An abstract framework for logic programming semantics

Thanos Tsouanas[1,2]

Instituto Metrópole Digital
Universidade Federal do Rio Grande do Norte
Natal, Brasil

Abstract

In this article an *abstract framework for logic programming semantics* is defined and various known semantic approaches are placed within this framework. This way, semantics become formal mathematical objects of study. In developing this framework, we introduce the general notion of a *truth value space*, on which we evaluate formulæ. As expected, the booleans form the canonical example of a truth value space, but we need to consider much more general ones when dealing with negation-as-failure. Then we define a *semantic operator* which transforms any given abstract semantics of a non-disjunctive language to a semantics of the "corresponding" disjunctive one. We exhibit the correctness of this transformation by proving that it preserves equivalences of semantics, and we present some applications of it. In particular, three new semantics for disjunctive programs with negation are constructed: a first model-theoretic semantics for infinite such programs, and two novel game-theoretic ones for finite ones.

Keywords: logic programming, denotational semantics, game semantics, semantic frameworks, disjunctive logic programming

1 Introduction

1.1 What is a logic program?

A logic program can be loosely described as a set of rules of the form

$$\texttt{this} \leftarrow \texttt{that},$$

read as "*this* holds, if *that* holds", or "I can solve *this* problem, if I know how to solve *that* one". Depending on what restrictions we impose on `this` (the *head* of the rule) and `that` (the *body*), we enable or disable features of the resulting programming language. In its simplest form, a rule looks like this:

$$\mathtt{a} \leftarrow \mathtt{b}_1 \,, \cdots, \mathtt{b}_m, \tag{LP}$$

where commas on the right stand for conjunctions. One extension allows *negations* to appear in body rules: [3]

$$\mathtt{a} \leftarrow \mathtt{b}_1 \,, \cdots, \mathtt{b}_m \,, {\sim}\mathtt{c}_1 \,, \cdots, {\sim}\mathtt{c}_k. \tag{LPN}$$

But the extension in which we are mostly interested in this text is the appearance of *disjunctions* in heads:

$$\mathtt{a}_1 \vee \cdots \vee \mathtt{a}_n \leftarrow \mathtt{b}_1 \,, \cdots, \mathtt{b}_m. \tag{DLP}$$

[1] PPgMAE/UFRN (Universidade Federal do Rio Grande do Norte, Brasil). Research funded by the projects: **RECRE ANR Blanc 11-BS02-010** and **CNPq project 400506/2014-9**.

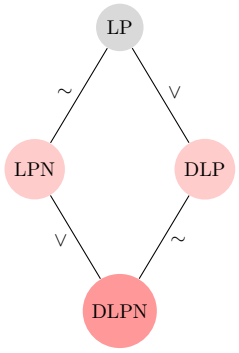
[2] Email: thanos@imd.ufrn.br

[3] By negation, we mean *negation-as-failure* (see [Cla78]).

This enables us to express uncertainty and to derive ambiguous information. Finally, one can consider both extensions simultaneously, by allowing *both* negations in bodies *and* disjunctions in heads:

$$\mathtt{a}_1 \vee \cdots \vee \mathtt{a}_n \leftarrow \mathtt{b}_1 , \cdots , \mathtt{b}_m , \sim\mathtt{c}_1 , \cdots , \sim\mathtt{c}_k . \tag{DLPN}$$

These extensions are summarized in the figure below:



LP : the so-called *definite* logic programs (no negations, no disjunctions);

DLP : disjunctive logic programs;

LPN : logic programs with negation;

DLPN : disjunctive logic programs with negation.

1.2 What is a semantics?

This question has more answers than one could hope for, and to my knowledge this is the first attempt to give an abstract yet formal answer. But let us first review a few different approaches. [4]

Model-theoretic semantics.
The standard denotational semantics for LP is provided by a specific two-valued model, the so-called *least Herbrand model*, with which the reader is assumed to be familiar, along with related notions: Herbrand universe, Herbrand base, Herbrand interpretation, etc. Consult [vEK76] or [Llo87] for further information. [5] For LPN, the semantics we have in mind is supplied by the *many-valued well-founded model*, defined in [VGRS91]. [6] Instead of a single least model, for DLP, we use a *set of minimal models* for the semantics, as defined in [Min82] and extensively studied in [LMR92]. Finally, a satisfactory, infinite-valued, model-theoretic semantics for DLPN was recently defined in [CPRW07]. However, this semantics only deals with *finite propositional* programs, and thus it is not adequate when we are dealing with first-order ones (see the remark at the end of this section). *In this article we will obtain "for free" a semantics for DLPN that is able to handle* infinite *propositional programs instead, thus filling this gap.*

Procedural or operational semantics.
The actual implementation of each of the above languages is usually given by refutation processes. Given a goal, the system tries to disprove it by constructing a counterexample: a proof that the program together with the goal is an inconsistent set of rules. Traditionally, such proofs make use of some inference rule based on *resolution*. This might be, for example, SLD resolution in the case of LP, or SLI resolution for DLP. In this work, we do not touch this operational side of semantics; see [Apt90] for the non-disjunctive and [LMR92] for the disjunctive cases.

Before turning to the next approach, game semantics, one should have a clear understanding of the nature of the aforementioned methods. On one side, we have the denotational semantics (e.g. model-theoretic and their fixpoint characterizations). These provide us with a notion of *correctness* for every possible answer to a goal that we might give to our program. On the operational side, the procedural semantics provide a construction

[4] This is not by any means a complete presentation. Two particularly interesting methods that we omit are *proof-theoretic* semantics (e.g. [MN12], [MNPS91], [AP91], [And92], [PR05], and [MS06]) as well as *coalgebraic* ones (e.g. [BM09], [KMP10], [KP11], [KPS13], and [BZ13]).

[5] Frequently, to construct the model-theoretic semantics we use an *immediate consequence operator* (traditionally denoted by $T_{\mathcal{P}}$) associated with each program $\mathcal{P}$, and look at its fixpoints; see [Llo87]. We will not concern ourselves with fixpoints in this article. An excellent survey of fixpoint semantics is [Fit99].

[6] This is one of the two mainly accepted semantics in this case. The other one is *stable model semantics* which was proposed in [GL88], and assigns to each program $\mathcal{P}$ a certain set of well-behaved models, called *stable models*. A program may have zero, one, or more such models, which is in this approach the price to pay in order to support negation. This school of programming eventually lead to what is now known as *answer set programming* (ASP). Consult [Gel08] for further information.

of an answer to our question (the so-called *computed answer*), and this answer has to be correct. Conversely, such a procedure is expected to be able to derive all of the answers that the denotational semantics considers correct. We then say that the procedural semantics is sound and complete with respect to the denotational one.

Game semantics.

Here we adopt an anthropomorphic point of view, and treat each program as a set of rules for a game, in which two players compete against each other with respect to the truth of a given goal. One player, who has the rôle of the "Doubter" doubts the goal's truthness, while the other player, being the "Believer", believing that the goal is true, tries to defend his stance. To get a meaningful semantics out of such games, we look at the *winning strategies* of the players, and depending on their existence, we assign an actual truth value to the given goal. This game-theoretic approach to semantics is influenced by Lorenzen's dialogue games for logic (see [Lor61]). A game semantics may have a denotational or an operational flavor, or lie somewhere in-between the two. In [DCLN98], for example, they stay close to the procedural side of semantics, dealing directly with first-order programs, while the game semantics of [GRW08] and [Tso13], which are the ones that we use here, are more of a denotational nature.

LP. Given an LP program $\mathcal{P}$, we say that the goal $\mathtt{p}$ *succeeds*, if Believer has a winning strategy in the game $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow \mathtt{p})$ (see [DCLN98] and [Tso13]).

DLP. To account for disjunctions in the game world, we only need to add a couple of rules to the original game. The definition of the semantics, stays the same: *a goal succeeds if Believer has a winning strategy* in the DLP game (see [Tso13]).

LPN. Again, starting from the LP game, we only add one rule to it and reach the LPN game (see [GRW08]) which we use to obtain a semantics for LPN. This time, the change of rules implies that there might be *ties* between the two players, and there might be the case that both players have a strategy which can guarantee *at least a tie*. With these changes we manage to capture the extra truth values of the well-founded models (either the infinite-valued or the three-valued one).

DLPN. At this point we do not know of a game for DLPN that we can use to obtain the required semantics. An obvious idea is to consider the LP game together with the extra rules of the DLP game and the ones of the LPN game, but it has proven difficult to prove its correctness. However, as an application of the abstract semantic framework, we will define an operator that acts on semantics of non-disjunctive languages, and yields a new semantics, for the corresponding disjunctive ones. Thus, using this operator on the LPN game semantics we will obtain, again "for free", a novel game semantics for DLPN.

