A game semantics for disjunctive logic programming

Thanos Tsouanas[1]

Laboratoire de l'Informatique du Parallélisme,
École Normale Supérieure de Lyon
Université de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA)
46 allée d' Italie,
69364 Lyon cedex 07, France

Abstract

Denotational semantics of logic programming and its extensions (by allowing negation, disjunctions, or both) have been studied thoroughly for many years. In 1998, a game semantics was given to definite logic programs by Di Cosmo, Loddo, and Nicolet, and a few years later it was extended to deal with negation by Rondogiannis and Wadge. Both approaches were proven equivalent to the traditional semantics. In this paper we define a game semantics for disjunctive logic programs and prove soundness and completeness with respect to the minimal model semantics of Minker. The overall development has been influenced by the games studied for PCF and functional programming in general, in the styles of Abramsky–Jagadeesan–Malacaria and Hyland–Ong–Nickau.

Keywords: logic programming, disjunctive logic programming, game semantics, logic programming semantics
2010 MSC: 03B70, 68N17, 68Q55, 91A40

Contents

Email address: `thanos.tsouanas@ens-lyon.fr` (Thanos Tsouanas)
URL: `http://perso.ens-lyon.fr/thanos.tsouanas/` (Thanos Tsouanas)

[1]The research leading to these results has received funding from the *European Community*'s Seventh Framework Programme `FP7/2007-2013` under grant agreement #238381 (the Marie Curie Initial Training Network in Mathematical Logic: MALOA).

1. Introduction

No matter where one meets logic programming, or what the specific features of the underlying language under investigation are, a logic program is always some sort of set of rules of the form

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

read as "*this* holds, if *that* holds", or "I can solve *this*, if I know how to solve *that*". 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:

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

where the commas on the right stand for conjunctions. The standard denotational or declarative semantics for this kind of programs is provided by a specific two-valued model, the so-called *least Herbrand model.* We will briefly review this in §**3**, in an attempt to be self-contained; consult [10] or [20] for further information. One extension is to allow *negations* to appear in bodies of rules:

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

By negation, we mean *negation-as-failure*; the semantics we have in mind here is supplied by the many-valued *well-founded model*, defined in [37]. The extension in which we are interested in this article is the appearance of *disjunctions* in heads:

$$\texttt{a}_1 \lor \cdots \lor \texttt{a}_n \leftarrow \texttt{b}_1 \ , \cdots , \ \texttt{b}_m. \tag{DLP}$$

This enables us to express uncertainty and to derive ambiguous information. Instead of a single least model, in this case, we use a *set of minimal models* for the semantics, as defined in [28]. Disjunctive logic programs are extensively studied in [22]. Finally, one can consider both extensions simultaneously, by allowing *both* negations in bodies *and* disjunctions in heads:[2]

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

A satisfactory, infinite-valued, model-theoretic semantics for this extension was recently defined in [6].

Unfortunately, in the logic programming literature, terminology is not as stable as one could hope. To contribute to this dismay, we introduce the following four abbreviations that will hopefully help the reader:

LP: plain logic programs with neither negation nor disjunctions;

DLP: disjunctive logic programs—our focus;

LPN: logic programs with negation;

DLPN: disjunctive logic programs with negation—a future target.

The model-theoretic approach to semantics outlined above, is not the only one that has proven itself worthy:

Fixpoint semantics. To construct the model-theoretic semantics of these languages, we use the *immediate consequence operator* (traditionally denoted by $T_{\mathcal{P}}$) associated with each program $\mathcal{P}$, and look at its fixpoints. In this development of game semantics, however, we will not need to use this approach. An excellent survey of fixpoint semantics for logic programming is [11].

Procedural 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, and SLI resolution for DLP. In this work, we do not touch this operational side of semantics either; see [5] for the non-disjunctive and [22] for the disjunctive cases.

Before turning to game semantics, one should have a clear understanding of the nature of the aforementioned methods. On one side, we have the denotational, model-theoretic semantics and their fixpoint characterizations. These provide us with a notion of *correctness* for every possible answer to a goal that

[2]What about disjunctions in bodies, or conjunctions in heads? It is an easy exercise to show that this does not affect the programming language in any meaningful way; it only makes the programmer happier. See Remark 2.3 for a solution.

we might give to our program. On the operational side, the procedural semantics provide a construction 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. Games made their debut in the logic programming scene with [9], where we find the first informal description of a game in the logic programming literature. But it was not until [8], that a game semantics was systematically studied for the case of LP. It was there shown, that it is in fact sound and complete with respect to SLD resolution. This is a rather involved game, which stays close to the procedural semantics, and therefore directly handles first-order programs, taking into account variables, function symbols, substitutions, etc. Approximately a decade later, this game—or, to be fair, its propositional version—was extended in [12], to cover negation for finite, propositional LPN programs. The LPN game semantics is proven to be equivalent to an infinite-valued refinement of the well-founded model semantics (as defined in [34]); it is a denotational game semantics. A couple of years after that, two games to deal with DLP and DLPN (again from a denotational point of view) were described informally in [35]. The history of denotational and game semantics for these four versions of logic programming is summarized in Table 1.

In this article, a game semantics for DLP is formally defined, studied, and proven correct:

Soundness and completeness of the DLP game semantics (Theorem 6F)**.** *The game semantics of DLP is equivalent to the minimal model semantics, i.e., given any DLP program $\mathcal{P}$, and any disjunction D,*

$$\begin{array}{c} D \text{ is true wrt the} \\ \text{DLP game semantics} \end{array} \iff \begin{array}{c} D \text{ is true wrt the} \\ \text{minimal model semantics.} \end{array}$$

Two key ideas are developed to prove the two directions of the above result: *combination* (for completeness) and *splitting* (for both). In short, we begin with a finite disjunctive logic program $\mathcal{P}$, and split it in two new DLP programs $\mathcal{P}_1$ and $\mathcal{P}_2$, such that they are in a sense, “less disjunctive”. Now, strategies for games in $\mathcal{P}$ can themselves be split to strategies for games in $\mathcal{P}_1$ and $\mathcal{P}_2$, and vice versa: strategies for such games can be combined to form new strategies for games in $\mathcal{P}$. By repeated splitting, we eventually arrive at programs that are not disjunctive at all (LP programs), which we know how to deal with since [8]. Finally, compactness will allow us to extend this result to the general case of infinite DLP programs.

Why games? There are various benefits of defining a correct game semantics for each of these languages. On the operational side, the LP game helps us prune down the space of SLD derivations, by grouping them together using the much smaller space of strategies. In fact, the *alpha-beta algorithm* was used in [23] to

Language	**Denotational semantics**	$\iff$	**Game semantics**
LP	least Herbrand model [10]	$\iff$	LP game [8]
LPN	well-founded model [37]	$\iff$	LPN game [12]
DLP	minimal models [28]	$\iff$	**this article**
DLPN	∞-valued minimal models [6]	–	–

Table 1: Development of game semantics for logic programming.

speed-up the resolution strategies even for the case of *constraint logic programming*. On the denotational side, these games impart elegant characterizations of these main versions of logic programming. As it turns out, starting from LP, one only needs to add a couple of simple game-rules to its game to arrive at DLP; the addition of a different one brings you to LPN. These rules are fairly modular, so that it even makes sense to consider adding all of them simultaneously to deal with DLPN—although this has yet to be verified. Contrast the simplicity of this approach to the difficulty of treating disjunction and negation relying solely on model-theoretic tools. In addition, this kind of games is also applicable to *intensional logic programming* (as explained in [31]), and even outside of the logic programming world, e.g., to *boolean grammars* (see [19]). For an encyclopædic treatment of the use of games in logic, consult [15].

