Logic Programming

Marco Bozzano[1] Giorgio Delzanno[2] Maurizio Martelli[3]

Dipartimento di Informatica e scienze dell'Informazione
Università di Genova, Via Dodecaneso 35
16146 Genova, Italy

Abstract

In this paper we investigate the relationship between Disjunctive Logic Programming as defined in [13] and a subset of Linear Logic, namely the fragment of LinLog [2] which corresponds to Andreoli and Pareschi's LO [3]. We analyze the two languages both from a *top-down*, operational perspective, and from a *bottom-up*, semantical one. From a proof-theoretical perspective, we show that, modulo a simple mapping between classical and linear connectives, LO can be viewed as a *sub-structural* fragment of DLP in which the rule of *contraction* is forbidden on the right-hand side of sequents. We also prove that LO is strictly more expressive than DLP in the propositional case. From a semantical perspective, after recalling the definition of a bottom-up fixpoint semantics for LO we have given in our previous work [5], we show that DLP fixpoint semantics can be viewed as an abstraction of the corresponding LO semantics, defined over a suitable abstract domain. We prove that the abstraction is *correct* and *complete* in the sense of [6,8]. Finally, we show that the previous property of the semantics is strictly related to proof-theoretical properties of the classical and linear logic fragments underlying DLP and LO.

1 Introduction

Disjunctive Logic Programming (DLP) [13] and Linear Logic Programming (LLP) [11] are among the more interesting extensions of the classical theory of Horn logic, underlying languages like Prolog. The motivations behind the introduction of these two paradigms look quite different. On the one hand, disjunctive logic programming has been introduced in order to represent *uncertain beliefs*. On the other hand, linear logic programming has been introduced in order to add *state-based computations* to pure Prolog programs. A closer

[1] Email: `bozzano@disi.unige.it`
[2] Email: `giorgio@disi.unige.it`
[3] Email: `martelli@disi.unige.it`

look at their formal definition reveals however very interesting connections. Let us focus on the linear logic programming language LO [3] (*Linear Objects*), a fragment of LinLog [2] which is perhaps the first proposal of 'linear' extension of Prolog [11]. Both DLP and LO programs extend Horn programs allowing clauses with *multiple* heads: in DLP, the head of a clause consists of a *classical* disjunction of atomic formulas, whereas, in LO, it consists of a *multiplicative* disjunction (see [9]) of atomic formulas. The differences between the meaning of classical and linear connectives become clear by looking at the operational semantics of the two languages. In DLP, a resolution step is extended so as to work over positive clauses (*sets/disjunctions* of atoms). Implicit *contraction* steps are applied over the selected clause. In contrast, being in a sub-structural logic in which contraction is forbidden, LO resolution has the same effect as *multiset* rewriting (applied to collections of atomic formulas).

Based on this intuition, in this paper we will investigate the view of LO as a *sub-structural* formulation of DLP, to formally compare the strength and weakness of the two languages. As mathematical tools, we will use *proof theory* and *abstract interpretation*. Proof theory allows us to compare the *top-down* semantics of the two paradigms working in a uniform setting (i.e. provability in a sequent calculus with or without structural rules). Abstract interpretation allows us to extend the comparison to the *bottom-up* evaluation of programs. More specifically, by exhibiting a Galois connection between the semantic domains of DLP and LO, we will describe the semantics of DLP programs as an abstraction of the semantics of LO programs. Furthermore, using the theory of abstract interpretation and the concept of *completeness* [6,8] we will discuss the quality of the resulting abstraction. This step is based on a new fixpoint semantics for LO we have defined in our recent work [4,5]. Finally, we will discuss the relation between proof-theoretical properties (*permutability* of rules) and properties of the abstraction (*completeness*).

The view of DLP as an abstraction of LO is appealing for several reasons. First of all, it opens the possibility of using techniques developed for DLP for the analysis of LO programs. Furthermore, it shows that the paradigm of DLP could have unexpected applications as a framework to reason about properties of Petri Nets, a well-known formalism for concurrent computations [10], as discussed in [5].

1.1 Plan of the paper

After introducing some preliminaries in Section 2, in Section 3 we recall the main concepts of disjunctive logic programming, and we reformulate its operational semantics in a sequent calculus context. In Section 4 we introduce the language LO, while in Section 5 we compare DLP and LO proof theories. In Section 6 we study the relations between DLP and LO via abstract interpretation, and we discuss the properties of the resulting abstraction. In Section

7, we discuss the connection between the notion of completeness in abstract interpretation and proof theory. Finally, in Section 8 we discuss conclusions and future work.

2 Preliminaries

In this paper we will extensively use operations on multisets. We will consider a fixed signature Σ, defining a finite set of object constants, function symbols and predicate symbols. The set of *ground* terms over Σ will be denoted T_Σ, while the set of *atoms* over T_Σ will be denoted A_Σ. Multisets over A_Σ will be hereafter called *facts*, and symbolically noted as $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$. A multiset with (possibly duplicated) elements $A_1, \ldots, A_n \in A_\Sigma$ will be simply indicated as $\{A_1, \ldots, A_n\}$, overloading the usual notation for sets.

A multiset $\mathcal{A}$ is uniquely determined by a map $Occ : A_\Sigma \to \mathbb{N}$ such that $Occ_{\mathcal{A}}(A)$ is the number of occurrences of A in $\mathcal{A}$. Multisets are ordered according to the *multiset inclusion* relation $\preccurlyeq$ defined as follows: $\mathcal{A} \preccurlyeq \mathcal{B}$ if and only if $Occ_{\mathcal{A}}(A) \leq Occ_{\mathcal{B}}(A)$ for every $A \in A_\Sigma$. The *empty* multiset is denoted ϵ and is such that $Occ_\epsilon(A) = 0$ for every $A \in A_\Sigma$, and $\epsilon \preccurlyeq \mathcal{A}$ for any $\mathcal{A}$. The *multiset union* $\mathcal{A}, \mathcal{B}$ (alternatively written $\mathcal{A}+\mathcal{B}$ when ',' is ambiguous) is such that $Occ_{\mathcal{A},\mathcal{B}}(A) = Occ_{\mathcal{A}}(A) + Occ_{\mathcal{B}}(A)$ for every $A \in A_\Sigma$. We also define a special operation $\bullet$ to compute the *least upper bound* of two multisets with respect to $\preccurlyeq$. Namely, $\mathcal{A} \bullet \mathcal{B}$ is such that $Occ_{\mathcal{A}\bullet\mathcal{B}}(A) = max(Occ_{\mathcal{A}}(A), Occ_{\mathcal{B}}(A))$ for every $A \in A_\Sigma$. Finally, we will use the notation $\mathcal{A}^n$, where n is a natural number, to indicate $\mathcal{A} + \ldots + \mathcal{A}$ (n times).

In the rest of the paper we will use $\Delta, \Theta, \ldots$ to denote multisets of possibly compound formulas. Given two multisets Δ and Θ, Δ, Θ indicates multiset union, as before, and $\Delta, \{G\}$ is written simply Δ, G.

Finally, let $T : \mathcal{I} \to \mathcal{I}$ be an operator defined over a complete lattice $\langle \mathcal{I}, \sqsubseteq \rangle$. We define $T\uparrow_0 = \emptyset$, where $\emptyset$ is the bottom element, $T\uparrow_{k+1} = T(T\uparrow_k)$ for all $k \geq 0$, and $T\uparrow_\omega = \bigsqcup_{k=0}^{\infty} T\uparrow_k$, where $\bigsqcup$ is the least upper bound wrt. $\sqsubseteq$. Furthermore, we use *lfp*(T) to denote the *least fixpoint* of T.

3 Disjunctive Logic Programming

A *disjunctive logic program* as defined in [13] is a finite set of clauses

$$A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m,$$

where $n \geq 1$, $m \geq 0$, and A_i and B_i are atomic formulas. A *disjunctive goal* is of the form $\leftarrow C_1, \ldots, C_n$, where C_i is a *positive clause* (i.e. a disjunction of atomic formulas) for $i : 1, \ldots, n$. To make the language symmetric, in this paper we will consider extended clauses of the form

$$A_1 \vee \ldots \vee A_n \leftarrow C_1 \wedge \ldots \wedge C_m$$

containing positive clauses in the body. Following [13], we will deal with first-order programs by considering the set of *ground* instances of program clauses. Given a program P, we will use the notation $\mathit{Gnd}(P)$ to indicate the set of ground instances of clauses in P. Moreover, we will identify positive clauses with *sets* of atoms.

In order to define the operational and denotational semantics of DLP, we need the following definitions.

Definition 3.1 (Disjunctive Herbrand Base) The *disjunctive Herbrand base* of a program P, for short DHB_P, is the set of all positive clauses formed by an arbitrary number of *distinct* ground atoms.

Definition 3.2 (Disjunctive Interpretation) A subset I of the disjunctive Herbrand base DHB_P is called a disjunctive Herbrand interpretation.