Infinite propositional vs. finite first-order programs.

It is well known that for a finite first-order logic program, there corresponds an infinite propositional one, with equivalent denotational semantics.[7] Therefore, once we have accomplished to define a semantics for infinite propositional programs, we can use it for finite first-order programs as well. Mathematically speaking, it is quite cumbersome to deal with function symbols and variables; instead we embrace infinity and restrict ourselves to propositional programs. See [Fit99] for a relevant discussion.

2 The syntax of logic programs

In order to study any language, we need a precise description of its syntax. Following mathematical tradition, in this section we give formal (set-theoretic) definitions of the four logic programming languages mentioned in the introduction.

Foremost we assume a countably infinite set $\mathcal{At}$ whose elements we denote by $a, b, c, \ldots$ and we call *atoms*. We use the binary connectives $\vee$, $\wedge$, and $\rightarrow$, and the unary connective $\sim$, which is meant to stand for *negation-as-failure* to build the well-formed formulæ (wff) of this logic. Atoms and their negations form the set $\mathcal{L}it$ of *literals*. The *language* $\mathcal{L}$ is the set of all wffs.[8]

Definition 2.1 An *L.P. disjunction* is a finite subset $D \subseteq \mathcal{L}it$. An *L.P. conjunction* is a finite sequence $\mathscr{D}$ of L.P. disjunctions. For obvious reasons we omit the "L.P." prefix whenever no confusion arises. A *clause* is a pair $(H, \mathscr{D})$, in which the *head* $H = \{a_1, \ldots, a_n\}$ is an L.P. disjunction, and the *body* $\mathscr{D} = \langle D_1, \ldots, D_m \rangle$ is

[7] This infinite program is essentially obtained by collecting into a set all ground instantiations of every rule that appears in the original first-order program.

[8] Note that we have not specified what the atoms in $\mathcal{At}$ really are. One may consider them to simply be propositional variables without any further structure, just like in propositional calculus. In this case, we have a *propositional program*. Another possibility is to let them be the atomic formulæ of a first-order language, built by its predicates, function symbols, variables, and constants. We then call it a *first-order program*.

an L.P. conjunction. If the head of a clause is non-empty we call it a *rule*, while if it is empty and $m = 1$, a *goal*.[9] A *fact* is a bodiless clause. In logic programs, rules will be written as

$$\underbrace{\mathtt{a}_1 \vee \cdots \vee \mathtt{a}_n}_{\text{head}} \leftarrow \underbrace{\ell_1^1 \vee \cdots \vee \ell_{s_1}^1 \,, \cdots, \ell_1^m \vee \cdots \vee \ell_{s_m}^m}_{\text{body}} .$$

Such a rule is called *disjunctive* (also *proper*) if $n > 1$; it is *clean*, if $s_j = 1$ for all $1 \leq j \leq m$. Therefore, a clean rule looks like this:

$$\mathtt{a}_1 \vee \cdots \vee \mathtt{a}_n \leftarrow \ell^1 \,, \cdots, \ell^m .$$

A *clean program* is a countable set of clean rules; it is *disjunctive* (also *proper*), if at least one of its rules is.[10]

Example 2.2 The set $\{a, b, c\}$ is understood to stand for the disjunction $a \vee b \vee c$, the sequence $\langle \{a\}, \{b, c\} \rangle$ for the conjunction $a \wedge (b \vee c)$, and the pair $(\{p, q\}, \langle \{a, b\}, \{b, c\} \rangle)$ for the implication $((a \vee b) \wedge (b \vee c)) \rightarrow p \vee q$.

Since a program is itself a set of rules, programs can also be translated in the same manner:

Example 2.3 Consider the program

$$\left\{ \begin{array}{l} \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}\,, \mathtt{b} \vee \mathtt{t} \\ \mathtt{r} \leftarrow \sim \mathtt{a}\,, \mathtt{t} \\ \mathtt{t} \leftarrow \end{array} \right\}.$$

Translating it into set-theoretic terms, we end up with the following set of pairs:

$$\{(\{p, q\}, \langle \{a\}, \{b, t\} \rangle), (\{r\}, \langle \{\sim a\}, \{t\} \rangle), (\{t\}, \langle\,\rangle)\} .$$

Definition 2.4 A logic programming language L is determined by:

- $\mathbf{H}_L$, the set of heads of L-rules;
- $\mathbf{B}_L$, the set of bodies of L-rules;
- $\mathbf{Q}_L$, the set of L-queries (goal clauses).

We define the set of *L-rules* as $\mathbf{R}_L \stackrel{\text{def}}{=} \mathbf{H}_L \times \mathbf{B}_L$. An *$L$-program* is a set of L-rules. We write $\mathbf{P}_L$ for the set of all L-programs and we drop all those "L-" prefixes when L is clear by the context. In most logic programming languages, the bodies of rules are required to be conjunctions, in which case we denote by $\mathbf{C}_L$ the set of all possible conjuncts out of which bodies are formed; in this case we have

$$\mathbf{B}_L \stackrel{\text{def}}{=} \mathbf{C}_L^\star$$

where $\mathbf{C}_L^\star$ is the set of all finite sequences of members of $\mathbf{C}_L$.

Four logic programming languages.

To formally define the languages we are interested in, we need to specify for each one of them its determining sets: its heads, its body-conjuncts, and its queries. Here they are:[11]

$$\begin{array}{llll} \mathbf{H}_{\mathrm{LP}} \stackrel{\text{def}}{=} \wp_1(\mathcal{A}t) & \mathbf{H}_{\mathrm{LPN}} \stackrel{\text{def}}{=} \wp_1(\mathcal{A}t) & \mathbf{H}_{\mathrm{DLP}} \stackrel{\text{def}}{=} \wp_{\mathrm{f}}(\mathcal{A}t) & \mathbf{H}_{\mathrm{DLPN}} \stackrel{\text{def}}{=} \wp_{\mathrm{f}}(\mathcal{A}t) \\ \mathbf{C}_{\mathrm{LP}} \stackrel{\text{def}}{=} \wp_1(\mathcal{A}t) & \mathbf{C}_{\mathrm{LPN}} \stackrel{\text{def}}{=} \wp_1(\mathcal{L}it) & \mathbf{C}_{\mathrm{DLP}} \stackrel{\text{def}}{=} \wp_{\mathrm{f}}(\mathcal{A}t) & \mathbf{C}_{\mathrm{DLPN}} \stackrel{\text{def}}{=} \wp_{\mathrm{f}}(\mathcal{L}it) \\ \mathbf{Q}_{\mathrm{LP}} \stackrel{\text{def}}{=} \wp_1(\mathcal{A}t) & \mathbf{Q}_{\mathrm{LPN}} \stackrel{\text{def}}{=} \wp_1(\mathcal{L}it) & \mathbf{Q}_{\mathrm{DLP}} \stackrel{\text{def}}{=} \wp_{\mathrm{f}}(\mathcal{A}t) & \mathbf{Q}_{\mathrm{DLPN}} \stackrel{\text{def}}{=} \wp_{\mathrm{f}}(\mathcal{L}it). \end{array}$$

Notice that for all of the languages above, the sets $\mathbf{C}_L$ and $\mathbf{Q}_L$ coincide.

---

[9] We have imposed the restriction $m = 1$ for goals. This will simplify the development without any significant loss: to deal with a goal like $\leftarrow \mathtt{D}_1 \,, \cdots, \mathtt{D}_m$, one can simply add the rule $\mathtt{w} \leftarrow \mathtt{D}_1 \,, \cdots, \mathtt{D}_m$ to the program, where $\mathtt{w}$ is a suitable fresh atom, and query $\mathtt{w}$ instead.

[10] For reasons of simplicity, when dealing with semantics, *we frequently assume that programs are clean.* This does not really impose any substantial restriction: for an unclean program $\mathcal{D}$, we simply use the semantics of its equivalent clean version $\widehat{\mathcal{D}}$ following the "Lloyd–Topor transformation" (see [LT84] or [Tso13]).

[11] Given a set X, its powerset is $\wp(X)$, and we subscript it with n to refer to the set of all subsets of X of cardinality n: e.g., $\wp_1(X)$ is the set of all singleton-subsets of X. For the set of all finite subsets of X we use $\wp_{\mathrm{f}}(X)$, so that $\wp_{\mathrm{f}}(X) = \bigcup_{i \in \omega} \wp_i(X)$.