Outline. First, we formalize all the notions of DLP that we need (§**2**) and review its denotational semantics (§**3**), staying in the traditional logic programming world—no games. Then we define games (§**4**): with each DLP program $\mathcal{P}$ and each goal $\leftarrow \mathtt{G}$ we associate a specific game $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, and explain how we can use it to derive semantics for the program. Plays and strategies are thoroughly studied (§**5**), with the objective to define combination and splitting for both of them. Lastly, we put every piece together to prove that the game semantics we defined is sound and complete (§**6**) and conclude with some promising directions for further research (§**7**).

Throughout the text, a general pattern emerges: (i) define a mathematical object of DLP; (ii) proceed to define what it means to combine two such objects into a weaker one (i.e., more disjunctive); (iii) define restriction of this object to a stronger version of it (i.e., less disjunctive); (iv) use restriction to obtain splitting. We will follow these steps, again and again.

Presentation. Be aware, that even though the DLP game developed here can be thought of as another extension of the LP game, its formalization is drastically different from those of the games of either LP or LPN, and appears to be novel in the field of logic programming. It has been influenced instead by game semantics in the style of Abramsky–Jagadeesan–Malacaria and Hyland–Ong–Nickau, used for PCF and functional programming in general (see [1], [2], [16] and [30]). Although we assume no such prior knowledge of this field, the initiated reader should hopefully feel at home.

Related works. Some recent treatments benefit by incorporating tools from areas such as proof theory and linear logic into logic programming:[3] the proof search is often presented in terms of *sequent calculi*, formulæ are not necessarily restricted to Horn clauses of first-order logic, linear connectives are taken into account, etc. [25], [26], [4], and [3], are some works along this line of research, just to mention a few of them. What is more, games have been used successfully in such settings as well: in [32] and [27], for example, two different approaches for game semantics are presented in the context of *computation-as-proof-search.* Category theory has also proven itself useful in providing semantics for logic programming. For instance, [17] gives a *coalgebraic semantics* to variable-free logic programs, which is extended to first-order programs in [18].

A different school of negation in logic programming, namely *stable model semantics* (see [14]) gave rise to a relatively new kind of declarative programming: *answer set programming*, or ASP (see [13]). It allows for disjunctions (besides negations), among other things. Unsurprisingly, a game semantics approach is being investigated as well: in [29], a game is briefly outlined for programs with single-headed clauses, i.e., for the non-disjunctive fragment of ASP.

Before moving on to typographical and notational matters, it should be emphasized that this small overview of logic programming and its semantics, is far from being complete, and thus the reader should not rely on it in any way.

Notation. The Greek letters ϕ, ψ, and ρ will stand for rules, as well as for the corresponding logic formulæ. Programs will usually be denoted by calligraphic capital letters like $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$. Lowercase letters such as a, d, f, and p, will always denote atoms, while uppercase such symbols will stand for sets of atoms. For instance, D could be the set $\{d_1, \ldots, d_n\}$ which, as you shall shortly see, is identified with the disjunction $d_1 \vee \cdots \vee d_n$. We will use a monospaced font when we show their appearances in programs. We use capital "script" letters for families or sequences of sets of atoms: $\mathscr{D}$ could stand for $\langle D_1, \ldots, D_n \rangle$, each D_i being a set of atoms $\left\{d^i_1, \ldots, d^i_{k_i}\right\}$. The *truth values true* and *false* are written as $\mathbf{T}$ and $\mathbf{F}$ respectively; logical equivalence as $\equiv$.

We will work with sequences a lot; we use $+\!\!+$ for concatenation and $|\cdot|$ for length. We shall write $s \sqsubseteq s'$ to indicate that the sequence s is a prefix of s', decorating it with an "$_\mathrm{e}$" in case s is of even length: $s \sqsubseteq_\mathrm{e} s'$ (note that $\sqsubseteq_\mathrm{e} \subseteq \sqsubseteq$). Proper (even) prefixes will be shown as $\sqsubset$ ($\sqsubset_\mathrm{e}$). $s{\restriction}_n$ stands for the sequence of the first n elements of s, and it is equal to the whole sequence if $|s| \leq n$. We will frequently need to extract the longest, even, proper prefix of a sequence s; we therefore introduce the notation s^- for this, with the convention that it leaves the empty sequence unaltered: $\langle\rangle^- \overset{\mathsf{df}}{=} \langle\rangle$. Influenced by lists in programming languages, we use :: for the *cons* operator:

$$x :: s \overset{\mathsf{df}}{=} \langle x \rangle +\!\!+ s.$$

[3]This one, does not.

Products of posets are equipped with the product order by default:

$$(s_1, s_2) \sqsubseteq (s_1', s_2') \overset{\text{df}}{\iff} s_1 \sqsubseteq s_1' \text{ and } s_2 \sqsubseteq s_2'.$$

As has been just demonstrated, $\overset{\text{df}}{=}$ and $\overset{\text{df}}{\iff}$ are used to introduce the definition of a function or symbol, while $:=$ is used to "let-bind" the variables appearing on its left to the corresponding expressions that appear on its right (or the other way around).

Further notational conventions will be introduced as soon as it is sensible to do so. Thus far, we have what we need to begin.

2. Syntax

In this section, we build the theory of disjunctive logic programs that is required to develop games, and also have our first hands-on experience with combinations, restrictions, and splittings.

2.1. Preliminaries

We assume the existence of a countable set $\mathcal{A}$ of all the atoms. It is convenient to represent disjunctions as sets of atoms and conjunctions as sequences of disjunctions; when we do so, we speak of "DLP" disjunctions and conjunctions respectively. For example, the formula $\mathtt{a} \land (\mathtt{b} \lor \mathtt{c}) \land (\mathtt{c} \lor \mathtt{b})$ is represented by $\langle \{a\}, \{b, c\}, \{b, c\} \rangle$, in which the two occurrences of the set $\{b, c\}$ are distinct. Notice that under this convention all DLP conjunctions are actually formulæ in CNF. Formally, we define:

Definition 2.1 (DLP)**.** A *DLP disjunction* is a finite subset $D \subsetneq \mathcal{A}$. A *DLP conjunction* is a finite sequence $\mathscr{D}$ of DLP disjunctions. A *DLP clause* is a pair $(H, \mathscr{D})$, in which the *head* $H = \{a_1, \dots, a_n\}$ is a DLP disjunction, and the *body* $\mathscr{D} = \langle D_1, \dots, D_m \rangle$ is a DLP conjunction. If the head of a DLP clause is non-empty we call it a *DLP rule*, while if it is empty and $m = 1$, a *DLP goal*.[4] A *DLP fact* is a bodiless DLP clause. In logic programs, DLP rules will be written as

$$\underbrace{\mathtt{a}_1 \lor \cdots \lor \mathtt{a}_n}_{\text{head}} \leftarrow \underbrace{\mathtt{d}^1_1 \lor \cdots \lor \mathtt{d}^1_{s_1} \;,\; \cdots \;,\; \mathtt{d}^m_1 \lor \cdots \lor \mathtt{d}^m_{s_m}}_{\text{body}}.$$

Such a rule is called *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:

$$\mathtt{a}_1 \lor \cdots \lor \mathtt{a}_n \leftarrow \mathtt{b}_1 \;,\; \cdots \;,\; \mathtt{b}_m.$$

[4]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.

A *clean DLP program* is a countable set of clean DLP rules; it is *proper*, if at least one of its rules is proper. If we drop the condition that the DLP rules are clean we speak of a *general DLP program*. For obvious reasons we omit the "DLP" prefix whenever no confusion arises. We have not specified what those atoms in $\mathcal{A}$ 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 DLP program*. Another possibility is to let them be the atomic formulæ of a first-order language $\mathbb{L}$, built by the predicates and terms of $\mathbb{L}$. We then call it a *first-order DLP program* in $\mathbb{L}$.