Definition 3.3 (Ground SLO-derivation) Let P be a DLP program. An SLO-derivation of a ground goal G from P consists of a sequence of goals $G_0 = G, G_1, \ldots$ such that for all $i \geq 0$, G_{i+1} is obtained from $G_i = \leftarrow (C_1, \ldots, C_m, \ldots, C_k)$ as follows:

- $C \leftarrow D_1 \wedge \ldots \wedge D_q$ is a ground instance of a clause in P such that C is contained in C_m (the selected clause);
- G_{i+1} is the goal $\leftarrow (C_1, \ldots, C_{m-1}, D_1 \vee C_m, \ldots, D_q \vee C_m, C_{m+1}, \ldots, C_k)$.

Notice that when the body of the program clause is empty, G_{i+1} is equal to $\leftarrow (C_1, \ldots, C_{m-1}, C_{m+1}, \ldots, C_k)$.

Definition 3.4 (SLO-refutation) Let P be a DLP program. An SLO-refutation of a ground goal G from P is an SLO-derivation $G_0, G_1, \ldots, G_k$ s.t. G_k consists of the empty clause only.

As SLD-resolution for Horn programs, SLO-resolution gives us a procedural interpretation of DLP programs. The operational semantics is defined then as follows:

$$O_P^{\mathit{dlp}} = \{C \mid C \in \mathit{DHB}_P, \ \leftarrow C \text{ has an SLO-refutation } \}.$$

As for Horn programs, it is possible to define a fixpoint semantics via the following operator.

Definition 3.5 (T_P^{dlp} Operator) Given a DLP program P and $I \subseteq \mathit{DHB}_P$,

$$T_P^{\mathit{dlp}}(I) = \{ \ C' \cup C_1 \cup \ldots \cup C_n \mid C' \leftarrow D_1 \wedge \ldots \wedge D_n \in \mathit{Gnd}(P), \\ D_i \cup C_i \in I, \ i : 1, \ldots, n \ \}$$

Note that in the previous definition we have implicitly considered positive clauses as sets of atoms. The operator T_P^{dlp} is monotonic and continuous on the lattice of interpretations ordered wrt. set inclusion. Based on this property, the fixpoint semantics is defined as $F_P^{dlp} = lfp(T_P^{dlp}) = T_P^{dlp}\uparrow_\omega$. As shown in [13], the fixpoint semantics is complete with respect to the operational semantics, i.e. for all $C \in O_P^{dlp}$ there exists $C' \in F_P^{dlp}$ s.t. C' implies C. Note that for two ground clauses C and C', C implies C' if and only if $C \subseteq C'$. This suggests that interpretations can also be ordered wrt. subset inclusion for their elements, i.e., $I \sqsubseteq J$ if and only if for all $A \in I$ there exists $B \in J$ such that $B \subseteq A$ (B implies A). In the rest of the paper we will consider the latter ordering.

Example 3.6 Consider the disjunctive program $P = \{r(a),\ p(X) \vee q(X) \leftarrow r(X)\}$ and the auxiliary predicate t. Then, $DHB_P = \{r(a), p(a), q(a), t(a), p(a) \vee r(a), p(a) \vee q(a), p(a) \vee q(a) \vee r(a), \ldots\}$. Furthermore, the goal $G_0 = \leftarrow (p(a) \vee q(a) \vee t(a))$ has the refutation $G_0, G_1 = \leftarrow (p(a) \vee q(a) \vee t(a) \vee r(a)), G_2$ where G_2 consists of the empty clause only. The fixpoint semantics of P is $F_P^{dlp} = \{\{r(a)\}, \{p(a), q(a)\}\}$. Note that $\{p(a), q(a)\}$ is contained in $\{p(a), q(a), t(a)\}$ (i.e. $p(a) \vee q(a)$ implies $p(a) \vee q(a) \vee t(a)$).

The definition of the T_P^{dlp} operator can be reformulated in such a way that its input and output domains contain multisets instead of sets of atoms (i.e. we can consider interpretations which are sets of multisets of atoms). In fact, we can always map a multiset to its *underlying set*, i.e. the set containing the elements with multiplicity greater than zero, and, vice versa, a set can be viewed as a multiset in which each element has multiplicity equal to one. A formulation based on multisets will make the comparison with the fixpoint operator for LO (see Section 6) easier, as this latter is defined on that kind of domains. The disjunctive Herbrand base and T_P^{dlp} operator definitions can be reformulated as follows (in the following we will always refer to these definitions).

Definition 3.7 (Disjunctive Herbrand Base) The *disjunctive Herbrand base* of a program P, for short DHB_P, is the set of all multisets formed by an arbitrary number of ground atoms, each with multiplicity at most one.

Definition 3.8 (α mapping) Let α be the following function on multisets of atoms. Given a multiset $\mathcal{A}$, $\alpha(\mathcal{A})$ is the multiset such that for every $A \in A_\Sigma$, $Occ_{\alpha(\mathcal{A})}(A) = 0$ if $Occ_{\mathcal{A}}(A) = 0$, $Occ_{\alpha(\mathcal{A})}(A) = 1$ otherwise (i.e. we abstract a multiset with the corresponding set).

We note in passing that α definition is related to the notion of *smallest factor* of a clause given in [13].

Definition 3.9 (T_P^{dlp} Operator) Given a DLP program P and $I \subseteq DHB_P$,

$$T_P^{dlp}(I) = \{\ \alpha(\widehat{C'} + C_1 + \ldots + C_n\ ) \mid C' \leftarrow D_1 \wedge \ldots \wedge D_n \in Gnd(P), \\ \widehat{D_i} + C_i \in I,\ i : 1, \ldots, n\ \}$$

$$\frac{}{P \Rightarrow \mathtt{tt}, \Delta}\ \mathtt{tt}_r \qquad \frac{P \Rightarrow A_1, \ldots, A_n, \Delta}{P \Rightarrow A_1 \vee \ldots \vee A_n, \Delta}\ \vee_r$$

$$\frac{P \Rightarrow C_1, \widehat{H}, \mathcal{A} \quad \ldots \quad P \Rightarrow C_m, \widehat{H}, \mathcal{A}}{P \Rightarrow \widehat{H}, \mathcal{A}}\ bc$$

with the proviso that $H \leftarrow C_1 \wedge \ldots \wedge C_m \in \; Gnd(P)$

Fig. 1. A proof system for DLP.

The notation $\widehat{\cdot}$ in the previous definition is used to formally map positive clauses (i.e. $\vee$-disjunctions of atoms) to multisets of atoms (whereas in Definition 3.5 the mapping from clauses to sets of atoms was left implicit). Without loss of generality, from now on we will make the further assumption that in clauses like $A_1 \vee \ldots \vee A_n \leftarrow C_1 \wedge \ldots \wedge C_m$, the A_i's are all *distinct* and each C_j consist of *distinct* atoms. This will simplify the embedding of DLP clauses into linear logic (see Definition 5.1).

3.1 A Proof System for DLP

We will now give a proof-theoretical presentation, based on sequent calculus, for DLP. Its formulation is directly related to the definition of SLO-derivation (see Definition 3.3). In order to simplify the comparison with LO, we introduce an explicit constant `tt` for *true* and we write *unit clauses* (i.e. with empty body) with the syntax $A_1 \vee \ldots \vee A_n \leftarrow \mathtt{tt}$. The definitions given in the previous section can be adapted in a straightforward manner. The resulting language can be described by the following grammar:

$$\mathbf{H} ::= \mathbf{A}_1 \vee \ \ldots \vee \ \mathbf{A}_n$$

$$\mathbf{D} ::= \mathbf{H} \leftarrow \mathbf{G} \mid \mathbf{D} \wedge \mathbf{D}$$

$$\mathbf{G} ::= \mathbf{H}_1 \wedge \ \ldots \wedge \ \mathbf{H}_n \mid \mathtt{tt}$$

where $\mathbf{A}_i$ is an atomic formula. A DLP program P is a $\mathbf{D}$-clause, whereas DLP goals are represented (modulo '$\leftarrow$') as $\mathbf{G}$-formulas. A proof system for DLP is presented in Figure 1 (again, we use the notation $\widehat{\cdot}$ to map $\vee$-disjunctions of atoms to multisets). Here, in a sequent $P \Rightarrow \Delta$, it is understood that P is a DLP program and Δ is a *set* of goals. Besides, $\mathcal{A}$ denotes a set of atomic formulas, while H denotes an $\mathbf{H}$-formula.