Example 2.5 Here are some sample programs written in these languages:

$$\mathcal{P}_1 = \underbrace{\left\{\begin{array}{l}\mathtt{p} \leftarrow \mathtt{a}\\ \mathtt{p} \leftarrow \mathtt{b}\\ \mathtt{b} \leftarrow\end{array}\right\}}_{\in \mathbf{P}_{\mathrm{LP}}} \qquad \mathcal{P}_2 = \underbrace{\left\{\begin{array}{l}\mathtt{p} \leftarrow\\ \mathtt{r} \leftarrow \sim\mathtt{p}\\ \mathtt{s} \leftarrow \sim\mathtt{q}\end{array}\right\}}_{\in \mathbf{P}_{\mathrm{LPN}}} \qquad \mathcal{P}_3 = \underbrace{\left\{\begin{array}{l}\mathtt{a} \vee \mathtt{b} \leftarrow\\ \quad \mathtt{p} \leftarrow \mathtt{a}\\ \quad \mathtt{p} \leftarrow \mathtt{b}\end{array}\right\}}_{\in \mathbf{P}_{\mathrm{DLP}}} \qquad \mathcal{P}_4 = \underbrace{\left\{\begin{array}{l}\mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow\\ \qquad \mathtt{p} \leftarrow \sim\mathtt{q}\\ \qquad \mathtt{q} \leftarrow \sim\mathtt{r}\\ \qquad \mathtt{r} \leftarrow \sim\mathtt{p}\end{array}\right\}}_{\in \mathbf{P}_{\mathrm{DLPN}}}.$$

3 Truth value spaces

The standard semantics of propositional logic is provided by Boolean logic, by mapping each binary connective above to the corresponding boolean operation on $\mathbb{B} = \{\mathbf{F}, \mathbf{T}\}$. As it turns out, we will need more truth values to handle negation, and therefore we will not tie ourselves to the booleans. Abstracting away the properties that we need, we reach the following:

Definition 3.1 A *truth value space* is a completely distributive Heyting algebra A with an additional unary operator $\sim$ in which the following law holds: [12]

$$a \Rightarrow \bigvee_{s \in S} s = \bigvee_{s \in S} (a \Rightarrow s), \quad \text{for any } S \subseteq A.$$

We impose no condition on $\sim$. Naturally we call members of a truth value space *truth values* and we use the term $\mathcal{V}$*-interpretation* for any Herbrand interpretation whose values lie in $\mathcal{V}$.

Note that a truth value space's structure allows us to interpret all symbols of our programming languages. The canonical example of a truth value space is $\mathbb{B}$, in which $\sim$ is defined as the classical negation that flips the two values. As mentioned above, this space turns out to be too poor for languages that actually use negation as failure, and so we now investigate spaces with more values.

The spaces $\mathbb{V}_\kappa$

Even though three-valued logics have been used for many years in the study of negation in logic programming (e.g., [VGRS91], [Fit85], and [Kun87]) we jump directly to a family of infinite-valued logics on which we will eventually base our semantics of negation-as-failure. We are actually dealing with refinements of the usual three-valued logic that was originally used for the well-founded semantics, enjoying some additional convenient properties. Spaces of this kind were first introduced in [RW05], and further studied in [GRW08] and [Lüd11].

Definition 3.2 Let $\kappa \geq \omega$ be an ordinal number. The structured set [13]

$$\mathbb{V}_\kappa = (\mathbb{V}_\kappa; \vee, \wedge, \Rightarrow, \sim)$$

consists of an infinite number of distinct elements, which we separate into three disjoint sets:

$$\mathcal{F}_\kappa \stackrel{\text{def}}{=} \{\mathbf{F}_\alpha \mid \alpha < \kappa\}; \qquad \mathcal{U} \stackrel{\text{def}}{=} \{\mathbf{U}\}; \qquad \mathcal{T}_\kappa \stackrel{\text{def}}{=} \{\mathbf{T}_\alpha \mid \alpha < \kappa\}.$$

We denote their union by $\mathbb{V}_\kappa \stackrel{\text{def}}{=} \mathcal{F}_\kappa \cup \mathcal{U} \cup \mathcal{T}_\kappa$, and equip it with the total order

$$\mathbf{F}_0 < \mathbf{F}_1 < \cdots < \mathbf{F}_\alpha < \cdots < \mathbf{U} < \cdots < \mathbf{T}_\alpha < \cdots < \mathbf{T}_1 < \mathbf{T}_0.$$

This turns $\mathbb{V}_\kappa$ into a complete bounded lattice, thus determining $\vee$, $\wedge$, and $\Rightarrow$:

$$x \vee y = \max\{x, y\}, \quad x \wedge y = \min\{x, y\}, \quad \text{and} \quad x \Rightarrow y = \begin{cases}\mathbf{T}_0 & \text{if } x \leq y,\\ y & \text{otherwise.}\end{cases}$$

[12] Consult [DP02] for more information about lattices, Heyting algebras, and related tools and notation.

[13] Here we follow the usual practice of abusing the notation by identifying the structured set with its carrier.

But for $\mathbb{V}_\kappa$ to be a valid candidate for a truth value space, it remains to define the operator $\sim$:

$$\sim x \stackrel{\text{def}}{=} \begin{cases} \mathbf{T}_{\alpha+1} & \text{if } x = \mathbf{F}_\alpha, \\ \mathbf{F}_{\alpha+1} & \text{if } x = \mathbf{T}_\alpha, \\ \mathbf{U} & \text{if } x = \mathbf{U}. \end{cases}$$

Unless explicitly mentioned, we will simply write $\mathbb{V}$ instead of $\mathbb{V}_\omega$ in case $\kappa = \omega$.

The intuition behind these truth values is easy to explain: we identify $\mathbf{F}_0$ and $\mathbf{T}_0$ with the usual boolean values $\mathbf{F}$ and $\mathbf{T}$, i.e., absolute truth and absolute falsity. The ordinal in the subscript corresponds to a level of doubt that we have, so that $\mathbf{F}_1$ represents a "false" value but with a little doubt, $\mathbf{F}_2$ one with a little more, etc., and similarly for the "true" values. In the middle lies $\mathbf{U}$, which we use in the case that we only have doubts without any bias towards truth or falsity: it is entirely uncertain.

Theorem 3.3 *For any $\kappa \geq \omega$, $\mathbb{V}_\kappa$ is a truth value space.*

Proof sketch. As $\mathbb{V}_\kappa$ has a unary operation $\sim$, we only need to verify that it is a completely distributive Heyting algebra and that the extra distributivity law is satisfied. We first verify that it is distributive, complete, a Heyting algebra, isomorphic to its dual $\mathbb{V}_\kappa^\partial$, and algebraic. These are sufficient conditions for it to be *completely distributive* (see Theorem 10.29 of [DP02]) so all it remains to check is the extra distributivity law, which is trivial for any chain and all $\mathbb{V}_\kappa$ are chains, so the result follows.

4 An abstract framework for semantics

We define in this section a formal framework of semantics and examine some semantics of the four languages we have met with respect to this framework.

Definition 4.1 Let L be a logic programming language, let $\mathcal{M}$ be a set whose elements we will call *meanings*, and let $\mathcal{V}$ be a truth value space. Then:

$$\text{an } \mathcal{M}\text{-semantics for } L \text{ is a function} \qquad \mathbf{m} : \mathbf{P}_L \to \mathcal{M}; \tag{4.1}$$

$$\text{a } \mathcal{V}\text{-answer function for } \mathcal{M} \text{ is a function} \qquad \mathbf{a} : \mathcal{M} \to \mathbf{Q}_L \to \mathcal{V}; \tag{4.2}$$

$$\text{and a } \mathcal{V}\text{-system for } L \text{ is a function} \qquad \mathbf{s} : \mathbf{P}_L \to \mathbf{Q}_L \to \mathcal{V}. \tag{4.3}$$

A pair $(\mathbf{m}, \mathbf{a})$ is simply called a *semantics for L*.

Remark 4.2 Composing a $\mathcal{V}$-answer function for $\mathcal{M}$ with an $\mathcal{M}$-semantics for L we obtain a $\mathcal{V}$-system for L. Therefore, a semantics $(\mathbf{m}, \mathbf{a})$ naturally gives rise to the $\mathcal{V}$-system $\mathbf{a} \circ \mathbf{m}$. In this way, we will be able to use $(\mathbf{m}, \mathbf{a})$ in a context where a $\mathcal{V}$-system is expected.