▶ *Example 2.1.* Consider the following sets of rules:

$$\mathcal{P} := \left\{ \begin{aligned} \mathtt{p} &\leftarrow \mathtt{a} \\ \mathtt{p} &\leftarrow \mathtt{b} \\ \mathtt{a} \vee \mathtt{b} &\leftarrow \end{aligned} \right\}, \quad \mathcal{Q} := \left\{ \begin{aligned} \mathtt{e} \vee \mathtt{p} &\leftarrow \mathtt{f} \vee \mathtt{g}\ ,\ \mathtt{h} \\ \mathtt{p} \vee \mathtt{q} &\leftarrow \mathtt{g}\ ,\ \mathtt{e} \vee \mathtt{r} \end{aligned} \right\}, \quad \mathcal{R} := \left\{ \begin{aligned} \mathtt{d} &\leftarrow \mathtt{f} \vee \mathtt{h} \\ \mathtt{p} &\leftarrow \mathtt{g} \vee \mathtt{e} \end{aligned} \right\}.$$

The DLP program $\mathcal{P}$ is proper and clean, $\mathcal{Q}$ is proper but not clean, and $\mathcal{R}$ is neither proper nor clean. ◀

▶ *Example 2.2.* Here is a first-order DLP program:

$$\left\{ \begin{aligned} \mathsf{p}(X) &\leftarrow \mathsf{a}(X)\ ,\ \mathsf{c}(X) \\ \mathsf{p}(X) &\leftarrow \mathsf{b}(X) \\ \mathsf{a}(\mathsf{mary}) \vee \mathsf{b}(\mathsf{mary}) &\leftarrow \\ \mathsf{c}(\mathsf{mary}) &\leftarrow \end{aligned} \right\}.$$

It is proper and clean. ◀

Remark 2.1 (Propositional vs first-order). Once we have reviewed the denotational semantics of DLP programs in the next section, it will become apparent that infinite, propositional programs are as powerful as finite, first-order ones. Hence, for simplicity, we will be dealing with propositional DLP programs unless mentioned otherwise. To see how we end up with infinite programs, start from a non-propositional, finite program $\mathcal{P}$, containing at least one function symbol, and replace each of its rules by all of its ground instances. What you get is a countably infinite program with equivalent denotational semantics. The definition and the example that follow will make this more precise.

Definition 2.2 (Ground)**.** If a term, an atom, a formula, or a set of formulæ of a first-order language $\mathbb{L}$ contains no variables, it is called *ground*. If it does, then by replacing all of its variables with ground terms of $\mathbb{L}$, we obtain a *ground instance* of it. We also define:

$$\begin{aligned} \mathsf{ground}(\phi) &\overset{\mathsf{df}}{=} \{\phi' \mid \phi' \text{ is a ground instance of } \phi\} \\ \mathsf{ground}(\mathcal{P}) &\overset{\mathsf{df}}{=} \bigcup \{\mathsf{ground}(\phi) \mid \phi \in \mathcal{P}\}. \end{aligned}$$

▶ *Example 2.3.* Consider the program

$$\mathcal{E} := \left\{ \begin{array}{c} \mathtt{even}(0) \leftarrow \\ \mathtt{even}(\mathsf{S}(\mathsf{S}(X))) \leftarrow \mathtt{even}(X) \end{array} \right\}.$$

We compute:

$$\mathsf{ground}(\mathcal{E}) := \left\{ \begin{array}{c} \mathtt{even}(0) \leftarrow \qquad\qquad \\ \mathtt{even}(\mathsf{S}^2(0)) \leftarrow \mathtt{even}(0) \\ \mathtt{even}(\mathsf{S}^3(0)) \leftarrow \mathtt{even}(\mathsf{S}(0)) \\ \mathtt{even}(\mathsf{S}^4(0)) \leftarrow \mathtt{even}(\mathsf{S}^2(0)) \\ \vdots \end{array} \right\}, \quad \mathcal{E}_0 := \left\{ \begin{array}{c} \mathtt{e}_0 \leftarrow \quad \\ \mathtt{e}_2 \leftarrow \mathtt{e}_0 \\ \mathtt{e}_3 \leftarrow \mathtt{e}_1 \\ \mathtt{e}_4 \leftarrow \mathtt{e}_2 \\ \vdots \end{array} \right\},$$

where $\mathcal{E}_0$ is a propositional DLP program, equivalent to $\mathsf{ground}(\mathcal{E})$. ◀

REMARK 2.2 (LP, DLP, and logic). There is an obvious mapping of program rules into logic formulæ. For instance, the DLP rule

$$\mathtt{a}_1 \vee \cdots \vee \mathtt{a}_n \leftarrow \mathtt{d}^1_1 \vee \cdots \vee \mathtt{d}^1_{s_1} \;,\; \cdots \;,\; \mathtt{d}^m_1 \vee \cdots \vee \mathtt{d}^m_{s_m}$$

corresponds to the formula

$$\left(d^1_1 \vee \cdots \vee d^1_{s_1}\right) \wedge \cdots \wedge \left(d^m_1 \vee \cdots \vee d^m_{s_m}\right) \rightarrow a_1 \vee \cdots \vee a_n.$$

This allows us to directly use some well-known jargon from Mathematical Logic: we speak of models, theories, consistency, logical consequences, etc.

Definition 2.3. On the set of rules we define two operators head and body as the projections

$$\mathsf{head}((H, \mathscr{D})) \overset{\text{df}}{=} H,$$
$$\mathsf{body}((H, \mathscr{D})) \overset{\text{df}}{=} \mathscr{D}.$$

REMARK 2.3 (Disjunctions in bodies). Following [22, §2.3], a *disjunctive clause* is the universal closure of a logic formula like

$$L_1 \vee \cdots \vee L_k,$$

where the L_i's are literals. Separating them into positive and negative, and omitting the quantifiers, this clause can be brought to the form

$$a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_m,$$

or, equivalently (by De Morgan) to

$$a_1 \vee \cdots \vee a_n \vee \neg\,(b_1 \wedge \cdots \wedge b_m),$$

which, in turn, is logically equivalent to the (reverse) implication

$$a_1 \vee \cdots \vee a_n \leftarrow b_1 \wedge \cdots \wedge b_m.$$

We adopt this as the logic programming notation of a clause, writing commas instead of $\wedge$. Coming this way, it is impossible for a disjunction to appear in the body of a rule. Here though, we bypass this construction as it is often more natural to express ideas using "unclean" rules.

When we are working with clean DLP programs, we can consider disjunctions in bodies as "syntactic sugar", thanks to the following transformation.

Definition 2.4 ($\widehat{\mathcal{P}}$ and $\widehat{\phi}$)**.** Let $\mathcal{P}$ be a general DLP program. Then $\widehat{\mathcal{P}}$ is the DLP program that results if we replace every unclean rule $\phi = (H, \langle D_1, \dots, D_n \rangle)$ of $\mathcal{P}$ by all rules in

$$\widehat{\phi} \overset{\text{df}}{=} \{(H, C) \mid C \in D_1 \times \cdots \times D_n\}.$$

We call $\widehat{\mathcal{P}}$ the *clean version* of $\mathcal{P}$.

Property 2.1. *$\widehat{\mathcal{P}}$ is clean and logically equivalent to $\mathcal{P}$.*

This is a simple case of the most general *Lloyd–Topor transformation*, which transforms logic programs containing arbitrary formulæ in their bodies into normal ones. See [21], [20, Chapter 4] or [22, pp. 188–189] for more details.