Without loss of generality, we have limited topmost goals to positive clauses only (the execution of a goal which is a conjunction of clauses can be simulated by introducing a fictitious atom and adding a clause to the program). The reader should convince himself/herself that the system in Figure 1 cor-

$$\frac{}{P \Rightarrow \texttt{tt}, \Delta}\ \texttt{tt}_r \qquad \frac{P \Rightarrow A_1, \ldots, A_n, \Delta}{P \Rightarrow A_1 \vee \ldots \vee A_n, \Delta}\ \vee_r \qquad \frac{P \Rightarrow A, A, \mathcal{A}}{P \Rightarrow A, \mathcal{A}}\ ctr_r$$

$$\frac{P \Rightarrow C_1, \mathcal{A} \quad \ldots \quad P \Rightarrow C_m, \mathcal{A}}{P \Rightarrow \widehat{H}, \mathcal{A}}\ bc$$

$$\textit{with the proviso that}\ \ H \leftarrow C_1 \wedge \ldots \wedge C_m \in\ Gnd(P)$$

Fig. 2. Modified proof system for DLP.

rectly models SLO-provability. The state of a computation, which in DLP is represented by a conjunctive goal $\leftarrow C_1, \ldots, C_k$, in the context of sequent calculus is represented by a derivation tree, whose frontier (open leaves) are the goals still to be proved $(C_1, \ldots, C_k)$. Rule *bc* models a *backchaining* step, and corresponds to a step of SLO-derivation; it can be applied only if the current context consists of atomic formulas only (indicated by $\mathcal{A}$). Rule $\vee_r$ formalizes the intuition that positive clauses are sets of atoms (i.e. it permits *exchange* on the right-hand side of sequents). Clauses having the form $a_1 \vee\ \ldots \vee\ a_n \leftarrow \texttt{tt}$ play the same role as unit clauses in DLP, in fact a backchaining step over such a clause leads to *success* independently of the current context $\mathcal{A}$, as shown in the following scheme:

$$\frac{\dfrac{}{P \Rightarrow \texttt{tt}, \mathcal{A}}\ \texttt{tt}_r}{P \Rightarrow A_1, \ldots, A_n, \mathcal{A}}\ bc$$

$$\textit{provided}\quad A_1 \vee\ \ldots \vee\ A_n \leftarrow \texttt{tt} \in Gnd(P)$$

This observation leads us to the following property, which states that the *weakening* rule is *admissible* in DLP.

Proposition 3.10 (Admissibility of the weakening rule in DLP) *Given a DLP program P and two sets of goals Δ and Δ' such that $\Delta \subseteq \Delta'$, if $P \Rightarrow \Delta$ then $P \Rightarrow \Delta'$.*

Proof. By simple induction on the structure of DLP proofs. □

In order to make a comparison between DLP and LO, we will now present a slight variation of the system in Figure 1. This new system can be proved equivalent to the previous one, and is directly related to the system for LO we will present in Section 5. Here, the right-hand side of sequents is a *multiset* of goals, and the structural rule of *contraction* is explicitly added. The system is presented in Figure 2. Adding the rule of contraction makes it possible a slight modification of the rule *bc* in which the atoms in the head of the relevant program clause are discharged in the upper sequents. The equivalence of the two systems can be proved by simple induction on the structure of derivations in the systems. The proof entails showing that the *contraction* and *weakening* rules are *admissible* in the first system. We have the following result.

Proposition 3.11 *Given a DLP program P and a goal G, there exists an SLO-refutation of G from P if and only if $P \Rightarrow G$ is provable in the system of Figure 1 if and only if $P \Rightarrow G$ is provable in the system of Figure 2.*

Proof. (sketch) For the sake of clarity, let us distinguish between provability in the systems in Figure 1 and 2 using the notation $\Rightarrow_1$ and $\Rightarrow_2$, respectively. We also use the notation bc_1 and bc_2 for the respective backchaining rules. The equivalence between SLO-refutations and $\Rightarrow_1$ provability is by definition (see Definitions 3.3 and 3.4). Note that the *exchange* rule (i.e. mapping between positive clauses and sets of atoms) is implicit in Definition 3.3. Also, rule $\mathtt{tt}_r$ can be derived from Definition 3.3 in the special case of clauses with empty body (i.e. unit clauses). The equivalence between $\Rightarrow_1$ and $\Rightarrow_2$ can be proved by showing that ctr_r and bc_2 rules are admissible in the first system, and bc_1 is admissible in the second one:

- admissibility of ctr_r in the first system follows from the fact that the set $A, A, \mathcal{A}$ is equal to $A, \mathcal{A}$ (in other words, contraction is implicit in the choice of sets in sequent representation);
- bc_2 admissible in the first system: suppose $H \leftarrow C_1 \wedge \ldots \wedge C_m \in \ Gnd(P)$; if $P \Rightarrow_1 C_1, \mathcal{A}, \ldots, P \Rightarrow_1 C_m, \mathcal{A}$, then by Proposition 3.10 we get $P \Rightarrow_1 C_1, \widehat{H}, \mathcal{A}, \ldots, P \Rightarrow_1 C_m, \widehat{H}, \mathcal{A}$, therefore it follows that $P \Rightarrow_1 \widehat{H}, \mathcal{A}$;
- bc_1 admissible in the second system: suppose $H \leftarrow C_1 \wedge \ldots \wedge C_m \in \ Gnd(P)$; if $P \Rightarrow_2 C_1, \widehat{H}, \mathcal{A}, \ldots, P \Rightarrow_2 C_m, \widehat{H}, \mathcal{A}$, then $P \Rightarrow_2 \widehat{H}, \widehat{H}, \mathcal{A}$; it follows that $P \Rightarrow_2, \widehat{H}, \mathcal{A}$ by applying ctr_r to every element in $\widehat{H}$.

□

As a corollary, given a positive clause $C = A_1 \vee \ldots \vee A_n$ we have that $P \Rightarrow C$ is provable if and only if there exists $C' \in F_P^{dlp}$ such that $C' \subseteq C$.

4 The Linear Logic Programming Language LO

Linear logic [9] can be viewed as a refinement of classical logic where the use of weakening and contraction is allowed only for formulas within the scope of special modalities (the exponentials '!' and '?'). This way, formulas (without modalities) can be viewed as 'resources' that can be used only a limited number of times in a proof. Among the possible applications, linear logic turned out to be the natural foundation for extensions of logic programming with a notion of *state* (in this setting, a collection of bounded-use formulas) [11].

LO [3] is a logic programming language based on linear logic. Its mathematical foundations lie on a proof-theoretical presentation of a fragment of linear logic called LinLog [2], and comprising the linear connectives $\multimap$ (*linear implication*), & (*additive conjunction*), ⅋ (*multiplicative disjunction*), and the constant $\top$ (*additive identity*). In the propositional case LO consists of the

$$\frac{}{P \vdash \top, \Delta}\ \top_r \qquad \frac{P \vdash G_1, G_2, \Delta}{P \vdash G_1 \parr G_2, \Delta}\ \parr_r \qquad \frac{P \vdash G_1, \Delta \quad P \vdash G_2, \Delta}{P \vdash G_1 \,\&\, G_2, \Delta}\ \&_r$$

$$\frac{P \vdash G, \mathcal{A}}{P \vdash \widehat{H}, \mathcal{A}}\ bc$$

$$\textit{with the proviso that}\ \ H \multimapinv G \ \in\ Gnd(P)$$

Fig. 3. A proof system for LO

following class of formulas:

$$\mathbf{D} \ ::= \ \mathbf{A}_1 \parr \ldots \parr \mathbf{A}_n \ \multimapinv\ \mathbf{G} \ \mid \ \mathbf{D} \ \& \ \mathbf{D}$$

$$\mathbf{G} \ ::= \ \mathbf{G} \ \parr \ \mathbf{G} \ \mid \mathbf{G} \ \& \ \mathbf{G} \ \mid \ \mathbf{A} \ \mid \ \top$$

Here $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and $\mathbf{A}$ range over propositional symbols from a fixed signature Σ. **G**-formulas correspond to *goals* to be evaluated in a given program. **D**-formulas correspond to multiple-headed *program clauses*. An LO program is a **D**-formula. Let P be the program $C_1 \,\&\, \ldots \,\&\, C_n$. The execution of a multiset of **G**-formulas $G_1, \ldots, G_k$ in P corresponds to a goal-driven proof for the two-sided LO sequent

$$P \vdash G_1, \ldots, G_k.$$

The LO sequent $P \vdash G_1, \ldots, G_k$ is an abbreviation for the following two-sided linear logic sequent:

$$!C_1, \ldots, !C_n \ \rightarrow\ G_1, \ldots, G_k.$$

The formula $!F$ on the left-hand side of a sequent indicates that F can be used in a proof an arbitrary number of times. This implies that an LO Program can also be viewed as a *set of reusable clauses.* According to this view, the operational semantics of LO is given via the *uniform* (goal-driven) proof system defined in Figure 3. In Figure 3, P is a set of implicational clauses, $\mathcal{A}$ denotes a multiset of atomic formulas, whereas Δ denotes a multiset of **G**-formulas. A sequent is provable if all branches of its proof tree terminate with instances of the $\top_r$ axiom. The proof system of Figure 3 is a specialization of more general uniform proof systems for linear logic like Andreoli's focusing proofs [2] and Forum [12]. The rule *bc* denotes a backchaining (resolution) step. Note that we have overloaded the $\widehat{\cdot}$ notation (which in Definition 3.9 was used to map $\vee$-disjunctions of atoms into multisets) to indicate the mapping of $\parr$-disjunctions of atoms into multisets. Also note that *bc* can be executed only if the right-hand side of the current LO sequent consists of atomic formulas. Thus, LO clauses behave like *multiset* rewriting rules.

LO clauses having the form $a_1 \parr \ \ldots \parr \ a_n \multimapinv \top$ play the same role as the unit clauses of DLP programs. In fact, a backchaining step over such a clause leads to *success* independently of the current context $\mathcal{A}$, exactly as in DLP:

$$\dfrac{\dfrac{}{P \vdash \top, \mathcal{A}}\ \top_r}{P \vdash A_1, \ldots, A_n, \mathcal{A}}\ bc$$

$$\textit{provided} \quad A_1 \mathbin{⅋} \ldots \mathbin{⅋} A_n \mathrel{\circ\!-} \top \in \textit{Gnd}(P)$$

An analogous result to that of Proposition 3.10 holds. Thus, LO can be viewed as an *affine* fragment of linear logic. Note that weakening and contraction are both admissible on the left-hand side (i.e. on the program part) of LO sequents.