Definition 4.3 Let L be a logic programming language. We call two semantics of L $(\mathbf{m}_1, \mathbf{a}_1)$ and $(\mathbf{m}_2, \mathbf{a}_2)$ *equivalent* iff the corresponding $\mathcal{V}$-systems are equal. In symbols,

$$(\mathbf{m}_1, \mathbf{a}_1) \approx (\mathbf{m}_2, \mathbf{a}_2) \iff \mathbf{a}_1 \circ \mathbf{m}_1 = \mathbf{a}_2 \circ \mathbf{m}_2.$$

Notice that $\approx$ is an equivalence relation. When the context clearly hints the intended $\mathcal{V}$-answer functions under consideration, we might abuse the notation and simply write $\mathbf{m}_1 \approx \mathbf{m}_2$ instead.

Definition 4.4 Let L be a logic programming language. We say that $(\mathbf{m}_1, \mathbf{a}_1)$ *refines* $(\mathbf{m}_2, \mathbf{a}_2)$ with respect to the operator $\mathcal{C}$ iff:

$$(\mathbf{m}_1, \mathbf{a}_1) \lhd_{\mathcal{C}} (\mathbf{m}_2, \mathbf{a}_2) \iff \begin{cases} \mathbf{m}_i : \mathbf{P}_L \to \mathcal{M}_i \\ \mathbf{a}_i : \mathcal{M}_i \to \mathbf{Q}_L \to \mathcal{V} \\ \mathcal{C} : \mathcal{M}_1 \to \mathcal{M}_2 \\ \mathbf{m}_2 = \mathcal{C} \circ \mathbf{m}_1 \\ \mathbf{a}_1 = \mathbf{a}_2 \circ \mathcal{C}. \end{cases}$$

Lemma 4.5 *The following implication holds:*

$$(\mathbf{m}_1, \mathbf{a}_1) \lhd_{\mathcal{C}} (\mathbf{m}_2, \mathbf{a}_2) \implies (\mathbf{m}_1, \mathbf{a}_1) \approx (\mathbf{m}_2, \mathbf{a}_2). \tag{4.4}$$

Proof We have $\mathbf{a}_1 \circ \mathbf{m}_1 = (\mathbf{a}_2 \circ \mathcal{C}) \circ \mathbf{m}_1 = \mathbf{a}_2 \circ (\mathcal{C} \circ \mathbf{m}_1) = \mathbf{a}_2 \circ \mathbf{m}_2$, which by the definition of $\approx$ is equivalent to $(\mathbf{m}_1, \mathbf{a}_1) \approx (\mathbf{m}_2, \mathbf{a}_2)$. □

Example 4.6 (The least Herbrand model semantics LHM.) We take $\mathcal{V}_{\mathrm{LHM}} \stackrel{\mathrm{def}}{=} \mathbb{B}$; $\mathcal{M}_{\mathrm{LHM}}$ is the set of all possible Herbrand interpretations; $\mathbf{m}_{\mathrm{LHM}}$ maps an LP program to its least Herbrand model; and

$$\mathbf{a}_{\mathrm{LHM}}(M)(p) \stackrel{\mathrm{def}}{=} \begin{cases} \mathbf{T}, & \text{if } p \in M \\ \mathbf{F}, & \text{otherwise.} \end{cases}$$

Example 4.7 (The game semantics LPG.) Again $\mathcal{V}_{\mathrm{LPG}} \stackrel{\mathrm{def}}{=} \mathbb{B}$; now $\mathcal{M}_{\mathrm{LPG}}$ is the set of strategies based on LP programs; $\mathbf{m}_{\mathrm{LPG}}$ maps every LP program $\mathcal{P}$ to the set of strategies for the LPG game based on $\mathcal{P}$; and finally

$$\mathbf{a}_{\mathrm{LPG}}(\Sigma)(q) \stackrel{\mathrm{def}}{=} \begin{cases} \mathbf{T}, & \text{if there is a winning strategy } \sigma \in \Sigma \text{ for } q \\ \mathbf{F}, & \text{otherwise.} \end{cases}$$

Similarly we define the DLPG semantics.

Example 4.8 (The minimal model semantics MM.) $\mathcal{V}_{\mathrm{MM}} \stackrel{\mathrm{def}}{=} \mathbb{B}$; $\mathcal{M}_{\mathrm{MM}}$ consists of all *sets* of Herbrand interpretations; $\mathbf{m}_{\mathrm{MM}}$ maps a DLP program to *the set* of its minimal models; and we take

$$\mathbf{a}_{\mathrm{MM}}(\mathscr{M})(Q) \stackrel{\mathrm{def}}{=} \begin{cases} \mathbf{T}, & \text{if } Q \text{ is true in every model } M \in \mathscr{M} \\ \mathbf{F}, & \text{otherwise.} \end{cases}$$

Example 4.9 (The infinite-valued well-founded semantics WF^κ.) Here we need $\mathcal{V}_{\mathrm{WF}^\kappa} \stackrel{\mathrm{def}}{=} \mathbb{V}_\kappa$; $\mathcal{M}_{\mathrm{WF}^\kappa}$ consists of all possible Herbrand $\mathbb{V}_\kappa$-interpretations of LPN programs; $\mathbf{m}_{\mathrm{WF}^\kappa}$ maps every LPN program to its $\mathbb{V}_\kappa$-valued, well-founded model; and $\mathbf{a}_{\mathrm{WF}^\kappa}(M)(p) \stackrel{\mathrm{def}}{=} M(p)$. Notice that the three-valued well-founded semantics WF is equal to WF^κ for $\kappa = 1$.

Remark 4.10 The ordinal κ that we use in the truth value spaces $\mathbb{V}_\kappa$ may vary depending on our needs. The reader should note at this point that if the programs are finite, an ordinal as small as ω suffices to give us satisfying semantics, in the sense that collapsing the obtained $\mathbb{V}_\omega$-valued model to a three-valued one will always yield the desired well-founded model. See [RW05] and [Lüd11] for more information.

Example 4.11 (The game semantics LPNG.) We set $\mathcal{V}_{\mathrm{LPNG}} \stackrel{\mathrm{def}}{=} \mathbb{V}_1$; again $\mathcal{M}_{\mathrm{LPNG}}$ is the set of strategies based on finite LPN programs; and $\mathbf{m}_{\mathrm{LPNG}}$ maps every finite LPN program $\mathcal{P}$ to the set of strategies for the LPN game on $\mathcal{P}$. Finally,

$$\mathbf{a}_{\mathrm{LPNG}}(\Sigma)(q) \stackrel{\mathrm{def}}{=} \begin{cases} \mathbf{T}, & \text{if there is a winning strategy in } \Sigma \text{ for } q \\ \mathbf{U}, & \text{else, if there is a non-losing strategy in } \Sigma \text{ for } q \\ \mathbf{F}, & \text{otherwise.} \end{cases}$$

Example 4.12 (The infinite-valued LPN game semantics LPNG^ω.) $\mathcal{V}_{\mathrm{LPNG}^\omega} \stackrel{\mathrm{def}}{=} \mathbb{V}_\omega$; $\mathcal{M}_{\mathrm{LPNG}^\omega}$ is the set of strategies based on LPN programs; $\mathbf{m}_{\mathrm{LPNG}^\omega}$ maps every LPN program $\mathcal{P}$ to the set of strategies for the LPNG^ω game on $\mathcal{P}$; and

$$\mathbf{a}_{\mathrm{LPNG}^\omega}(\Sigma)(q) \stackrel{\mathrm{def}}{=} \bigvee \left\{ \bigwedge \{\Phi_\omega(\pi) \mid \pi \in \sigma\} \;\middle|\; \sigma \text{ is a strategy in } \Sigma \text{ for } q \right\},$$

where $\Phi_\omega(-)$ is the payoff function defined in [GRW08].

Example 4.13 (The infinite-valued minimal model semantics MM^ω.) $\mathcal{V}_{\mathrm{MM}^\omega} \stackrel{\mathrm{def}}{=} \mathbb{V}_\omega$; $\mathcal{M}_{\mathrm{MM}^\omega}$ consists of all possible $\mathbb{V}_\omega$-valued Herbrand interpretations of finite DLPN programs; $\mathbf{m}_{\mathrm{MM}^\omega}$ maps every finite DLPN program to the set of its minimal, infinite-valued models; and finally set

$$\mathbf{a}_{\mathrm{MM}^\omega}(\mathscr{M})(q) \stackrel{\mathrm{def}}{=} \bigwedge \{M(q) \mid M \in \mathscr{M}\}.$$

For the time being, we have no game semantics for even finite DLPN programs, and no semantics of any kind for infinite DLPN programs. We are about to ameliorate this situation in the following sections in which we will obtain a model-theoretic semantics for infinite DLPN programs, as well as a couple of game semantics for finite ones.