► *Example 2.4.* Here is the desugaring of $\mathcal{Q}$ from the previous example:

$$\mathcal{Q} := \left\{ \begin{matrix} \mathtt{e} \vee \mathtt{p} \leftarrow \mathtt{f} \vee \mathtt{g}\ ,\ \mathtt{h} \\ \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{g}\ ,\ \mathtt{e} \vee \mathtt{r} \end{matrix} \right\} \quad \overset{\wedge}{\longmapsto} \quad \widehat{\mathcal{Q}} := \left\{ \begin{matrix} \mathtt{e} \vee \mathtt{p} \leftarrow \mathtt{f}\ ,\ \mathtt{h} \\ \mathtt{e} \vee \mathtt{p} \leftarrow \mathtt{g}\ ,\ \mathtt{h} \\ \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{g}\ ,\ \mathtt{e} \\ \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{g}\ ,\ \mathtt{r} \end{matrix} \right\}. \qquad ◄$$

In the sequel, whenever we refer to DLP programs, we mean general programs. However, all the fundamental program constructions that we investigate easily preserve "cleanliness"; the one time that it really makes a difference is in §**6**: there, we prove soundness and completeness for clean DLP programs first, and then proceed to consider general DLP programs.

2.2. Disjunctive combinations

As mentioned in the introduction, given a sequence of disjunctions, we will frequently want to *disjunctively combine* them into a single DLP disjunction. In a similar fashion, we wish to combine DLP conjunctions, DLP rules, and later even plays and strategies! Informally speaking, the idea is always the same: we combine two or more "DLP things" into a single such thing, by using some kind of logical disjunction. Thus, the resulting combination will be "more disjunctive" than either of them. Even though we use the same notation for all of the different kinds of disjunctive combinations, it will always be clear what type of elements we are dealing with, and so no confusion should arise.

This overloaded notation is rather convenient and really pays off in terms of readability.

In the definitions that follow, we introduce combination to deal with DLP disjunctions, conjunctions, and rules.

Definition 2.5 (Combination of disjunctions)**.** The (disjunctive) *combination* of two DLP disjunctions is simply their union:

$$D \curlyvee E \overset{\mathsf{df}}{=} D \cup E.$$

Definition 2.6 (Combination of conjunctions)**.** Given two DLP conjunctions $\mathscr{D} \coloneqq \langle D_1, \dots, D_n \rangle$ and $\mathscr{E} \coloneqq \langle E_1, \dots, E_m \rangle$, their *combination* is another DLP conjunction, logically equivalent to $\mathscr{D} \vee \mathscr{E}$, and defined by the following equation:

$$\mathscr{D} \curlyvee \mathscr{E} \overset{\mathsf{df}}{=} \langle D_1 \curlyvee E_1, \dots, D_1 \curlyvee E_m, \dots, D_n \curlyvee E_1, \dots, D_n \curlyvee E_m \rangle .$$

Notice that the sequence on the right is empty iff any of $\mathscr{D}$ or $\mathscr{E}$ is empty.

Definition 2.7 (Combination of rules)**.** The *combination* of two DLP rules $\phi_1 \coloneqq (H_1, \mathscr{D}_1)$ and $\phi_2 \coloneqq (H_2, \mathscr{D}_2)$ is the rule defined by

$$\phi_1 \curlyvee \phi_2 \overset{\mathsf{df}}{=} (H_1 \curlyvee H_2, \mathscr{D}_1 \curlyvee \mathscr{D}_2).$$

Remark 2.4. It follows that $\phi_1 \curlyvee \phi_2$ will not be a clean rule in general, unless one of the ϕ_i's is a fact. This is the only construction that does not preserve cleanliness. However, we will only use it to extract the head of the combined rule (which causes no trouble),[5] and not to create new, potentially unclean rules.

▶ *Example 2.5.* Combining the following two rules

$$\begin{array}{ll} \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a} \vee \mathtt{b} \ \mathtt{,} \ \mathtt{c} & \text{i.e.,} \quad (\{p,q\}, \langle \{a,b\}, \{c\} \rangle) \\ \mathtt{p} \vee \mathtt{r} \leftarrow \mathtt{d} \ \mathtt{,} \ \mathtt{e} & \text{i.e.,} \quad (\{p,r\}, \langle \{d\}, \{e\} \rangle) \end{array}$$

we obtain

$$\mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow \mathtt{a} \vee \mathtt{b} \vee \mathtt{d} \ \mathtt{,} \ \mathtt{a} \vee \mathtt{b} \vee \mathtt{e} \ \mathtt{,} \ \mathtt{c} \vee \mathtt{d} \ \mathtt{,} \ \mathtt{c} \vee \mathtt{e},$$

i.e., $(\{p,q,r\}, \langle \{a,b,d\}, \{a,b,e\}, \{c,d\}, \{c,e\} \rangle)$. ◀

Hitherto we have defined combination for pairs of disjunctions, conjunctions, and rules. We can generalize these definitions from pairs to sequences in a straightforward way:

Definition 2.8 (Combining sequences)**.** Given a sequence $\langle T_1, \dots, T_n \rangle$ of DLP disjunctions, conjunctions, or rules, we set

$$[\,\langle T_1, \dots, T_n \rangle\,] \overset{\mathsf{df}}{=} T_1 \curlyvee \cdots \curlyvee T_n,$$

[5] Notice that $\mathsf{head}(\phi_1 \curlyvee \phi_2) = \mathsf{head}(\phi_1) \curlyvee \mathsf{head}(\phi_2)$.

where $\curlyvee$ is understood to associate to the left, and $\emptyset$, $\langle\emptyset\rangle$, or $(\emptyset, \langle\emptyset\rangle)$ is its unit, depending on whether we are combining DLP disjunctions, conjunctions, or rules respectively.

An alternative presentation of the same definition uses recursion:

$$[\,\langle\,\rangle\,] \overset{\text{df}}{=} \begin{cases} \emptyset & \text{(DLP disjunctions)} \\ \langle\emptyset\rangle & \text{(DLP conjunctions)} \\ (\emptyset, \langle\emptyset\rangle) & \text{(DLP rules)} \end{cases}$$

$$[\,\langle T_1, \dots, T_{n+1}\rangle\,] \overset{\text{df}}{=} [\,\langle T_1, \dots, T_n\rangle\,] \curlyvee T_{n+1}.$$

2.3. Restrictions and splitting

Bypassing definitions of restriction and splitting for disjunctions and conjunctions, we proceed to define those notions directly for rules and programs.

Definition 2.9 (Rule restriction)**.** For any given rule ϕ and any set of atoms A, we define the *restriction* of ϕ to A by

$$\phi|_A \overset{\text{df}}{=} (\mathsf{head}(\phi) \cap A, \mathsf{body}(\phi))\,.$$

It follows that a rule's body is unaffected by restriction: $\mathsf{body}(\phi) = \mathsf{body}(\phi|_A)$.

Definition 2.10 (Program restriction)**.** Let $\mathcal{P}$ be a DLP program and let $\phi \in \mathcal{P}$. Then for any set of atoms A, we can define the *restricted program* $\mathcal{P}|_A^\phi$ by restricting the rule ϕ of $\mathcal{P}$ to A:

$$\mathcal{P}|_A^\phi \overset{\text{df}}{=} (\mathcal{P} \setminus \{\phi\}) \cup \{\phi|_A\}\,.$$

In words, $\mathcal{P}|_A^\phi$ is the program which is identical to $\mathcal{P}$, with the exception that the rule ϕ has been replaced by $\phi|_A$.