Proposition 4.1 (Admissibility of the weakening rule in LO) *Given an LO program P and two multisets of goals Δ and Δ' such that $\Delta \preccurlyeq \Delta'$, if $P \vdash \Delta$ then $P \vdash \Delta'$.*

Proof. By simple induction on the structure of LO proofs. □

Example 4.2 Let P be the LO program consisting of the clauses

1. $a \mathrel{\circ\!-} b \mathbin{⅋} c$
2. $b \mathrel{\circ\!-} (d \mathbin{⅋} e) \mathbin{\&} f$
3. $c \mathbin{⅋} d \mathrel{\circ\!-} \top$
4. $e \mathbin{⅋} e \mathrel{\circ\!-} b \mathbin{⅋} c$
5. $c \mathbin{⅋} f \mathrel{\circ\!-} \top$

and consider an initial goal e, e. A proof for this goal is shown in Figure 4, where we have denoted by $bc^{(i)}$ the application of the backchaining rule over clause number i of P. The proof proceeds as follows. Using clause 4., to prove e, e we have to prove $b \mathbin{⅋} c$, which, by LO $⅋_r$ rule, reduces to prove b, c. At this point we can backchain over clause 2., and we get the new goal $(d \mathbin{⅋} e) \mathbin{\&} f, c$. By applying $\&_r$ rule, we get two separate goals $d \mathbin{⅋} e, c$ and f, c. The first, after a reduction via $⅋_r$ rule, is provable by means of clause (axiom) 3., while the latter is provable directly by clause (axiom) 5. Note that $\top$ succeeds in a non-empty context (i.e. containing e) in the left branch. A similar proof shows that the goal a is also provable from P.

5 A Proof-Theoretical Perspective: from DLP to LO

Let us now go back to disjunctive logic programming. In Section 3, we have shown that the operational (top-down) semantics of DLP can be presented in terms of a proof system with an explicit contraction rule. Furthermore, the weakening rule is admissible in DLP. A natural question arises: what happens

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash e,\top}\,\top_r}{P \vdash d,e,c}\,bc^{(3)}}{P \vdash d \parr e, c}\,\parr_r \qquad \dfrac{\dfrac{}{P \vdash \top}\,\top_r}{P \vdash f, c}\,bc^{(5)}}{P \vdash (d \parr e) \,\&\, f, c}\,\&_r}{P \vdash b, c}\,bc^{(2)}}{P \vdash b \parr c}\,\parr_r}{P \vdash e, e}\,bc^{(4)}$$

Fig. 4. An LO proof for the goal e, e in the program of Example 4.2

if we forbid the use of the structural rules in DLP? Is the resulting language related to the extension of LP based on linear logic? We will answer these questions in two steps. We will first embed DLP into linear logic by means of the following mapping.

Definition 5.1 ($\lceil\cdot\rceil$ mapping) The mapping $\lceil\cdot\rceil$ from DLP formulas into linear logic (LO) formulas is defined by induction on the structure of DLP formulas as follows: $\lceil F \vee G\rceil = \lceil F\rceil \parr \lceil G\rceil$, $\lceil F \wedge G\rceil = \lceil F\rceil \,\&\, \lceil G\rceil$, $\lceil F \leftarrow G\rceil = \lceil F\rceil \multimapinv \lceil G\rceil$, $\lceil \mathtt{tt}\rceil = \top$.

Based on this mapping, the grammar for D- and G-formulas can be rewritten as follows:

$$\mathbf{H} ::= \mathbf{A}_1 \parr \ \ldots \parr \mathbf{A}_n$$

$$\mathbf{D} ::= \mathbf{H} \multimapinv \mathbf{G} \mid \mathbf{D} \,\&\, \mathbf{D}$$

$$\mathbf{G} ::= \mathbf{H}_1 \,\&\, \ \ldots \,\&\, \mathbf{H}_n \mid \top$$

where $\mathbf{A}_i$ is an atomic formula. By definition of DLP program, the image of $\lceil\cdot\rceil$ returns a class of LO programs where both the head and the disjuncts in the body have no repeated occurrences of the same atom, Syntactically, the resulting language is a proper subset of the language LO [3] presented in Section 4, where, in addition to the previous formulas, the connectives $\parr$ and $\&$ can be arbitrarily nested within LO goals.

It remains to analyze the relation between the operational semantics of the two languages. The operational semantics of LO specialized to the fragment under consideration is given via the proof system defined in Figure 5, where $\mathcal{A}$ denotes a multiset of atomic formulas, H denotes an $\mathbf{H}$-formula, and the right-hand side of sequents is a multiset of goals. The proof system of Figure 5 is a specialization of the proof system of Figure 3. As before, the *bc* rule can be executed only if the right-hand side of the current LO sequent consists of atomic formulas, and a sequent is provable if all branches of its proof tree

$$\frac{}{P \vdash \top, \Delta}\ \top_r \qquad \frac{P \vdash A_1, \ldots, A_n, \Delta}{P \vdash A_1 \parr \ldots \parr A_n, \Delta}\ \parr_r$$

$$\frac{P \vdash C_1, \mathcal{A} \quad \ldots \quad P \vdash C_m, \mathcal{A}}{P \vdash \widehat{H}, \mathcal{A}}\ bc$$

with the proviso that $H \circ\!- C_1 \,\&\, \ldots \,\&\, C_m \in \ Gnd(P)$

Fig. 5. A specialized proof system for a fragment of LO

terminate with instances of the $\top_r$ axiom.

The comparison between DLP and LO proof systems is now straightforward. By looking at the system for DLP in Figure 2 and at the one for LO in Figure 5, we can see that, modulo a direct encoding of classical connectives into linear ones, DLP is obtained from LO by adding the structural rule of *contraction*. Equivalently, LO can be viewed as a sub-structural logic of DLP in which *contraction* is forbidden. We note that the *weakening* rule, on the contrary, is allowed both in DLP and in LO. By denoting with $\vdash_{\mathsf{C}}$ the provability in the language LO augmented with a contraction rule analogous to that of Figure 2, and with $\vdash_{\mathsf{LL}}$ provability in linear logic, we can then state the following proposition. Note that the the exponentials ! and ? in the last sequent are needed to augment linear logic provability with both weakening and contraction (on both sides of the sequent).

Proposition 5.2 *Given a DLP program P, $P \Rightarrow G$ is provable in DLP if and only if $\lceil P \rceil \vdash_{\mathsf{C}} \lceil G \rceil$ is provable in LO, if and only if $!\lceil P \rceil \vdash_{\mathsf{LL}} ?\lceil G \rceil$ is provable in LL.*

5.1 A Comparison between Propositional DLP and LO

As a consequence of the proof-theoretical analysis carried out so far, we now present some simple results concerning the propositional fragments of DLP and LO. These results state that in the propositional case LO is strictly more expressive than DLP.

Proposition 5.3 *There exists no translation $|\cdot|$ from the classes of* propositional *LO programs and goals into those of* propositional *DLP, which preserves provability, i.e. such that for any P and G, $P \vdash G$ is provable if and only if $|P| \Rightarrow |G|$ is provable.*

Proof. (sketch) In the propositional case the set of logically distinct clauses is finite (i.e. $a \vee a \equiv a$ etc.). So is the disjunctive Herbrand base. Let us consider the infinite sequence of LO goals a, $a \parr a$, $a \parr a \parr a$, etc. From the observation above, there exists $n > m$ such that $|a \parr \ldots \text{(n-times)} \ldots \parr a|$ is logically equivalent (in DLP) to $|a \parr \ldots \text{(m-times)} \ldots \parr a|$. It remains to exhibit a program P that distinguishes the two goals in LO. We define P as

$a \parr \ldots$ n-times $\ldots \parr a \leftarrow \top$. Clearly, the first goal is provable from P in LO, while the latter is not. Whatever translation $|P|$ of P we give we will obtain that either the translations (that coincide) of both goals are provable in DLP or they are both non-provable. □

On the other hand, the translation from DLP to LO is straightforward. We have the following proposition.

Proposition 5.4 *There exists a translation $|\cdot|$ from the classes of* propositional *DLP programs and goals into those of* propositional *LO, which preserves provability, i.e. such that for any P and G, $P \Rightarrow G$ is provable if and only if $|P| \vdash |G|$ is provable.*

Proof. It is sufficient to take $|G| = \lceil G \rceil$, and $|P| = \lceil P \rceil \cup P_{ctr}$, where $\lceil P \rceil$ is the program obtained by taking the $\lceil \cdot \rceil$ translation (see Definition 5.1) of every clause in P, and P_{ctr} is the finite set of clauses $a_i \multimapinv a_i \parr a_i$, one for every propositional symbol a_i in the language. These clauses give a direct encoding of contraction. □

6 A Semantic-based Comparison

In our recent work [4,5], we have shown that LO programs are amenable of a fixpoint semantics that characterizes the set of provable LO goals. As the corresponding semantics for Horn and disjunctive programs, in the propositional case the fixpoint semantics for LO can be computed in a finite number of steps. The fixpoint semantics of LO allows us to investigate in more depth the relationship between LO and DLP. For this purpose, we can employ the mathematical tools provided by *abstract interpretation* [6], and in particular the notion of *completeness* that can be used to estimate the precision of an abstraction.