5 The disjunctifier operator

5.1 Definite instantiations and D-sections

First we need to define what is a definite instantiations of a disjunctive logic program $\mathcal{D}$. Informally, this is what we get by replacing each head of $\mathcal{D}$ by one of its elements. Formally, we define:

Definition 5.1 Let $\phi = (H, \mathcal{B})$ be a disjunctive rule. If $h \in H$, then the definite rule $(h, \mathcal{B})$ is a *definite instantiation of* ϕ. $\mathrm{D}(\phi)$ is the set of all definite instantiations of ϕ.

Example 5.2 Here are some disjunctive rules and their respective definite instantiations:

$$\phi_1 = \mathtt{a} \vee \mathtt{b} \leftarrow \mathtt{p}, {\sim}\mathtt{q} \qquad \phi_2 = \mathtt{e} \vee \mathtt{f} \vee \mathtt{g} \leftarrow \qquad \phi_3 = \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}, \mathtt{b} \vee \mathtt{c}$$

$$\mathrm{D}(\phi_1) = \left\{ \begin{array}{l} \mathtt{a} \leftarrow \mathtt{p}, {\sim}\mathtt{q} \\ \mathtt{b} \leftarrow \mathtt{p}, {\sim}\mathtt{q} \end{array} \right\}, \qquad \mathrm{D}(\phi_2) = \left\{ \begin{array}{l} \mathtt{e} \leftarrow \\ \mathtt{f} \leftarrow \\ \mathtt{g} \leftarrow \end{array} \right\}, \qquad \mathrm{D}(\phi_3) = \left\{ \begin{array}{l} \mathtt{p} \leftarrow \mathtt{a}, \mathtt{b} \vee \mathtt{c} \\ \mathtt{q} \leftarrow \mathtt{a}, \mathtt{b} \vee \mathtt{c} \end{array} \right\}.$$

Definition 5.3 Let $\mathcal{D} = \{(H_i, \mathcal{B}_i)\}_{i \in I}$ be a disjunctive program, indexed by some set of indices I. A $\mathcal{D}$-*section* is any choice function $f \in \prod_{i \in I} H_i$. We write $\mathrm{S}(\mathcal{D})$ for the set of all $\mathcal{D}$-sections. If f is a $\mathcal{D}$-section, we define the *definite instantiation of* $\mathcal{D}$ *under* f to be the definite program

$$\mathcal{D}_f \stackrel{\text{def}}{=} \{(\{f(i)\}, B_i)\}_{i \in I}.$$

We call $\mathcal{P}$ a *definite instantiation* of $\mathcal{D}$, if there is a $\mathcal{D}$-section f such that $\mathcal{P} = \mathcal{D}_f$. Finally, we write $\mathrm{D}(\mathcal{D})$ for the set of all definite instantiations of $\mathcal{D}$.

Example 5.4 Consider the disjunctive program

$$\mathcal{D} = \left\{ \begin{array}{l} \mathtt{s} \vee \textcircled{\mathtt{t}} \leftarrow \mathtt{p}, \mathtt{b} \vee \mathtt{c} \\ \textcircled{\mathtt{a}} \vee \mathtt{b} \leftarrow \\ \quad \textcircled{\mathtt{p}} \leftarrow \mathtt{a} \\ \quad \textcircled{\mathtt{p}} \leftarrow \mathtt{b} \\ \textcircled{\mathtt{b}} \vee \mathtt{c} \leftarrow \end{array} \right\}.$$

There are 8 $\mathcal{D}$-sections in total, and 8 definite instantiations of $\mathcal{D}$. Let $f, g \in \mathrm{S}(\mathcal{D})$ be the following two:

$$f = \{(1,t),(2,a),(3,p),(4,p),(5,b)\} \qquad g = \{(1,s),(2,b),(3,p),(4,p),(5,b)\}.$$

From these two $\mathcal{D}$-sections we obtain two elements of the $\mathrm{D}(\mathcal{D})$ set:

$$\mathcal{D}_f = \left\{ \begin{array}{l} \mathtt{t} \leftarrow \mathtt{p}, \mathtt{b} \vee \mathtt{c} \\ \mathtt{a} \leftarrow \\ \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{b} \leftarrow \end{array} \right\} \qquad \mathcal{D}_g = \left\{ \begin{array}{l} \mathtt{s} \leftarrow \mathtt{p}, \mathtt{b} \vee \mathtt{c} \\ \mathtt{b} \leftarrow \\ \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{b} \leftarrow \end{array} \right\}.$$

Notice that f and $\mathcal{D}_f$ correspond to the choices that appear circled on the program $\mathcal{D}$ above.

5.2 Definitions and theory

Definition 5.5 The operator $(-)^{\vee}$ is an overloaded operator that can be applied to:

(1) $\mathcal{M}$-meanings of LP[N] programs (that is, LP [or LPN] programs):

$$\begin{array}{ll} \text{if} & \mathbf{m} \ : \ \mathbf{P}_{\text{LP[N]}} \to \mathcal{M}, \\ \text{then} & (\mathbf{m})^{\vee} \ : \ \mathbf{P}_{\text{DLP[N]}} \to \wp(\mathcal{M}), \\ \text{is defined by} & (\mathbf{m})^{\vee}(\mathcal{D}) \stackrel{\text{def}}{=} \mathbf{m}(\text{D}(\mathcal{D})). \end{array}$$

(2) $\mathcal{V}$-answers of LP[N] programs:

$$\begin{array}{ll} \text{if} & \mathbf{a} \ : \ \mathcal{M} \to \mathbf{Q}_{\text{LP[N]}} \to \mathcal{V}, \\ \text{then} & (\mathbf{a})^{\vee} \ : \ \wp(\mathcal{M}) \to \mathbf{Q}_{\text{DLP[N]}} \to \mathcal{V}, \\ \text{is defined by} & (\mathbf{a})^{\vee}(\mathcal{S})(Q) \stackrel{\text{def}}{=} \bigwedge_{S \in \mathcal{S}} \bigvee_{q \in Q} \mathbf{a}(S)(q). \end{array}$$

(3) $\mathcal{V}$-systems of LP[N] programs:

$$\begin{array}{ll} \text{if} & \mathbf{s} \ : \ \mathbf{P}_{\text{LP[N]}} \to \mathbf{Q}_{\text{LP[N]}} \to \mathcal{V}, \\ \text{then} & (\mathbf{s})^{\vee} \ : \ \mathbf{P}_{\text{DLP[N]}} \to \mathbf{Q}_{\text{DLP[N]}} \to \mathcal{V}, \\ \text{is defined by} & (\mathbf{s})^{\vee}(\mathcal{D})(Q) \stackrel{\text{def}}{=} \bigwedge_{\mathcal{P} \in \text{D}(\mathcal{D})} \bigvee_{q \in Q} \mathbf{s}(\mathcal{P})(q). \end{array}$$

The following theorem justifies the definitions above, and is the driving idea behind them.

Theorem 5.6 *Let $\mathcal{V}$ be a truth value space, $\mathcal{D}$ a DLP-program, G a DLP-goal, and I a $\mathcal{V}$-interpretation for $\mathcal{D}$. Then*

$$I\Big(\bigwedge \mathcal{D} \to \bigvee G\Big) = \bigwedge_{\mathcal{P} \in \text{D}(\mathcal{D})} \bigvee_{g \in G} I\Big(\bigwedge \mathcal{P} \to g\Big).$$

Proof Since I is a $\mathcal{V}$-interpretation, it respects the structure of $\mathcal{V}$. Proof is a long computation based on the properties of $\mathcal{V}$ as a truth value space, and on the definitions of $\text{S}(\mathcal{D})$, $\text{D}(\mathcal{D})$, and $\mathcal{D}_f$. Here we go. Pick a set of indices J to index $\mathcal{D}$, and denote its rules by R_j, each having a head H_j and a body $\mathcal{B}_j$, so that we have $\mathcal{D} \coloneqq \{R_j \mid j \in J\} = \{H_j \leftarrow \mathcal{B}_j \mid j \in J\}$. Now compute:

$$\begin{array}{lll}
I\Big(\bigwedge \mathcal{D} \to \bigvee G\Big) & & \\
= I\Big(\bigwedge_{j \in J} R_j \to \bigvee_{g \in G} g\Big) & (1) & = \bigvee_{f \in \text{S}(\mathcal{D})} \bigwedge_{j \in J} \Big(I(f(j)) \Leftarrow I(\mathcal{B}_j)\Big) \Rightarrow \bigvee_{g \in G} I(g) \quad (5) \\
= I\Big(\bigwedge_{j \in J} R_j\Big) \Rightarrow I\Big(\bigvee_{g \in G} g\Big) & (*) & = \bigwedge_{f \in \text{S}(\mathcal{D})} \Big[\bigwedge_{j \in J} \Big(I(f(j)) \Leftarrow I(\mathcal{B}_j)\Big) \Rightarrow \bigvee_{g \in G} I(g)\Big] \quad (6) \\
= \bigwedge_{j \in J} I(R_j) \Rightarrow \bigvee_{g \in G} I(g) & (*) & = \bigwedge_{f \in \text{S}(\mathcal{D})} \Big[\bigwedge_{j \in J} I(f(j) \leftarrow \mathcal{B}_j) \Rightarrow \bigvee_{g \in G} I(g)\Big] \quad (*) \\
= \bigwedge_{j \in J} I(H_j \leftarrow \mathcal{B}_j) \Rightarrow \bigvee_{g \in G} I(g) & (2) & = \bigwedge_{f \in \text{S}(\mathcal{D})} \Big[I\Big(\bigwedge_{j \in J} (f(j) \leftarrow \mathcal{B}_j)\Big) \Rightarrow \bigvee_{g \in G} I(g)\Big] \quad (*) \\
= \bigwedge_{j \in J} I\Big(\bigvee_{h \in H_j} h \leftarrow \mathcal{B}_j\Big) \Rightarrow \bigvee_{g \in G} I(g) & (3) & = \bigwedge_{f \in \text{S}(\mathcal{D})} \bigvee_{g \in G} \Big[I\Big(\bigwedge_{j \in J} (f(j) \leftarrow \mathcal{B}_j)\Big) \Rightarrow I(g)\Big] \quad (7) \\
= \bigwedge_{j \in J} \Big[I\Big(\bigvee_{h \in H_j} h\Big) \Leftarrow I(\mathcal{B}_j)\Big] \Rightarrow \bigvee_{g \in G} I(g) & (*) & = \bigwedge_{f \in \text{S}(\mathcal{D})} \bigvee_{g \in G} \Big[I\Big(\bigwedge \mathcal{D}_f\Big) \Rightarrow I(g)\Big] \quad (8) \\
= \bigwedge_{j \in J} \Big[\bigvee_{h \in H_j} I(h) \Leftarrow I(\mathcal{B}_j)\Big] \Rightarrow \bigvee_{g \in G} I(g) & (*) & = \bigwedge_{\mathcal{P} \in \text{D}(\mathcal{D})} \bigvee_{g \in G} \Big[I\Big(\bigwedge \mathcal{P}\Big) \Rightarrow I(g)\Big] \quad (9) \\
= \bigwedge_{j \in J} \bigvee_{h \in H_j} [I(h) \Leftarrow I(\mathcal{B}_j)] \Rightarrow \bigvee_{g \in G} I(g) & (4) & = \bigwedge_{\mathcal{P} \in \text{D}(\mathcal{D})} \bigvee_{g \in G} I\Big(\bigwedge \mathcal{P} \to g\Big) \quad (*)
\end{array}$$

where each step of the computation is justified as follows: (1) by assumption for $\mathcal{D}$ and G; (2) by assumption for R_j; (3) by assumption for H_j; (4) property of completely distributive Heyting Algebras (c.d.H.A.) ($\mathcal{V}$ is a truth value space (t.v.s.), and therefore a c.d.H.A.); (5) by the fact that $\mathcal{V}$, as a t.v.s., is c.d., and by the definition of $\text{S}(\mathcal{D})$; (6) property of c.d.H.A.; (7) property of c.d.H.A.; (8) by the definiton of $\mathcal{D}_f$; (9) by the definitions of $\text{S}(\mathcal{D})$, $\text{D}(\mathcal{D})$, and $\mathcal{D}_f$; and all steps marked by (*) follow from the fact that I is a $\mathcal{V}$-interpretation. □

Lemma 5.7 *Let* L *be LP or LPN. Suppose that* $\mathcal{M}$ *is some set of meanings for* L *and* $\mathcal{V}$ *a truth value space. Let* $\mathbf{m}$ *and* $\mathbf{a}$ *be an* $\mathcal{M}$*-semantics and a* $\mathcal{V}$*-answer function for* L *respectively. Then*

$$(\mathbf{a} \circ \mathbf{m})^{\vee} = (\mathbf{a})^{\vee} \circ (\mathbf{m})^{\vee};$$

or, following Remark 4.2, $(\mathbf{m}, \mathbf{a})^{\vee} = ((\mathbf{m})^{\vee}, (\mathbf{a})^{\vee})$*. It follows that if* $(\mathbf{m}_1, \mathbf{a}_1)$ *and* $(\mathbf{m}_2, \mathbf{a}_2)$ *are two semantics for* L*, then*

$$(\mathbf{m}_1, \mathbf{a}_1) \approx (\mathbf{m}_2, \mathbf{a}_2) \implies (\mathbf{m}_1, \mathbf{a}_1)^{\vee} \approx (\mathbf{m}_2, \mathbf{a}_2)^{\vee}. \tag{5.1}$$

Proof We compute:

$$\begin{aligned}
((\mathbf{a})^{\vee} \circ (\mathbf{m})^{\vee})(\mathcal{D})(Q) &= ((\mathbf{a})^{\vee}((\mathbf{m})^{\vee}(\mathcal{D})))(Q) \\
&= \bigwedge_{M \in (\mathbf{m})^{\vee}(\mathcal{D})} \bigvee_{q \in Q} \mathbf{a}(M)(q) && \text{(def. of } (\mathbf{a})^{\vee}) \\
&= \bigwedge_{M \in \mathbf{m}(\mathrm{D}(\mathcal{D}))} \bigvee_{q \in Q} \mathbf{a}(M)(q) && \text{(def. of } (\mathbf{m})^{\vee}) \\
&= \bigwedge_{\mathcal{P} \in \mathrm{D}(\mathcal{D})} \bigvee_{q \in Q} \mathbf{a}(\mathbf{m}(\mathcal{P}))(q) \\
&= \bigwedge_{\mathcal{P} \in \mathrm{D}(\mathcal{D})} \bigvee_{q \in Q} (\mathbf{a} \circ \mathbf{m})(\mathcal{P})(q) \\
&= (\mathbf{a} \circ \mathbf{m})^{\vee}(\mathcal{D})(Q). && \text{(def. of } (\mathbf{s})^{\vee})
\end{aligned}$$

□

Lemma 5.8 *Let* $\mathcal{V}$ *be a totally ordered, truth value space, and let* $\mathcal{D}$ *be a clean DLP (or DLPN) program. If* M *is a model of* $\mathcal{D}$*, then there is an LP (or LPN) program* $\mathcal{P} \in \mathrm{D}(\mathcal{D})$ *such that* M *is a model of* $\mathcal{P}$*. In symbols,*

$$\{M \mid M \text{ is a model of } \mathcal{D}\} \subseteq \{M \mid M \text{ is a model of } \mathcal{P} \text{ for some } \mathcal{P} \in \mathrm{D}(\mathcal{D})\}.$$

Proof Let us index the rules of $\mathcal{D}$ by some index set J, so that we have $\mathcal{D} = \{R_j \mid j \in J\}$ where for each j, $R_j \coloneqq H_j \leftarrow \mathcal{B}_j$. Now let M be a model of $\mathcal{D}$. Therefore, M satisfies every rule R_j of $\mathcal{D}$, i.e.,

$$\text{for every } j \in J, \quad M(H_j) \geq_{\mathcal{V}} M(\mathcal{B}_j).$$

Since H_j is a finite set of atoms, and since $\mathcal{V}$ is totally ordered, we have

$$M(H_j) = \bigvee \{M(h) \mid h \in H_j\} = \max\{M(h) \mid h \in H_j\} = M(h_j),$$

where h_j is an element of H_j for which the above equality holds. Picking for each $j \in J$ such an h_j, we obtain a $\mathcal{D}$-section and correspondingly the definite instantiation $\mathcal{P} = \{h_j \leftarrow \mathcal{B}_j \mid j \in J\} \in \mathrm{D}(\mathcal{D})$. We observe that since $M(h_j) = M(H_j) \geq_{\mathcal{V}} M(\mathcal{B}_j)$, M satisfies every rule of $\mathcal{P}$; in other words, M is a model of $\mathcal{P}$, which is what we wanted to show. □

6 Applications and examples

As promised, we investigate the application of the $(-)^{\vee}$ operator on the semantics of the non-disjunctive languages that interest us and investigate the equivalences of the resulting semantics.

6.1 Applications on model-theoretic semantics

From LP to DLP.