► *Example 2.6.* Let $\mathcal{P}$ be the proper DLP program

$$\mathcal{P} \coloneqq \left\{ \begin{array}{c} \mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow \mathtt{f}\ ,\ \mathtt{g} \\ \mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow \mathtt{a}\ ,\ \mathtt{c} \\ \mathtt{f} \leftarrow \end{array} \right\},$$

and let $\phi \coloneqq \mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow \mathtt{a}\ ,\ \mathtt{c}$. Here is a couple of restrictions of $\mathcal{P}$ with respect to ϕ:

$$\mathcal{P}|_{\{p,q\}}^\phi \coloneqq \left\{ \begin{array}{r} \mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow \mathtt{f}\ ,\ \mathtt{g} \\ \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}\ ,\ \mathtt{c} \\ \mathtt{f} \leftarrow \quad\quad \end{array} \right\}, \quad \mathcal{P}|_{\{r\}}^\phi \coloneqq \left\{ \begin{array}{r} \mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow \mathtt{f}\ ,\ \mathtt{g} \\ \mathtt{r} \leftarrow \mathtt{a}\ ,\ \mathtt{c} \\ \mathtt{f} \leftarrow \quad\quad \end{array} \right\}. \quad ◄$$

We usually fix a disjunction H, and break it into logically stronger, less disjunctive parts. For this we introduce the notion of a proper partition:

Definition 2.11 (Proper partition)**.** Suppose that H is a set of atoms. Then a tuple $\mathcal{H} \coloneqq (H_1, \dots, H_n)$ is a proper partition of H iff

(i) $\emptyset \neq H_i \subsetneq H$ for all i;

(ii) $H = H_1 \cup \cdots \cup H_n$;

(iii) $H_i \cap H_j = \emptyset$ for all $i \neq j$.

Once we know how to restrict a program, splitting it with respect to some partition $\mathcal{H}$ becomes trivial:

Definition 2.12 (Splitting a program)**.** Let ϕ be a proper DLP rule, and let $\mathcal{H} \coloneqq (H_1, \dots, H_n)$ be a proper partition of its head. Then the *splitting of* $\mathcal{P}$ *with respect to* ϕ *over* $\mathcal{H}$ is the tuple

$$\mathcal{P}|^{\phi}_{\mathcal{H}} \overset{\text{df}}{=} \left(\mathcal{P}|^{\phi}_{H_1}, \dots, \mathcal{P}|^{\phi}_{H_n}\right).$$

► *Example 2.7.* The pair $\left(\mathcal{P}|^{\phi}_{\{p,q\}}, \mathcal{P}|^{\phi}_{\{r\}}\right)$ of Example 2.6 is the splitting of $\mathcal{P}$ with respect to ϕ over $\mathcal{H} \coloneqq (\{p,q\}, \{r\})$. ◄

We have formalized disjunctive logic programs, and are now ready to define their semantics.

3. Declarative semantics

In this section, we define the denotational or declarative semantics for DLP programs:[6] we present the minimal model semantics of Minker (see [28] or [22]). This is the *de facto* denotational semantics for disjunctive logic programs, and will be the criterion for the soundness and completeness of our game semantics. Before diving into the minimal model semantics of DLP, we quickly review the denotational semantics of LP, i.e., the least Herbrand model. But we are by no means thorough; consult [20] for more information.

3.1. Declarative semantics of LP—The least Herbrand model

Given a *propositional* logic program $\mathcal{P}$, we define its *Herbrand base* $\mathrm{HB}(\mathcal{P})$ to be the set of all atoms that appear in $\mathcal{P}$. By a *Herbrand interpretation* $\mathfrak{I}$ of $\mathcal{P}$, we mean any assignment of truth values ($\{\mathbf{T}, \mathbf{F}\}$) to the elements of the Herbrand base. We say that $\mathfrak{I}$ satisfies a rule $\mathtt{p} \leftarrow \mathtt{a}_1 \texttt{ , } \cdots \texttt{ , } \mathtt{a}_n$, if it satisfies the logic formula $a_1 \wedge \cdots \wedge a_n \to p$. A Herbrand interpretation that satisfies every rule of $\mathcal{P}$ is called a *Herbrand model* of $\mathcal{P}$. It is customary in the logic programming literature to identify interpretations with sets of atoms when the underlying logic is two-valued: for any interpretation $\mathfrak{I}$,

$$a \in \mathfrak{I} \overset{\text{df}}{\iff} \mathfrak{I}(a) = \mathbf{T}.$$

[6]We use the terms "denotational semantics" and "declarative semantics" interchangeably.

LP programs enjoy the following very useful property:

Model intersection property. *The intersection of a non-empty family of models is itself a model.*

Observe now that for any program $\mathcal{P}$ at least one satisfying Herbrand interpretation exists; to wit, the Herbrand base itself (i.e., the interpretation that assigns the truth value $\mathbf{T}$ to every element of the Herbrand base). Therefore the family of all Herbrand models $\mathrm{HM}(\mathcal{P})$ is always non-empty, which allows us to define the *least Herbrand model* as the intersection of this family:

$$\mathrm{LHM}(\mathcal{P}) \overset{\mathrm{df}}{=} \bigcap \mathrm{HM}(\mathcal{P}).$$

Definition 3.1 (Least Herbrand model semantics)**.** Let $\mathcal{P}$ be an LP program. The goal $\leftarrow \mathtt{p}$ *succeeds* if $p \in \mathrm{LHM}(\mathcal{P})$.

The following result, due to van Emden and Kowalski, justifies the use of $\mathrm{LHM}(\mathcal{P})$ as the denotational semantics of $\mathcal{P}$.

Theorem 3A. *Let $\mathcal{P}$ be an LP program. Then*

$$\mathrm{LHM}(\mathcal{P}) = \{p \in \mathrm{HB}(\mathcal{P}) \mid p \textit{ is a logical consequence of } \mathcal{P}\}.$$

Proof. See [10, §5] or [20, Theorem 6.2] ∎

This concludes our short summary of LP semantics. We proceed to DLP.

3.2. Declarative semantics of DLP—Minimal models

We summarize the minimal model semantics of disjunctive logic programming, using the following DLP program as an example:

$$\mathcal{P} := \left\{ \begin{array}{r} \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{a} \vee \mathtt{b} \leftarrow\phantom{\mathtt{a}} \end{array} \right\}.$$

First, we compute its Herbrand models:

$$\{a, p\}, \quad \{b, p\}, \quad \{a, b, p\}.$$

If we try to follow the practice of LP, we will want to select the $\subseteq$-least Herbrand model to provide semantics for $\mathcal{P}$. But none of them is least! The model intersection property which LP programs enjoy, fails to hold in the presence of disjunctions:

$$\bigcap \mathrm{HM}(\mathcal{P}) = \bigcap \{\{a, p\}, \{b, p\}, \{a, b, p\}\} = \{p\}.$$

And $\{p\}$ is not a model, since according to the third rule of our program, at least one of the atoms a or b must be true. However, the first two models are

$\subseteq$-*minimal.* In fact, we can rely on the set $\{\{a,p\},\{b,p\}\}$ of minimal models to obtain semantics for $\mathcal{P}$.

Let $\mathcal{P}$ be a DLP program. We write $\mathrm{MM}(\mathcal{P})$ for the set of all minimal Herbrand models of $\mathcal{P}$. By the definition that follows, $\mathrm{MM}(\mathcal{P})$ provides the denotational semantics for the DLP program $\mathcal{P}$, which we call the *minimal model semantics* à la Minker.

Definition 3.2 (Minimal model semantics)**.** Let $\mathcal{P}$ be a DLP program. The goal $\leftarrow$ `G` *succeeds* if `G` is true in every minimal Herbrand model of $\mathcal{P}$.