Informally, the comparison between LO and DLP fixpoint semantics is based on the abstraction that maps multisets into sets of atomic formulas (positive clauses). This abstraction induces a Galois connection between the semantic domains of DLP and LO. We prove that the fixpoint semantics of the translation of an LO program in DLP is a correct abstraction of the fixpoint semantics of the original LO program. Furthermore, we show that this abstraction is not *fully complete* with respect to LO semantics. In a *fully complete* abstraction the result of interleaving the application of the abstract fixpoint operator with the abstraction α coincides with the abstraction of the concrete fixpoint operator. For a *complete* abstraction, a similar relation holds for fixpoints, i.e., the fixpoint of the abstract operator coincides with the abstraction of the fixpoint of the concrete one. We will show that this weaker property of *completeness* holds for the abstraction for the entire class of LO programs, thus extending the result presented in [5], which was limited to the class of LO programs without conjunction in the body.

Let us start by giving some definitions used to introduce the fixpoint semantics for LO (a detailed presentation can be found in [5]).

Definition 6.1 (LO Herbrand base) The Herbrand base HB_P of a propositional LO program P over Σ is the set of all multisets of ground atoms over A_Σ.

The previous definition is the same as Definition 3.7, the only difference being that elements are not required to have multiplicity at most one. Given a multiset $\mathcal{A}$, we denote by $[\![A]\!]$ the *upward closure* of $\mathcal{A}$, i.e. $\{\mathcal{B} \mid \mathcal{A} \preccurlyeq \mathcal{B}\}$ (e.g. $[\![\{a,b\}]\!] = \{\{a,b\},\{a,a,b\},\{a,b,b\},\{a,b,c\},\ldots\}$). The following definition formalizes the intuition that interpretations are always implicitly considered upward-closed sets. Interpretations whose upward closures are equal are therefore considered equivalent. This is justified by Prop. 4.1 (admissibility of the weakening rule).

Definition 6.2 (LO Herbrand interpretation) The lattice $\langle \mathcal{I}, \sqsubseteq \rangle$ of LO Herbrand interpretations is defined as follows:

- $\mathcal{I} = \mathcal{P}(HB_P)/\simeq$ where $I \simeq J$ if and only if $[\![I]\!] = [\![J]\!]$;
- $[I]_\simeq \sqsubseteq [J]_\simeq$ if and only if for all $\mathcal{B} \in I$ there exists $\mathcal{A} \in J$ such that $\mathcal{A} \preccurlyeq \mathcal{B}$;
- the bottom element is the empty set $\emptyset$, the top element is the $\simeq$-equivalence class of the singleton $\{\epsilon\}$ (ϵ=empty multiset, $\epsilon \preccurlyeq \mathcal{A}$ for any $\mathcal{A} \in HB_P$);
- the least upper bound $I \sqcup J$ is the $\simeq$-equivalence class of $I \cup J$.

The equivalence $\simeq$ allows us to reason modulo *redundancies*. For instance, any $\mathcal{A}$ is redundant in $\{\epsilon, \mathcal{A}\}$, which, in fact, is equivalent to $\{\epsilon\}$. For ease of notation, in the rest of the paper we will identify an interpretation I with its class $[I]_\simeq$. The definition for the fixpoint T_P^{lo} operator, for the subclass of LO programs we are considering in this paper (i.e. without nesting of connectives), can be specialized as follows:

Definition 6.3 (T_P^{lo} operator) Given an LO program P and an interpretation I, the operator T_P^{lo} is defined as follows:

$$\begin{aligned} T_P^{lo}(I) = \{\widehat{H} + (\mathcal{C}_1 \bullet \ldots \bullet \mathcal{C}_n) \mid H \multimapinv D_1 \,\&\, \ldots \,\&\, D_n \in Gnd(P),\\ \forall i: 1,\ldots,n,\ \widehat{D_i} + \mathcal{C}_i \in I\} \quad \cup \quad \{\widehat{H} \mid H \multimapinv \top \in P\} \end{aligned}$$

This definition is similar to Definition 3.9. Here, we have two different operations which are both modeled by multiset union in Definition 3.9: multiset union ($+$) and *least upper bound* of multisets ($\bullet$). For instance, $\{a,b\} + \{b,c\} = \{a,b,b,c\}$, while $\{a,b\} \bullet \{b,c\} = \{a,b,c\}$. Notice however that the two operations are equivalent under α (see Definition 3.8), i.e. $\alpha(\mathcal{A} + \mathcal{B}) = \alpha(\mathcal{A} \bullet \mathcal{B})$. Also note that the case for unit clauses is left implicit in Definition 3.9.

It can be proved that the operator T_P^{lo} is monotonic and continuous over the lattice of Herbrand interpretations (ordered wrt. $\sqsubseteq$). Thus, the fixpoint semantics of an LO program P can be defined as

$$F_P^{lo} = \bigsqcup_{i=0}^{\omega} T_P^{lo}{\uparrow}_i \ .$$

The relation between operational semantics and fixpoint semantics can be stated as follows.

Proposition 6.4 (Completeness [5]) *Let P be an LO program and $\mathcal{A} \in HB_P$, then $P \vdash \mathcal{A}$ is provable if and only if there exists $\mathcal{A}' \in F_P^{lo}$ s.t. $\mathcal{A}' \preccurlyeq \mathcal{A}$.*

Example 6.5 Consider the LO program $P = \{r(a) \multimapinv \top,\ p(X) \parr p(X) \parr q(X) \multimapinv r(X)\}$ and the auxiliary predicate t. Then, the Herbrand base HB_P is defined as follows $\{\{r(a)\}, \{p(a)\}, \{q(a)\}, \{t(a)\}, \{p(a), r(a)\}, \{p(a), p(a), r(a)\}, \ldots\}$. The LO sequent $P \vdash p(a), p(a), q(a), t(a)$ is provable in LO. In fact, by applying a backchaining step we obtain the sequent $P \vdash r(a), t(a)$. Now, we can apply the fact $r(a) \multimapinv \top$ and we obtain an instance of the axiom $\top_r$. The fixpoint semantics of P is as follows $F_P^{lo} = \{\{r(a)\}, \{p(a), p(a), q(a)\}\}$. Note that $\{p(a), p(a), q(a)\} \preccurlyeq \{p(a), p(a), q(a), t(a)\}$.

We note that, as proved in [5], the fixpoint semantics of a propositional LO program can be computed in finitely many steps. Though the Herbrand base is always infinite (it contains all possible multisets over A_Σ, therefore it is infinite even if A_Σ is finite), this property is ensured by the *well-quasi ordering* of the lattice of Herbrand interpretations [5].

In the following, we will use the previous semantics and the framework of abstract interpretation to give an alternative, bottom-up account, of the relation between DLP and LO. First of all, we will give a brief summary of some notions related to the theory of abstract interpretation.

6.1 Abstract Interpretation

Abstract Interpretation [6,7] is a classical framework for semantics approximation which is used for the construction of semantics-based program analysis algorithms. Given a semantics and an abstraction of the language constructors and standard data, abstract interpretation determines an abstract representation of the language which is, by construction, sound with respect to the standard semantics. This new representation enables the calculation of the abstract semantics in finite time, although it implies some loss of precision. We recall here some key concepts in abstract interpretation, which the reader can find in [6,7,8].

Given a concrete semantics $\langle C, T_P \rangle$, specified by a *concrete domain* (complete lattice) C and a (monotone) fixpoint operator $T_P : C \rightarrow C$, the abstract semantics can be specified by an *abstract domain* A and an abstract fixpoint operator $T_P^{\#}$. In this context, program semantics is given by $\mathit{lfp}(T_P)$, and its abstraction is $lfp(T_P^{\#})$. The concrete and abstract semantics $S = \langle C, T_P \rangle$ and $S^{\#} = \langle A, T_P^{\#} \rangle$ are related by a Galois connection $\langle \alpha, C, A, \gamma \rangle$, where $\alpha : C \rightarrow A$ and $\gamma : A \rightarrow C$ are called *abstraction* and *concretization* functions, respectively.

$S^{\#}$ is called a *sound* abstraction of S if for all P, $\alpha(lfp(T_P)) \leq_A lfp(T_P^{\#})$. This condition is implied by the strongest property of *full soundness*, which requires that $\alpha \circ T_P \leq_A T_P^{\#} \circ \alpha$. The notions of *completeness* and *full completeness* are dual with respect to those of soundness. Namely, $S^{\#}$ is a (fully) *complete* abstraction of S if for all P, $(T_P^{\#} \circ \alpha \leq_A \alpha \circ T_P)$ $lfp(T_P^{\#}) \leq_A \alpha(lfp(T_P))$. Often, the notion of completeness is assumed to include soundness (i.e. we impose '=' in the previous equations). It is well-known[7] that the abstract domain A induces a *best* correct approximation of T_P, which is given by $\alpha \circ T_P \circ \gamma$, and that it is possible to define a (fully) complete abstract operator $T_P^{\#}$ if and only if the best correct approximation is (fully) complete. It can be proved that for a fixed concrete semantics, (full) completeness of an abstract interpretation only depends on the choice of the abstract domain. The problem of achieving a (fully) complete abstract interpretation starting from a correct one, by either refining or simplifying the abstract domain, is studied in [8].