Let us start with the simplest case of LP programs and their least Herbrand model semantics, LHM. We first notice that using $(-)^{\vee}$ on LHM we obtain a semantics for DLP, which we will denote by $\mathrm{LHM}_{\vee}$. We have:

$$\mathcal{V}_{\mathrm{LHM}_{\vee}} = \mathcal{V}_{\mathrm{LHM}} = \mathbb{B} \qquad\qquad \mathcal{M}_{\mathrm{LHM}_{\vee}} = \wp(\mathcal{M}_{\mathrm{LHM}}).$$

We proceed following the definitions:

$$\begin{aligned}
\mathbf{m}_{\mathrm{LHM}_\vee}(\mathcal{P}) &= (\mathbf{m}_{\mathrm{LHM}})^\vee(\mathcal{P}) = \mathbf{m}_{\mathrm{LHM}}(\mathrm{D}(\mathcal{P})),\\
\mathbf{a}_{\mathrm{LHM}_\vee}(\mathcal{S})(Q) &= (\mathbf{a}_{\mathrm{LHM}})^\vee(\mathcal{S})(Q) = \bigwedge_{S\in\mathcal{S}} \bigvee_{q\in Q} \mathbf{a}_{\mathrm{LHM}}(S)(q),\\
\mathbf{s}_{\mathrm{LHM}_\vee}(\mathcal{D})(Q) &= (\mathbf{s}_{\mathrm{LHM}})^\vee(\mathcal{D})(Q) = \bigwedge_{\mathcal{P}\in\mathrm{D}(\mathcal{D})} \bigvee_{q\in Q} \mathbf{s}_{\mathrm{LHM}}(\mathcal{P})(q).
\end{aligned}$$

Theorem 6.1 *The* $\mathrm{LHM}_\vee$ *and the* MM *semantics are equivalent.*

Proof To exhibit the equivalence between the minimal model semantics MM and the obtained semantics $\mathrm{LHM}_\vee$, we appeal to Lemma 4.5: we define a collector operator $\mathcal{C} : \mathcal{M}_{\mathrm{LHM}_\vee} \to \mathcal{M}_{\mathrm{MM}}$ by

$$\mathcal{C}(\mathcal{M}) \stackrel{\text{def}}{=} \{M \in \mathcal{M} \mid M \text{ is } \subseteq\text{-minimal in } \mathcal{M}\},$$

and verify that $(\mathbf{m}_{\mathrm{LHM}_\vee}, \mathbf{a}_{\mathrm{LHM}_\vee}) \lhd_{\mathcal{C}} (\mathbf{m}_{\mathrm{MM}}, \mathbf{a}_{\mathrm{MM}})$. Indeed, according to Definition 4.4, this amounts to two things: (1) $\mathbf{m}_{\mathrm{MM}} = \mathcal{C} \circ \mathbf{m}_{\mathrm{LHM}_\vee}$, and (2) $\mathbf{a}_{\mathrm{LHM}_\vee} = \mathbf{a}_{\mathrm{MM}} \circ \mathcal{C}$. The latter is immediate from the definitions of the three objects involved. For the first one, observe first that $\mathcal{C}$ is monotone. Next, suppose that $\mathcal{D} \in \mathbf{P}_{\mathrm{DLP}}$. Using the monotonicity of $\mathcal{C}$, and Lemma 5.8 (as $\mathbb{B}$ is totally ordered) we compute:

$$\begin{aligned}
\mathbf{m}_{\mathrm{MM}}(\mathcal{D}) &= \mathcal{C}(\{M \mid M \text{ is a model of } \mathcal{D}\}\\
&\subseteq \mathcal{C}(\{M \mid M \text{ is a model of } \mathcal{P} \text{ for some } \mathcal{P} \in \mathrm{D}(\mathcal{D})\})\\
&= \mathcal{C}(\mathbf{m}_{\mathrm{LHM}_\vee}(\mathcal{D})) = (\mathcal{C} \circ \mathbf{m}_{\mathrm{LHM}_\vee})(\mathcal{D}).
\end{aligned}$$

For the other direction, we have $\mathbf{m}_{\mathrm{LHM}_\vee}(\mathcal{D}) \subseteq \{M \mid M \text{ is a model of } \mathcal{D}\}$, on which we apply the monotone $\mathcal{C}$ on both sides to obtain

$$\mathcal{C}(\mathbf{m}_{\mathrm{LHM}_\vee}(\mathcal{D})) \subseteq \mathcal{C}\left(\{M \mid M \text{ is a model of } \mathcal{D}\}\right) = \mathbf{m}_{\mathrm{MM}}(\mathcal{D}).$$

Therefore, since $\mathcal{D}$ was arbitrary, we have $\mathbf{m}_{\mathrm{MM}} = \mathcal{C} \circ \mathbf{m}_{\mathrm{LHM}_\vee}$. □

From LPN to DLPN.

Similarly to the LP case, this time we describe the shift from the WF^κ semantics of LPN and obtain a new semantics for DLPN, which we denote by $\mathrm{WF}^\kappa_\vee$. It has:

$$\mathcal{V}_{\mathrm{WF}^\kappa_\vee} = \mathcal{V}_{\mathrm{WF}^\kappa} = \mathbb{V}_\kappa \qquad\qquad \mathcal{M}_{\mathrm{WF}^\kappa_\vee} = \wp(\mathcal{M}_{\mathrm{WF}^\kappa}).$$

Just like in the case of LHM, we follow the definitions and obtain

$$\begin{aligned}
\mathbf{m}_{\mathrm{WF}^\kappa_\vee}(\mathcal{P}) &= (\mathbf{m}_{\mathrm{WF}^\kappa})^\vee(\mathcal{P}) = \mathbf{m}_{\mathrm{WF}^\kappa}(\mathrm{D}(\mathcal{P})),\\
\mathbf{a}_{\mathrm{WF}^\kappa_\vee}(\mathcal{S})(Q) &= (\mathbf{a}_{\mathrm{WF}^\kappa})^\vee(\mathcal{S})(Q) = \bigwedge_{S\in\mathcal{S}} \bigvee_{q\in Q} \mathbf{a}_{\mathrm{WF}^\kappa}(S)(q),\\
\mathbf{s}_{\mathrm{WF}^\kappa_\vee}(\mathcal{D})(Q) &= (\mathbf{s}_{\mathrm{WF}^\kappa})^\vee(\mathcal{D})(Q) = \bigwedge_{\mathcal{P}\in\mathrm{D}(\mathcal{D})} \bigvee_{q\in Q} \mathbf{s}_{\mathrm{WF}^\kappa}(\mathcal{P})(q).
\end{aligned}$$

Remember that MM^κ is defined only for finite programs, for which ω is a long enough ordinal. Therefore the obtained semantics $\mathrm{WF}^\kappa_\vee$ is in fact more general than MM^κ as it appears in the literature, since $\mathrm{WF}^\kappa_\vee$ gives meaning to any DLPN program, finite or not. Yet, as long as we restrict ourselves to *finite* programs, we have the following theorem:

Theorem 6.2 *The* $\mathrm{WF}^\omega_\vee$ *and the* MM^ω *semantics on finite DLPN programs, are equivalent.*

Proof We define the collector operator $\mathcal{C} : \mathcal{M}_{\mathrm{WF}^\omega_\vee} \to \mathcal{M}_{\mathrm{MM}^\omega}$ by

$$\mathcal{C}(\mathcal{M}) \stackrel{\text{def}}{=} \{M \in \mathcal{M} \mid M \text{ is } \sqsubseteq_\omega\text{-minimal in } \mathcal{M}\},$$

and verify that $(\mathbf{m}_{\mathrm{WF}^\omega_\vee}, \mathbf{a}_{\mathrm{WF}^\omega_\vee}) \lhd_{\mathcal{C}} (\mathbf{m}_{\mathrm{MM}^\omega}, \mathbf{a}_{\mathrm{MM}^\omega})$, so that the result will again be a direct consequence of Lemma 4.5. The remaining of the proof is similar to the one of Theorem 6.1, except that this time we use the fact that $\mathbb{V}_\omega$ is totally ordered. □

6.2 Applications on game semantics

As there has been no formal definition of a game semantics for *infinite* LPN programs, *programs with negation are also assumed to be finite.*

A different game semantics for DLP.
By applying the $(-)^\vee$ operator on the LPG semantics, we can obtain a new game semantics for DLP, which we can prove to be equivalent to the DLPG one. Since we already have a game semantics for DLP programs, we omit the details.