The equivalent of Theorem 3A for the disjunctive case is due to Minker:

Theorem 3B. *Let $\mathcal{P}$ be a DLP program. A ground clause C is a logical consequence of $\mathcal{P}$ iff C is true in every minimal Herbrand model of $\mathcal{P}$.*

PROOF. See [28] or [22, Theorem 3.3] ∎

▶ *Example 3.1.* Consider the DLP program

$$\mathcal{Q} := \left\{ \begin{array}{r} \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{b} \leftarrow \mathtt{c} \\ \mathtt{a} \vee \mathtt{c} \leftarrow \phantom{\mathtt{a}} \end{array} \right\},$$

compute its Herbrand models, and identify the ones that are minimal:

$$\begin{aligned} \mathrm{HM}(\mathcal{Q}) &= \left\{\underline{\{a,p\}}, \{a,b,p\}, \underline{\{c,b,p\}}, \{a,b,c,p\}\right\}, \\ \mathrm{MM}(\mathcal{Q}) &= \{\{a,p\},\{c,b,p\}\}. \end{aligned}$$

Under the minimal model semantics, p and $a \vee c$ are both $\mathbf{T}$, as

$$\begin{array}{rcr} \{a,p\} \models p & & \{a,p\} \models a \vee c \\ & \text{and} & \\ \{c,b,p\} \models p & & \{c,b,p\} \models a \vee c, \end{array}$$

while both a and $b \vee c$ are $\mathbf{F}$ because of

$$\{c,b,p\} \not\models a \qquad \text{and} \qquad \{a,p\} \not\models b \vee c.$$ ◀

REMARK 3.1. In [22], another denotational semantics for DLP is defined, which they call the *least model-state semantics*. However, they prove that the two approaches are equivalent, so we stick with the minimal model semantics here.

Looking back at the definition of rule restriction, we can now see that it implies the following property:

Property 3.1. *The restriction of a rule ϕ is stronger than the original rule, in the sense that any interpretation which satisfies $\phi|_A$ must also satisfy ϕ. In the same sense, restricting a DLP program makes it stronger.*

3.3. Declarative semantics of first-order DLP

Even though we will base the development of our game semantics completely on propositional DLP programs, we summarize below the corresponding notions of the ones we described above, for the first-order case. They are presented in detail in [22, §2.2, §2.4, §3.2]. The reader is advised to skip this subsection on a first reading, coming back to it only before the very last result of §**6**: Corollary 6.4.

Given a first-order DLP program $\mathcal{P}$ in some first-order language $\mathbb{L}$, we define:

Definition 3.3. The *Herbrand universe* of $\mathcal{P}$, $\mathrm{HU}(\mathcal{P})$, is the set of all ground terms which can be formed by constants and function symbols of $\mathbb{L}$. The *Herbrand base* of $\mathcal{P}$, $\mathrm{HB}(\mathcal{P})$, is the set of all ground atoms which can be formed using the predicate symbols of $\mathbb{L}$ on the ground terms of $\mathrm{HU}(\mathcal{P})$.

Definition 3.4. A *Herbrand interpretation* of $\mathcal{P}$ is a mapping $\mathfrak{I}$ that maps each n-ary predicate symbol p of $\mathbb{L}$ to an n-ary relation on $\mathrm{HU}(\mathcal{P})$. If $\mathfrak{I}$ is a model of $\mathcal{P}$ we call it a *Herbrand model* of $\mathcal{P}$.

Notice, that since the Herbrand models of a first-order logic program $\mathcal{P}$ are constructed using only ground instances of rules in $\mathcal{P}$, the following property is immediate:

Property 3.2. *Let $\mathcal{P}$ be a first-order DLP. Then $\mathcal{P}$ and* $\mathsf{ground}(\mathcal{P})$ *have the same minimal models:*

$$\mathrm{MM}(\mathcal{P}) = \mathrm{MM}(\mathsf{ground}(\mathcal{P})).$$

Remark 3.2. As long as we are studying denotational semantics, the above property allows us to focus on infinite, propositional DLP programs. This is a very common practice in the field of logic programming semantics. In fact, we assume given an implementation of our DLP language (based on some operational semantics—which, is immaterial). We then load our finite, first-order DLP program $\mathcal{P}$, and the implementation computes answers to our queries. Then, a denotational semantics of the equivalent, propositional, and infinite DLP program $\mathsf{ground}(\mathcal{P})$, provides a correctness criterion for those answers. Thus, we avoid variables and function symbols at the cost of finiteness, but this is a fair bargain, as no difficulties arise on the declarative side of semantics (see also [11] for a relevant discussion).[7]

This completes our study of the syntax and the semantics of DLP; we now have just enough tools to develop the DLP game, and use it to define the game semantics of DLP. For an extensive study of the foundations of disjunctive logic programming, the interested reader is once more referred to [22], whose treatment far exceeds our needs.

[7] In exactly the same sense, when giving a fixpoint semantics, we only use the set of ground instances of clauses in $\mathcal{P}$ to define $T_{\mathcal{P}}$ (see [20] and [22]).

4. Games

In this section, we define the DLP game. First, we describe the LP game in our propositional setting. Next, we present a simplified, informal version of the actual game that we will use for DLP, and then, we proceed to make things formal. Our objective is to use this game, in order to give a game semantics in terms of the model-theoretic semantics outlined in the previous section.

4.1. An outline of the propositional LP game

The general picture, shared by all of the games that we consider here, is based on Lorenzen dialogue games (see [24]). Two players (Doubter vs Believer) argue over whether a given goal $\leftarrow$ `p` succeeds or not.[8] The player who doubts it (Doubter) begins the game by saying:

Doubter: "Why `p`?".

The defending player (Believer) must give a convincing argument of why he thinks that `p` is true. He must select a program rule that has `p` as its head and play it. For instance, selecting `p` $\leftarrow$ `a , b , c`, he replies:

Believer: "`p` because `a`, `b`, and `c`.",

to which Doubter must respond by doubting a specific atom from the body, viz. `a`, `b`, or `c`. Selecting the second one, for example, she replies:

Doubter: "Why `b`?".

The game continues in this manner, until a player cannot argue anymore, in which case they lose the game. This means that either Believer played a rule with an empty body (i.e., a fact) or Doubter played an atom which is not the head of any program rule.

This conveys the essence of the games we use for logic programming; the DLP game defined here is based on the same principles. We will denote believer moves by β and doubter moves by δ, and refer to the game just described as simply "LP game".

4.2. The simplified DLP game[9]

This game extends the one above to deal with disjunctions. First of all, the goal here is assumed to consist of a *disjunction* of atoms. The key difference from the LP (and LPN) games is that *in the DLP game the Believer can play combo moves: he can use more than one rule to support his belief.* What is more, he can use *not only* rules from the given program, but also *implicit rules*,

[8]We agree to refer to Believer as a *he*, and to Doubter as a *she*. There is no particular reason for the specific choice, but keeping their genres separate makes talking about them easier. I first encountered this convention in [36], and I adopted it.

[9]This simplified game is based on a corrected version of the DLP game described in [35].

i.e., rules of the form $\mathtt{a} \leftarrow \mathtt{a}$. Thanks to rule combination, he can disjunctively combine his selection of rules into one, say $\mathtt{G} \leftarrow \mathtt{D}_1 \;,\; \cdots \;,\; \mathtt{D}_n$, and play it against his opponent:

$$\text{Believer:}\quad \text{“}\mathtt{G}\text{ because }\mathtt{D}_1, \mathtt{D}_2, \dots, \text{ and }\mathtt{D}_n\text{.”;}$$

and it is now Doubter's turn to choose which disjunction $\mathtt{D}_i$ to doubt:

$$\text{Doubter:}\quad \text{“Why }\mathtt{D}_i\text{?”.}$$

And the game goes on.