We conclude by observing that an equivalent presentation of abstract interpretation is based on *closure operators*[7], i.e. functions from a concrete domain C to itself which are *monotone*, *idempotent* and *extensive.* This approach provides independence from specific representations of abstract domain's objects (the abstract domain is given by the image, i.e. the set of fixpoints, of the closure operator).

We are now in the position of connecting the LO (concrete) semantics with the DLP (abstract) semantics.

6.2 DLP as an Abstraction of LO

We will define the abstract interpretation by resorting to closure operators. According to this view, we can define the abstract interpretation as a closure operator on the lattice $\mathcal{I}$, the domain of LO interpretations of Definition 6.2. In fact, we can consider disjunctive interpretations as the subclass of $\mathcal{I}$ comprising all sets in $\mathcal{I}$ (i.e. all multisets in $\mathcal{I}$ with element multiplicity at most one). In other words, the class of disjunctive interpretations is an abstract domain for $\mathcal{I}$. We recall that in $\mathcal{I}$ the ordering of interpretations is defined as follows: $I \sqsubseteq J$ if and only if for all $B \in I$ there exists $A \in J$ such that A is a submultiset of B (i.e., for disjunctive interpretations, $A \subseteq B$). We give the following definitions.

Definition 6.6 (Abstract Interpretation from LO to DLP) The abstract interpretation is defined by the closure operator $\alpha : \mathcal{I} \rightarrow \mathcal{I}$ such that for every $I \in \mathcal{I}$, $\alpha(I) = \{\alpha(\mathcal{A}) \mid \mathcal{A} \in I\}$.

Note that we have overloaded the operator α of Definition 3.8 to indicate its extension to interpretations.

Definition 6.7 (Abstract semantics) The abstract fixpoint semantics is given by $\mathit{lfp}(T_P^{\#})$, where the abstract fixpoint operator $T_P^{\#}$ is defined as $(\alpha \circ T_P^{lo})$.

According to [7], $\alpha \circ T_P^{lo}$ is the best correct approximation of the concrete fixpoint operator T_P^{lo}, for the fixed abstraction α. The abstraction α transforms multisets into sets by forgetting multiple occurrences of atoms. The $T_P^{\#}$ operator is indeed the T_P^{dlp} operator for disjunctive logic programs of Definition 3.9 (we defined it over domains containing multisets for this purpose). In other words, T_P^{dlp} input domain corresponds to the abstract domain which is given by the set of fixpoints, i.e. the image, of the closure operator α. The operations $\bullet$ (*least upper bound* of multisets) and $+$ (multiset union) used in the definition of T_P^{lo} are interchangeable, as already noted before, because of the subsequent application of the operator α, and both correspond to multiset union in the definition of T_P^{dlp}. Also, the case for unit clauses is left implicit in the definition of T_P^{dlp}. We have the following results.

Proposition 6.8 (DLP is an abstraction of LO) *Given a DLP program P and a* disjunctive *interpretation I, $T_P^{dlp}(I) = T_{\lceil P \rceil}^{\#}(I)$.*

Proof. By definitions. □

Proposition 6.9 (DLP is a sound abstraction of LO) *For every LO program P, the abstract semantics is a fully sound abstraction of the concrete semantics, that is, for every interpretation I, $\alpha(T_P^{lo}(I)) \sqsubseteq T_P^{\#}(\alpha(I))$.*

Proof. As $T_P^{\#} = \alpha \circ T_P^{lo}$ and $I \sqsubseteq \alpha(I)$, the proposition follows by monotonicity of $T_P^{\#}$. □

The previous result implies *soundness*, i.e. $\alpha(\mathit{lfp}(T_P^{lo})) \sqsubseteq \mathit{lfp}(T_P^{\#})$. The strong property of *full completeness* does not hold for the abstraction. To see why, take as a counterexample the single clause $a \multimap b$ and the interpretation I with the single multiset $\{b, b\}$. Then, $\alpha(T_P^{lo}(I)) = \{\{a, b\}\}$, $T_P^{\#}(\alpha(I)) = \{\{a\}\}$, and $T_P^{\#}(\alpha(I)) \not\sqsubseteq \alpha(T_P^{lo}(I))$.

In [5] we have proved a preliminary result, namely that the abstraction is complete for the subclass of LO programs whose clauses contain *at most* one conjunct in the body (i.e. conjunction is forbidden). This subclass is particularly interesting, as discussed in [5] and briefly mentioned in Section 7.2, because it can be used to encode Petri Nets. We address now the problem of proving completeness of the abstraction for the entire class of LO programs (we remind the reader that we are actually considering the subclass of LO programs corresponding to DLP programs as defined in [13], i.e. without nesting of connectives). We will give an indirect proof of this fact based on the connection between the notion of completeness and proof theory. The proof will be presented in Section 7.1.

$$\frac{}{P \Rightarrow \mathtt{tt}, \Delta}\ \mathtt{tt}_r \qquad \frac{P \Rightarrow A, A, \mathcal{A}}{P \Rightarrow A, \mathcal{A}}\ \mathit{ctr}_r$$

$$\frac{P \Rightarrow \widehat{C_1}, \mathcal{A} \quad \ldots \quad P \Rightarrow \widehat{C_m}, \mathcal{A}}{P \Rightarrow \widehat{H}, \mathcal{A}}\ \mathit{bc}$$

$$\textit{with the proviso that}\ \ H \leftarrow C_1 \wedge \ldots \wedge C_m \in\ \mathit{Gnd}(P)$$

Fig. 6. An equivalent proof system for DLP.

7 A Proof-Theoretical Account of Completeness

In this section we discuss the relations between the notion of *completeness* of the abstraction and the proof-theoretic notion of *permutability* of rules, thus creating a bridge between Sections 3.1 and 5, on the one hand, and Section 6.2, on the other hand.

First of all, let us reformulate the proof system for DLP presented in Figure 2. The new system is presented in Figure 6. Here, we have directly plugged the rule $\vee_r$ inside the backchaining rule. The resulting system can be easily proved equivalent in the following sense: a disjunction of atoms $A_1 \vee \ldots \vee A_n$ is provable in the first system if and only if the multiset $A_1, \ldots, A_n$ is provable in the second. This new proof system corresponds more directly to the formulation of the T_P^{lo} operator. In particular, if we consider the system which comprises only rules $\mathtt{tt}_r$ and *bc* in Figure 6 (i.e. without contraction) we get, modulo the usual mapping between classical and linear connectives, a proof system for LO. Moreover, the α step in the definition of the T_P^{dlp} operator is the analogous of the contraction rule in the proof system in Figure 6 (to be precise, α corresponds to multiple applications of ctr_r).

To complete the circle, let us now consider the property of completeness, i.e. $\alpha(\mathit{lfp}(T_P^{lo})) = \mathit{lfp}(\alpha \circ T_P^{lo})$. The first expression, $\alpha(\mathit{lfp}(T_P^{lo}))$, represents the abstraction of the set of goals which can be proved from the proof system for LO (without contraction), i.e. the set of goals which can be proved using the rule of contraction only at the root of the proof tree. The second expression, $\mathit{lfp}(\alpha \circ T_P^{lo})$, represents the set of goals which can be proved by freely using the contraction rule (to be precise, an application of *bc* is followed by every possible application of ctr_r). Thus, in this view, we have that completeness of the abstraction is equivalent to the following proof-theoretic property:

> By restricting the set of proofs in the system in Figure 6 to those in which the contraction rule is applied only at the root of the proof tree, the set of theorems (provable multisets) of the system does not change.

In other words, completeness implies that ctr_r and *bc* are *permutable*, in the sense that if an application of ctr_r occurs above an application of *bc*, it is possible to find a different proof in which contraction is pushed down, below the application of *bc*.

7.1 An indirect proof of completeness of the abstraction

We now present a proof of completeness of the abstraction based on the proof-theoretic property of rule permutability discussed above. In particular, we will show an effective algorithm which, given a proof for a multiset $\mathcal{A}$ in the system of Figure 6 as input, builds a proof of the same multiset in which the contraction rule is applied only at the root of the proof tree. For illustrative purposes, we will keep the discussion about the algorithm somewhat informal, but a completely formal presentation can be derived in a straightforward way from our description. First of all, let us consider a fixed program P and a generic proof for a multiset $\mathcal{A}$ in the system in Figure 6. Except for the leaves, which are instances of the axiom scheme $\mathtt{tt}_r$, the proof tree is built using only *bc* and ctr_r steps. The algorithm operates by *recursively* pushing down applications of the contraction rule starting from the top and going down to the root. To be precise, the algorithm operates on *bunches* of contractions, i.e. on groups of *consecutive* applications of the contraction rule. In particular, given a proof, we can isolate an application of *bc* s.t.:

- at least one of its departing branches contains an application of ctr_r (otherwise, we have finished);
- for every departing branch, either it does not contain any application of ctr_r, or it is of the form

  $$\dfrac{\begin{array}{c}\alpha\\ \vdots\\ P \Rightarrow \Delta'\end{array}}{P \Rightarrow \Delta}\ ctr_r{}^*$$

  where $ctr_r{}^*$ represents a bunch of contractions and α does not contain any application of ctr_r.