A first game semantics for DLPN.
As we have already mentioned, there appears to be no game semantics for DLPN in the literature. Now we get two such semantics by using the $(-)^\vee$ operator on LPNG and LPNG$^\omega$. According to its definition,

$$\mathcal{V}_{\mathrm{LPNG}_\vee} = \mathcal{V}_{\mathrm{LPNG}} = \mathbb{V}_1, \qquad \mathcal{V}_{\mathrm{LPNG}^\omega_\vee} = \mathcal{V}_{\mathrm{LPNG}^\omega} = \mathbb{V}_\omega,$$
$$\mathcal{M}_{\mathrm{LPNG}_\vee} = \wp(\mathcal{M}_{\mathrm{LPNG}}); \qquad \mathcal{M}_{\mathrm{LPNG}^\omega_\vee} = \wp(\mathcal{M}_{\mathrm{LPNG}^\omega}).$$

Focusing on LPNG$^\omega$, we have

$$\mathbf{m}_{\mathrm{LPNG}^\omega_\vee}(\mathcal{P}) = (\mathbf{m}_{\mathrm{LPNG}^\omega})^\vee(\mathcal{P}) = \mathbf{m}_{\mathrm{LPNG}^\omega}(\mathrm{D}(\mathcal{P})),$$
$$\mathbf{a}_{\mathrm{LPNG}^\omega_\vee}(\mathcal{S})(Q) = (\mathbf{a}_{\mathrm{LPNG}^\omega})^\vee(\mathcal{S})(Q) = \bigwedge\nolimits_{S\in\mathcal{S}} \bigvee\nolimits_{q\in Q} \mathbf{a}_{\mathrm{LPNG}^\omega}(S)(q),$$
$$\mathbf{s}_{\mathrm{LPNG}^\omega_\vee}(\mathcal{D})(Q) = (\mathbf{s}_{\mathrm{LPNG}^\omega})^\vee(\mathcal{D})(Q) = \bigwedge\nolimits_{\mathcal{P}\in\mathrm{D}(\mathcal{D})} \bigvee\nolimits_{q\in Q} \mathbf{s}_{\mathrm{LPNG}^\omega}(\mathcal{P})(q);$$

and similarly for LPNG$_\vee$.

Again, interpreting these in terms of game rules is straightforward: Opponent begins by playing a definite instantiation $\mathcal{P} \in \mathrm{D}(\mathcal{D})$, Player then chooses an element of the goal $q \in Q$, and after this point, the players begin playing the game $\Gamma^{\mathrm{LPN}}_{\mathcal{P}}(\leftarrow q)$ normally, and the outcome of their play in it becomes the outcome of the play on the new game.

Theorem 6.3 *For finite DLPN programs, the game semantics* LPNG$^\omega_\vee$ *and the model-theoretic semantics* MM$^\omega$ *are equivalent.*

Proof Starting from the equivalence

$$\mathbf{s}_{\mathrm{LPNG}^\omega} = \mathbf{s}_{\mathrm{WF}^\omega}, \qquad \text{(main result of [GRW08])}$$

we apply the $(-)^\vee$ operator on both sides, and compute:

$$\begin{aligned}(\mathbf{s}_{\mathrm{LPNG}^\omega})^\vee &= (\mathbf{s}_{\mathrm{WF}^\omega})^\vee && \text{(by Lemma 5.7)}\\ &= \mathbf{s}_{\mathrm{MM}^\omega} && \text{(by Theorem 6.2).}\end{aligned}$$

□

Since we can collapse any infinite-valued space into the three-valued $\mathbb{V}_1$, we have chosen to present only the more general, infinite-valued semantics. But if for any reason we want to restrict ourselves to the three-valued space $\mathbb{V}_1$, we can easily obtain the analogous results.

7 Conclusion

We have defined the notion of a truth value space, and used it in the development of the abstract semantic framework. We saw how various model-theoretic and game-theoretic semantics fit in this framework, becoming concrete mathematical objects of study. We then proceeded to define the $(-)^\vee$ operator, which we applied on them to obtain a first model-theoretic semantics for infinite DLPN programs, as well as a first game-theoretic semantics for finite ones. This process has a pleasant impact on the already-known DLPN semantics, because using this completely different approach, we end up with equivalent semantics; thus raising our condifence in their correctness.

It remains to be investigated how other interesting semantics can be placed within this framework. In particular, proof-theoretic and coalgebraic ones, thus obtaining some new semantics of these kinds for disjunctive languages as further applications of $(-)^\vee$.

References

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2:297–347, 1992.

[AP91] Jean-Marc Andreoli and Remo Pareschi. Logic programming with sequent systems. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming*, volume 475 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 1991.

[Apt90] Krzysztof R. Apt. Logic programming. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 493–574. MIT Press, 1990.

[BM09] Filippo Bonchi and Ugo Montanari. Reactive systems, (semi-)saturated semantics and coalgebras on presheaves. *Theoretical Computer Science*, 410(41):4044–4066, 2009.

[BZ13] Filippo Bonchi and Fabio Zanasi. Saturated semantics for coalgebraic logic programming. In Reiko Heckel and Stefan Milius, editors, *Algebra and Coalgebra in Computer Science*, volume 8089 of *LNCS*, pages 80–94. Springer, 2013.

[Cla78] Keith Clark. Negation as failure. *Logic and Databases*, pages 293–322, 1978.

[CPRW07] Pedro Cabalar, David Pearce, Panos Rondogiannis, and William Wadge. A purely model-theoretic semantics for disjunctive logic programs with negation. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2007.

[DCLN98] Roberto Di Cosmo, Jean-Vincent Loddo, and Stephane Nicolet. A game semantics foundation for logic programming. In Catuscia Palamidessi, Hugh Glaser, and Karl Meinke, editors, *Principles of Declarative Programming*, volume 1490 of *Lecture Notes in Computer Science*, pages 355–373. Springer, 1998. 10.1007/BFb0056626.

[DP02] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order (2. ed.)*. CUP, 2002.

[Fit85] Melvin Fitting. A Kripke–Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[Fit99] Melvin Fitting. Fixpoint semantics for logic programming—a survey. *Theoretical Computer Science*, 278:25–51, 1999.

[Gel08] Michael Gelfond. Answer sets. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7. Elsevier, 2008.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. MIT Press, 1988.

[GRW08] Chrysida Galanaki, Panos Rondogiannis, and William W. Wadge. An infinite-game semantics for well-founded negation in logic programming. *Annals of Pure and Applied Logic*, 151(2-3):70–88, 2008.

[KMP10] Ekaterina Komendantskaya, Guy McCusker, and John Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In Michael Johnson and Dusko Pavlovic, editors, *AMAST*, volume 6486 of *Lecture Notes in Computer Science*, pages 111–127. Springer, 2010.

[KP11] Ekaterina Komendantskaya and John Power. Coalgebraic semantics for derivations in logic programming. In *Proceedings of the 4th intl. conf. on Algebra and coalgebra in Comp. Sc.*, CALCO'11, pages 268–282. Springer, 2011.

[KPS13] Ekaterina Komendantskaya, John Power, and Martin Schmidt. Coalgebraic logic programming: from semantics to implementation. *CoRR*, abs/1312.6568, 2013.

[Kun87] Kenneth Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(4):289–308, 1987.

[Llo87] John Wylie Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1987.

[LMR92] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of disjunctive logic programming*. MIT Press, 1992.

[Lor61] Paul Lorenzen. Ein dialogisches konstruktivitätskriterium. In *Infinitistic Methods: Proceedings of the Symposium on Foundations of Mathematics, Warsaw, 2–9 September 1959*, pages 193–200. Pergamon Press, 1961.

[LT84] J. W. Lloyd and R. W. Topor. Making Prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.

[Lüd11] Rainer Lüdecke. Every formula-based logic program has a least infinite-valued model. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *INAP/WLP*, volume 7773 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2011.

[Min82] Jack Minker. *On indefinite databases and the closed world assumption*, volume 138 of *LNCS*. Springer-Verlag, 1982.

[MN12] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. CUP, June 2012.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[MS06] Dale Miller and Alexis Saurin. A game semantics for proof search: Preliminary results. *ENTCS*, 155:543–563, 2006.

[PR05] David Pym and Eike Ritter. A games semantics for reductive logic and proof-search. In Dan R. Ghica and Guy McCusker, editors, *Games for Logic and Programming Languages (GALOP 2005), University of Edinburgh, 2–3 April 2005*, pages 107–123, 2005.

[RW05] Panos Rondogiannis and William W. Wadge. Minimum model semantics for logic programs with negation-as-failure. *ACM Trans. Comput. Logic*, 6(2):441–467, 2005.

[Tso13] Thanos Tsouanas. A game semantics for disjunctive logic programming. *Annals of Pure and Applied Logic*, 164(11):1144–1175, 2013.

[vEK76] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23:569–574, 1976.

[VGRS91] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):619–649, 1991.