To sum up, Believer is constantly challenged by Doubter to justify why he believes some disjunction δ. He does that by $\curlyvee$-combining a sequence of DLP rules to form a single rule β, such that the head of β is a *subset* of δ. Doubter must then select which disjunction from the body of β she doubts, and so on. Informally, this can be further summarized thus:

$$\text{DLP game} = \text{LP game} + \text{implicit rules} + \text{combo moves.}$$

The decision to allow the believer to include implicit rules not from the program is backed up by the following example:[10]

► *Example 4.1.* Consider the DLP program

$$\mathcal{P} := \left\{ \begin{array}{r} \mathsf{p} \leftarrow \mathsf{a} \\ \mathsf{p} \leftarrow \mathsf{b} \\ \mathsf{b} \leftarrow \mathsf{c} \\ \mathsf{a} \vee \mathsf{c} \leftarrow \phantom{\mathsf{a}} \end{array} \right\}.$$

For the goal $\leftarrow \mathtt{p}$, two plays in this game could look like this:

$$\pi_1 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathsf{p}} \\ \;\mathsf{D}_0 : \mathsf{p} \\ \;\mathsf{B}_0 : \mathsf{p} \leftarrow \underline{\mathsf{a} \vee \mathsf{b}} \\ \;\mathsf{D}_1 : \mathsf{a} \vee \mathsf{b} \\ \;\mathsf{B}_1 : \mathsf{a} \vee \mathsf{b} \leftarrow \underline{\mathsf{a} \vee \mathsf{c}} \\ \;\mathsf{D}_2 : \mathsf{a} \vee \mathsf{c} \\ \;\mathsf{B}_2 : \mathsf{a} \vee \mathsf{c} \leftarrow \end{array} \right| , \qquad \pi_2 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathsf{p}} \\ \;\mathsf{D}_0 : \mathsf{p} \\ \;\mathsf{B}_0 : \mathsf{p} \leftarrow \underline{\mathsf{a} \vee \mathsf{b}} \\ \;\mathsf{D}_1 : \mathsf{a} \vee \mathsf{b} \\ \;\mathsf{B}_1 : \mathsf{b} \leftarrow \underline{\mathsf{c}} \\ \;\mathsf{D}_2 : \mathsf{c} \end{array} \right| .$$

In both plays, Believer justifies the move B_0 by combining the first two program rules. Then, in π_1, he combines the *program* rule $\mathtt{b} \leftarrow \mathtt{c}$ with the *implicit* rule $\mathtt{a} \leftarrow \mathtt{a}$, while in π_2, he only uses program rules. You can easily verify that without implicit rules it is impossible for him to win this game. ◄

[10]The need for implicit rules was Olivier Laurent's objection to the correctness of the DLP game as I describe it in [35], according to which the believer can only combine program rules. I am grateful to him for spotting this mistake.

Conveniently enough, if we underline Doubter's selections—as we have done in the example above—we can condense each play by entirely omitting the lines that correspond to her moves. For the sake of laziness, we will follow this practice in the sequel.

Note that in both plays of Example 4.1, since every believer move has a body with only one element in it, Doubter does not really have any choice to make; she is simply following the lead of Believer. Here is an example where she can actually enjoy the game as well:

▶ *Example 4.2.* The program now is

$$\mathcal{Q} := \left\{ \begin{array}{r} \mathtt{p} \leftarrow \mathtt{a}\ ,\ \mathtt{b} \\ \mathtt{q} \leftarrow \mathtt{a}\ ,\ \mathtt{b} \\ \mathtt{a} \leftarrow \mathtt{d}\ ,\ \mathtt{c} \\ \mathtt{b} \leftarrow \phantom{\mathtt{a}\ ,\ \mathtt{b}} \\ \mathtt{c} \lor \mathtt{d} \leftarrow \mathtt{b} \phantom{\ ,\ \mathtt{b}} \end{array} \right\},$$

and the goal is $\leftarrow \mathtt{p} \lor \mathtt{q}$. Here are two valid plays for this game:

$$\pi_1 := \left| \begin{array}{rl} \mathsf{goal}: & \leftarrow \underline{\mathtt{p} \lor \mathtt{q}} \\ \mathsf{B}_0: & \mathtt{p} \lor \mathtt{q} \leftarrow \mathtt{a} \lor \mathtt{a}\ ,\ \underline{\mathtt{a} \lor \mathtt{b}}\ ,\ \mathtt{a} \lor \mathtt{b}\ ,\ \mathtt{b} \lor \mathtt{b} \\ \mathsf{B}_1: & \mathtt{b} \leftarrow \end{array} \right|,$$

won by Believer, and

$$\pi_2 := \left| \begin{array}{rl} \mathsf{goal}: & \leftarrow \underline{\mathtt{p} \lor \mathtt{q}} \\ \mathsf{B}_0: & \mathtt{p} \lor \mathtt{q} \leftarrow \underline{\mathtt{a} \lor \mathtt{a}}\ ,\ \mathtt{a} \lor \mathtt{b}\ ,\ \mathtt{a} \lor \mathtt{b}\ ,\ \mathtt{b} \lor \mathtt{b} \\ \mathsf{B}_1: & \mathtt{a} \leftarrow \mathtt{d}\ ,\ \underline{\mathtt{c}} \end{array} \right|,$$

by Doubter. ◀

Having seen the basic idea of the simplified game, it is time to formalize things: we define the real DLP game.

4.3. The DLP game

To study this game and prove it correct, we need to refine it substantially. To be specific, believer moves will not be played as a single (combined) DLP rule as was done in the simplified version. Instead, Believer will now specify the exact sequence of rules that he combined to create his move. Likewise, Doubter will not select a single disjunction from the combined move of Believer; instead she will select a sequence of occurrences: *one from each body* of the rules in β—which, we stress, is now a *sequence* of rules. Things become clearer and formal after the definitions and the examples that follow.

Given a DLP program $\mathcal{P}$ and a goal clause $\leftarrow \mathtt{G}$, we will define the associated DLP game, and denote it by $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. It will always be a two-player game (Doubter vs Believer). There are two *player rôles*: the doubter and the believer.

The statement	**is read as...**
D	“Why D?” or “I doubt D.”
E $\leftarrow$ D$_1$, $\cdots$, D$_n$	“E because D$_1$, ..., and D$_n$.”
E $\leftarrow$	“E because it is a fact.”

Table 2: How to read believer and doubter statements.

The rôles of the players never change throughout the game; this is why we simply call the players Doubter and Believer.[11]

Doubter starts by doubting G, the body of the goal clause, and Believer tries to defend it. A player who cannot play a valid move loses the game. Doubter has the *benefit of the doubt*, which means that if she can keep doubting forever, she wins. See Remark 4.7 (p. 25) for a discussion about the reasoning behind this decision. Now let us be precise.

Definition 4.1 (Extended program)**.** Given any set of DLP rules $\mathcal{P}$, we can extended it to $\mathcal{P}_+$, which includes all meaningful implicit rules. In detail,

$$\mathcal{P}_+ \overset{\text{df}}{=} \mathcal{P} \cup \{\mathtt{a} \leftarrow \mathtt{a} \mid a \in \mathrm{HB}(\mathcal{P})\}\,.$$

Definition 4.2 (Moves)**.** A *doubter move* δ is a sequence of occurrences of disjunctions in bodies of DLP clauses, We refer to these occurrences as the *doubts* of δ. A *believer move* β from $\mathcal{P}$ is a finite sequence of DLP rules from the extended set $\mathcal{P}_+$. Given a move m, we call $[\,m\,]$ the *statement* of the move and refer to the sequence m as the *justification* of the statement. We say that the elements of a believer move β that belong to $\mathcal{P}$ constitute the *proper part* of the justification; the rest, the *implicit.* If $|\beta| > 1$ we call β a *combo move*.