In other words, the algorithm selects for reduction an application of *bc* which has the form

$$\dfrac{\dfrac{\begin{array}{c}\alpha_1\\ \vdots\\ P \Rightarrow \Delta'_1\end{array}}{P \Rightarrow \Delta_1}\ ctr_r{}^{p_1} \quad \begin{array}{c}\vdots\\ \ldots\end{array} \quad \dfrac{\begin{array}{c}\alpha_m\\ \vdots\\ P \Rightarrow \Delta'_m\end{array}}{P \Rightarrow \Delta_m}\ ctr_r{}^{p_m}}{P \Rightarrow \Delta}\ bc$$

where $ctr_r{}^{p_i}$ represents a bunch of p_i contractions, all p_i's (but one) can be possibly null, and $\alpha_1,\ldots,\alpha_m$ do not contain ctr_r. The result of an application of a single step of the algorithm is to push down the bunches of contractions in the above scheme, transforming that proof fragment in the following:

$$\begin{array}{c}
\begin{array}{ccc}\alpha'_1 & & \alpha'_m \\ \vdots & & \vdots \\ & \dots & \end{array} \\
\hline
P \Rightarrow \Delta' \\
\hline
P \Rightarrow \Delta
\end{array}
\begin{array}{l} \\ \\ bc \\ {ctr_r}^p \\ \end{array}$$

where the natural number p is related to $p_1,\dots,p_m$, and $\alpha'_1,\dots,\alpha'_m$ do not contain ctr_r.

Now, before showing how the single step operates, let us clarify that this is sufficient in order to prove that all contractions can be pushed down to the root. In other words, we must show an inductive measure on proofs which guarantees that the recursive application of a reduction step is well founded. There is a very simple induction measure which can be used for this purpose. Namely, given a proof α, let us call *reducible* the applications of *bc* which are preceded in the proof tree by at least an application of ctr_r (i.e. there is a path from a leaf to that application of *bc* which includes at least one application of ctr_r) and *reduced* the remaining applications of *bc*. The induction measure is simply given by the number of reducible applications of *bc* in the proof tree. After every step, this number decreases by one. In fact, it is clear that the *bc* application involved in the step was reducible (by hypothesis) and it is reduced afterwards (for the hypothesis on $\alpha'_1,\dots,\alpha'_m$). Furthermore, it is also clear that the other applications of *bc* in the tree are not affected (in particular, an application of *bc* which was reduced is still reduced). Note that the proof fragments $\alpha'_1,\dots,\alpha'_m$ in general can be much bigger than $\alpha_1,\dots,\alpha_m$, but this is harmless because they do not contain any ctr_r application. The claim that all ctr_r applications can be pushed down to the root then follows from the following facts: i) it is always possible to choose an occurrence of *bc* for reduction, *satisfying the requirements described before*, as soon as the proof tree contains reducible occurrences of *bc*; ii) after every step, the number of reducible applications of *bc* decreases; iii) if the number of reducible applications of *bc* in a proof tree becomes 0, then all ctr_r applications are grouped in a bunch at the root of the proof tree.

Let us now describe how a single step of the algorithm works. Let us consider a reducible application of *bc* satisfying the requirements described before, and let $H \leftarrow C_1 \wedge \ldots \wedge C_m \in \ Gnd(P)$ be the relevant instance of program clause. The fragment of proof tree involved in the transformation has the following form:

$$\dfrac{\dfrac{\begin{matrix}\alpha_1\\ \vdots\\ P \Rightarrow \Delta'_1\end{matrix}}{P \Rightarrow \widehat{C_1}, \mathcal{A}}\; ctr_r{}^{p_1} \qquad \vdots\!\!\ldots \qquad \dfrac{\begin{matrix}\alpha_m\\ \vdots\\ P \Rightarrow \Delta'_m\end{matrix}}{P \Rightarrow \widehat{C_m}, \mathcal{A}}\; ctr_r{}^{p_m}}{P \Rightarrow \widehat{H}, \mathcal{A}}\; bc$$

For simplicity of illustration, let us assume for the moment that all *ctr*$_r$ applications in this fragment are performed over atoms in $\widehat{C_1}$,...,$\widehat{C_m}$. Contractions over the context $\mathcal{A}$ are much simpler to address, as we will describe later on. Now, we can safely assume that our proof fragment has the form

$$\dfrac{\dfrac{\begin{matrix}\alpha_1\\ \vdots\\ P \Rightarrow \widehat{C_1}^{n_1}, \mathcal{A}\end{matrix}}{P \Rightarrow \widehat{C_1}, \mathcal{A}}\; ctr_r{}^{p_1} \qquad \vdots\!\!\ldots \qquad \dfrac{\begin{matrix}\alpha_m\\ \vdots\\ P \Rightarrow \widehat{C_m}^{n_m}, \mathcal{A}\end{matrix}}{P \Rightarrow \widehat{C_m}, \mathcal{A}}\; ctr_r{}^{p_m}}{P \Rightarrow \widehat{H}, \mathcal{A}}\; bc$$

where p_i is given by $n_i - 1$ times the cardinality of the multiset $\widehat{C_i}$. That is, we are assuming that $n_i - 1$ applications of *ctr*$_r$ are performed over every element of $\widehat{C_i}$. This is possible because, if we take $n_i - 1$ to be the maximum number of applications of *ctr*$_r$ which are performed over a single element in $\widehat{C_i}$, then the original Δ'_i will be a submultiset of $\widehat{C_i}^{n_i}, \mathcal{A}$, and, thanks to Proposition 4.1, if Δ'_i is provable in LO than $\widehat{C_i}^{n_i}, \mathcal{A}$ is provable as well (note that LO corresponds to the system in Figure 6 *without contraction*). Now, the transformed proof will have the following form (we show only a small fragment):

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\; bc}{P \Rightarrow \widehat{C_1^2}, \widehat{H}^{z-2}, \mathcal{A}} \quad \ldots \quad \dfrac{\vdots\; bc}{P \Rightarrow \widehat{C_1}, \widehat{C_m}, \widehat{H}^{z-2}, \mathcal{A}}}{P \Rightarrow \widehat{C_1}, \widehat{H}^{z-1}, \mathcal{A}}\; bc \qquad \ldots \quad \dfrac{\vdots\; bc}{P \Rightarrow \widehat{C_m}, \widehat{H}^{z-1}, \mathcal{A}}}{P \Rightarrow \widehat{H}^{z}, \mathcal{A}}\; bc}{P \Rightarrow \widehat{H}, \mathcal{A}}\; ctr_r{}^{q}$$

The idea is to perform q applications of *ctr*$_r$ below the relevant *bc*, where q is given by $z - 1$ times the cardinality of $\widehat{H}$, and we choose z to be $n_1 + n_2 + \ldots + n_m - m + 1$. The goal $\widehat{H}^z, \mathcal{A}$ contains z occurrences of $\widehat{H}$, therefore, after performing one step of backchaining, we will get m departing branches which contain $z - 1$ occurrences of $\widehat{H}$. We can recursively apply backchaining (always using the original program clause) on each of the sub-branches, until all z

occurrences of $\widehat{H}$ have been consumed. What we get is a (partial) proof tree made only of *bc* applications; this proof tree has depth z and every node has exactly m children. Now, let us consider the leaves of this (partial) proof tree. One single leaf is on a path leading to $P \Rightarrow \widehat{H}^z, \mathcal{A}$, and at every step one of the occurrences of $\widehat{H}$ is rewritten in one multiset taken from $\widehat{C_1},\ldots,\widehat{C_m}$. Therefore, the leaf will contain a combination of z multisets taken from $\widehat{C_1},\ldots,\widehat{C_m}$. Now, if the leaf contains at least n_1 occurrences of the multiset $\widehat{C_1}$, we have finished. In fact, this means that $\widehat{C_1}^{n_1}, \mathcal{A}$ is a submultiset of the leaf, and, by the hypothesis on α_1 and Proposition 4.1, this implies that there exists a proof α_1' for the leaf which does not contain contractions. If the leaf contains less than n_1 occurrences of $\widehat{C_1}$, then it will contain at least $z-(n_1-1) = n_2+\ldots+n_m-m+2$ occurrences of multisets taken from $\widehat{C_2},\ldots,\widehat{C_m}$. We can repeat the same kind of reasoning (formally, the proof is by induction on m), and the last alternative will be that the leaf contains at least $n_m-m+m = n_m$ occurrences of the multiset $\widehat{C_m}$, from which we can conclude. The same reasoning can be repeated for every leaf, thus we can complete the partial proof tree using only *bc* and $\mathtt{tt}_r$ rules.

Finally, we note that contractions performed over the multiset $\mathcal{A}$ which is not directly involved in the backchaining step are even simpler to address. In fact, $\mathcal{A}$ is simply passed unchanged to every sub-branch, therefore we can transfer these contraction steps below the original *bc* application. It is sufficient to perform, for every element in $\mathcal{A}$, as many ctr_r steps as the maximum number of such steps performed over that element, where the maximum is taken among the original m bunches of contractions.

On the basis of the above discussion and of the connection between LO and DLP provability and the proof system in Figure 6, we can then state the following lemma.

Lemma 7.1 *A multiset $\mathcal{A}$ is provable in DLP if and only if there exists a multiset $\mathcal{A}'$ s.t. $\mathcal{A}'$ is provable in LO and $\mathcal{A} = \alpha(\mathcal{A}')$.*

The result of completeness of the abstraction then easily follows.

Proposition 7.2 (DLP is a complete abstraction of LO) *Let P be an LO program. Then $lfp(T_P^{\#}) \sqsubseteq \alpha(lfp(T_P^{lo}))$.*

Proof. Given $\mathcal{A} \in lfp(T_P^{\#})$, by Lemma 7.1 there exists $\mathcal{A}'$ provable in LO s.t. $\mathcal{A} = \alpha(\mathcal{A}')$. As $\mathcal{A}'$ provable in LO, by Proposition 6.4 there exists $\mathcal{A}'' \in lfp(T_P^{lo})$ s.t. $\mathcal{A}'' \preccurlyeq \mathcal{A}'$. Then, $\alpha(\mathcal{A}'') \in \alpha(lfp(T_P^{lo}))$ and $\alpha(\mathcal{A}'') \preccurlyeq \alpha(\mathcal{A}') = \mathcal{A}$. □

We conclude the section with a brief discussion about the application of LO and DLP to Petri Nets.

7.2 LO, Petri Nets, and DLP

As shown in [5], the class of propositional LO programs with one conjunct in the body is equivalent to Petri Nets. In fact, a multiset rewriting rule can be used to describe the effect of firing a Petri Net transition. For instance, the clause $a \parr b \parr b \multimapinv c \parr c$ can be interpreted as the Petri Net transition that removes one token from place a, two tokens from place b, and adds two tokens to place c. As a consequence, a (possibly infinite) execution of a restricted LO program denotes an execution of the corresponding Petri Net. The initial goal can be viewed as the initial marking of the Petri Net, while the set of axioms can be used to implicitly give a description of the reachable states. As a consequence of the admissibility of weakening (Proposition 4.1), an axiom represents an infinite, upward-closed set of markings. As a consequence, provability in LO can be used to axiomatize *coverability* problems of Petri Nets. What does coverability mean? Let us first recall the following notions. Given two markings $\vec{m}$ and $\vec{n}$, $\vec{m}$ *covers* $\vec{n}$ if and only if, for every place, the number of tokens in $\vec{m}$ is greater or equal than the number of tokens in $\vec{n}$. The coverability problem for a Petri Net N is formulated as follows: given the initial marking $\vec{m_0}$ and a marking $\vec{n}$, is there a marking $\vec{m}$ reachable from $\vec{m_0}$ that covers $\vec{n}$? Let P be the LO program that represents the Petri Net N, and let $\mathcal{A}_0$ and $\mathcal{A}$ the multisets that represent $\vec{m_0}$ and $\vec{n}$, respectively. The coverability problem for N, $\vec{m_0}$ and $\vec{n}$ can be formulated as the following LO provability question: is there $\mathcal{A}' \preccurlyeq \mathcal{A}_0$ such that $\mathcal{A}'$ is provable in the program P enriched with the clause $H \multimapinv \top$, where $H = A_1 \parr \ldots \parr A_n$ for $\mathcal{A} = \{A_1, \ldots, A_n\}$? This problem can be solved via the bottom-up computation of all logical consequences of $P \cup \{H \multimapinv \top\}$. Coverability problems are strictly related to verification of *safety properties* of concurrent systems expressed as Petri Nets (see e.g. [1]). In fact, in many practical examples *unsafe states* can be represented as upward-closed sets (e.g. the set of states where there are *at least two processes in the critical section* are the typical bad states of a mutual exclusion algorithm). In all these situations, the verification of the corresponding safety property can thus be reduced to the dual of a coverability problem for the minimal violations. More specifically, this kind of problems can be solved by computing the set of predecessors $Pre^*(U)$ of the unsafe states, i.e., the set of logical consequences obtained by expressing U as an LO axiom. The results of Section 6.2 show that the fixpoint semantics of DLP can be used to approximate $Pre^*(U)$. Completeness implies that all properties that are preserved by the abstraction can be checked equivalently over the concrete and the abstract domain. In our setting the kind of properties that satisfy this requirement can be informally characterized as *at-least-one* properties (e.g. the set of markings which contain *at least one* token in a given place).

8 Conclusions and Future Work

In this paper we have used operational (proof-theoretical) and declarative (fixpoint semantic) tools to compare the paradigm of DLP with the paradigm of linear logic programming (more specifically, the language LO). The comparison is based on the abstraction that maps multisets into sets of atomic formulas (positive clauses). This abstraction induces a Galois connection between the semantic domains of DLP and LO. The abstraction is not fully complete. In a *fully complete* abstraction the result of interleaving the application of the abstract fixpoint operator with the abstraction α coincides with the abstraction of the concrete fixpoint operator. For a *complete* abstraction, a similar relation holds for fixpoints, i.e., the fixpoint of the abstract operator coincides with the abstraction of the fixpoint of the concrete one. In proof-theoretical terms, the abstraction corresponds to allowing unlimited use of contraction in the proof system for LO. In this view, we have shown that the property of completeness implies that contraction steps permute with the other inference rules, in the sense that given a proof for a goal G it is always possible to find an alternative proof for G in which the contraction steps are performed only at the root of the proof tree.

We think that our study can help in getting a new insight into the relations between provability in fragments of classical and linear logic, on the one hand, and into the relations between *top-down* and *bottom-up* semantics, on the other hand. We hope that our research will also give rise to new ideas for the analysis of LO programs. As an example, it could be interesting to study weak notions of negation for LO that are based on the negation of DLP. Also, we have mentioned another possible application of DLP operational and fixpoint semantics, namely Petri Nets analysis. Finally, there is still some on-going work concerning the relation between DLP and LO in the setting of abstract interpretation. In particular, we are studying a direct proof of completeness of the abstraction and its relationship with the indirect proof presented in this paper. Also, formally studying the complexity of the transformation between generic DLP proofs and restricted DLP proofs (as shown in the proof of completeness) could be worth in order to quantify the gain obtained by proving properties on the abstract domain rather than on the concrete one.

9 Acknowledgments

We would like to thank the anonymous referees of AGP 2000 for useful comments and suggestions that helped us to improve the paper.

References

[1] Abdulla, P. A., K. Cerāns, B. Jonsson and Y.-K. Tsay, *General Decidability Theorems for Infinite-State Systems*, in: *Proc. 11th Annual IEEE Int. Symposium on Logic in Computer Science (LICS'96)* (1996), pp. 313–321.

[2] Andreoli, J.-M., *Logic Programming with Focusing Proofs in Linear Logic*, Journal of Logic and Computation **2** (1992), pp. 297–347.

[3] Andreoli, J.-M. and R. Pareschi, *Linear Objects: Logical Processes with Built-In Inheritance*, New Generation Computing **9** (1991), pp. 445–473.

[4] Bozzano, M., G. Delzanno and M. Martelli, *A Bottom-up Semantics for Linear Logic Programs*, in: M. Gabbrielli and F. Pfenning, editors, *Proc. 2nd International Conference on Principles and Practice of Declarative Programming (PPDP'00)* (2000), pp. 92–102.

[5] Bozzano, M., G. Delzanno and M. Martelli, *An Effective Fixpoint Semantics for Linear Logic Programs*, Theory and Practice of Logic Programming (2001), To appear.

[6] Cousot, P. and R. Cousot, *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fix-Points*, in: *Proc. 4th Symposium on Principles of Programming Languages (POPL'77)* (1977), pp. 238–252.

[7] Cousot, P. and R. Cousot, *Systematic Design of Program Analysis Frameworks*, in: *Proc. 6th Symposium on Principles of Programming Languages (POPL'79)* (1979), pp. 269–282.

[8] Giacobazzi, R. and F. Ranzato, *Completeness in Abstract Interpretation: A Domain Perspective*, in: M. Johnson, editor, *Proc. 6th International Conference on Algebraic Methodology and Software Technology (AMAST'97)*, Lecture Notes in Computer Science **1349** (1997), pp. 231–245.

[9] Girard, J., *Linear logic*, Theoretical Computer Science **50:1** (1987), pp. 1–102.

[10] Karp, R. M. and R. E. Miller, *Parallel Program Schemata*, Journal of Computer and System Sciences **3** (1969), pp. 147–195.

[11] Miller, D., *A Survey of Linear Logic Programming*, Computational Logic: The Newsletter of the European Network of Excellence in Computational Logic **2** (1995), pp. 63–67.

[12] Miller, D., *Forum: A Multiple-Conclusion Specification Logic*, Theoretical Computer Science **165** (1996), pp. 201–232.

[13] Minker, J., A. Rajasekar and J. Lobo, *Theory of Disjunctive Logic Programs*, in: J. Lassez and G. Plotkin, editors, *Computational Logic. Essays in Honor of Alan Robinson*, MIT Press, 1991 pp. 613–639.