Table 2 suggests how we can read moves aloud. Notice that they resemble an actual dialogue. This motivates the following definition:

Definition 4.3 (Dialogue)**.** A *quasidialogue* from $\mathcal{P}$ is a finite sequence

$$\pi := \langle \delta_0, \beta_0, \delta_1, \beta_1, \dots \rangle\,,$$

such that:

- for all i, δ_i is a doubter move and β_i a believer move (if they exist);
- for all i, if β_i exists then $\mathsf{head}([\,\beta_i\,]) \subseteq [\,\delta_i\,]$;
- for all $i > 0$, if δ_i is $\langle D_1, \dots, D_k\rangle$, and β_{i-1} is $\langle \psi_1, \dots, \psi_{k'}\rangle$ then $k = k'$ and $D_j \in \mathsf{body}(\psi_j)$ for all $1 \leq j \leq k$.

[11]This is not the case for the LPN game: to capture the third, “unknown” truth value of the well-founded model, we need to allow ties in the game, as well as *rôle-switching* moves. See [12] for more information on this.

It is a *dialogue* if it also satisfies:

- for all i, if β_i exists then $\beta_i \cap \mathcal{P} \neq \emptyset$,

i.e., the believer always selects at least one rule from $\mathcal{P}$.

REMARK 4.1. Regarding the first restriction imposed on believer moves, one could make the seemingly stronger demand that $\mathsf{head}([\beta_i]) = [\delta_i]$. It is easy to check, however, that this changes nothing in the game, since the believer can bring up implicit rules to combine them with his move in case the subset was proper. To see that he can still win exactly the same arguments, remember that by definition, if there is at least one fact in a combination, the resulting rule is also a fact.

Definition 4.4 (Play)**.** A *(quasi)play* of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ is a (quasi)dialogue π from $\mathcal{P}$ which, if non-empty, satisfies the additional property that $[\delta_0] = \mathtt{G}$. We denote the empty play by $\varepsilon \overset{\mathsf{df}}{=} \langle\,\rangle$.

Note that dialogues (and plays) are sequences and thus inherit the partial orderings from $\sqsubseteq$ and $\sqsubseteq_e$.

► *Example 4.3.* Consider the same program $\mathcal{Q}$ as in Example 4.2, repeated here for convenience:

$$\mathcal{Q} := \left\{ \begin{array}{r} \mathtt{p} \leftarrow \mathtt{a}\ ,\ \mathtt{b} \\ \mathtt{q} \leftarrow \mathtt{a}\ ,\ \mathtt{b} \\ \mathtt{a} \leftarrow \mathtt{d}\ ,\ \mathtt{c} \\ \mathtt{b} \leftarrow \phantom{\mathtt{a}\ ,\ \mathtt{b}} \\ \mathtt{c} \vee \mathtt{d} \leftarrow \mathtt{b} \phantom{\ ,\ \mathtt{b}} \end{array} \right\}$$

(the goal is still $\leftarrow \mathtt{p} \vee \mathtt{q}$). The two plays that follow correspond to the ones we considered previously. Here, the statements of the believer moves that use more than one rule are explicitly shown in parentheses merely for the convenience of the reader; they are not part of the actual plays.

$$\pi_1 := \left| \begin{array}{rl} \mathsf{goal}: & \leftarrow \underline{\mathtt{p} \vee \mathtt{q}} \\ \beta_0: & \mathtt{p} \leftarrow \underline{\mathtt{a}}\ ,\ \mathtt{b} \\ & \mathtt{q} \leftarrow \mathtt{a}\ ,\ \underline{\mathtt{b}} \\ \big([\beta_0]: & \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a} \vee \mathtt{a}\ ,\ \underline{\mathtt{a} \vee \mathtt{b}}\ ,\ \mathtt{b} \vee \mathtt{a}\ ,\ \mathtt{b} \vee \mathtt{b}\big) \\ \beta_1: & \mathtt{b} \leftarrow \end{array} \right|$$

is (still) won by Believer, and

$$\pi_2 := \left| \begin{array}{rl} \mathsf{goal}: & \leftarrow \underline{\mathtt{p} \vee \mathtt{q}} \\ \beta_0: & \mathtt{p} \leftarrow \underline{\mathtt{a}}\ ,\ \mathtt{b} \\ & \mathtt{q} \leftarrow \underline{\mathtt{a}}\ ,\ \mathtt{b} \\ \big([\beta_0]: & \mathtt{p} \vee \mathtt{q} \leftarrow \underline{\mathtt{a} \vee \mathtt{a}}\ ,\ \mathtt{a} \vee \mathtt{b}\ ,\ \mathtt{b} \vee \mathtt{a}\ ,\ \mathtt{b} \vee \mathtt{b}\big) \\ \beta_1: & \mathtt{a} \leftarrow \mathtt{d}\ ,\ \underline{\mathtt{c}} \end{array} \right|$$

by Doubter. ◄

Property 4.1. *Any dialogue $\pi := \langle \delta_0, \beta_0, \dots \rangle$ from $\mathcal{P}$ can be considered as a play of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, where $\mathtt{G} := [\,\delta_0\,]$.*

Remark 4.2 (Disallowing stalling). We have forced the believer to always include at least one rule from the actual program $\mathcal{P}$, thus banning what we are about to call "stalling" from our games. Still, this concept will be crucial for our exposition: the introduction of plays in which stalling is allowed (i.e., quasi-plays) will make a lot of the statements that follow easier to prove.

Definition 4.5 (Follow)**.** We say that Doubter *follows*, if she has only one possible valid move that she can play in response to a believer move β, i.e., if for every rule $\psi \in \beta$, $|\mathsf{body}(\psi)| = 1$. We denote such a follow move by $\overline{\beta}$. In symbols,

$$\overline{\beta} \overset{\mathsf{df}}{=} \langle \text{the unique } \mathtt{D} \in \mathsf{body}(\psi) \mid \psi \in \beta \rangle\,.$$

Definition 4.6 (Stalling)**.** We say that a Believer is *stalling*, if he responds to a doubter move δ by playing the sequence of implicit rules

$$\overline{\delta} \overset{\mathsf{df}}{=} \langle \mathtt{a} \leftarrow \mathtt{a} \mid a \in [\,\delta\,] \rangle\,.$$

In this way he is forcing the doubter to follow with $\overline{\overline{\delta}}$, which has the same doubts as δ again. Notice that despite the fact that the occurrences will be different, the actual doubts will be the same, which is what really matters here. Easily, stalling is a valid response to δ since $\mathsf{head}\big([\,\overline{\delta}\,]\big) = [\,\delta\,]$.

Remark 4.3 (Notation). Once more we have overloaded a symbol here, but once more the end justifies the means: we increase readability with no possibility of ambiguity, since the follow-bar can only be applied to believer moves, while the stalling-bar only covers doubter moves. This also brings the handy double-bar notation for the only possible reply to a stalling move, in which case the follow is always defined. In addition, it is easy to verify the following cute, bar-cancellation properties:

$$\overline{\overline{\overline{\delta}}} = \overline{\delta} \qquad \text{and} \qquad \overline{\overline{\overline{\beta}}} = \overline{\beta}.$$

Remark 4.4. For any doubter move δ, the move $\overline{\overline{\delta}}$ is equal to δ modulo a change in occurrences: it contains exactly the same doubts. Therefore, it also shares the same statement

$$[\,\delta\,] = \big[\,\overline{\overline{\delta}}\,\big]\,.$$

If we remove stallings and their corresponding follow moves from a quasidialogue, we obtain an actual dialogue; and similar for plays. This is exactly what the function $\mathsf{rmstall}$ (defined below) does; colloquially speaking, it removes the "quasi-". Note that if the original quasidialogue ended with a stall move, the resulting dialogue will be of odd length.

Definition 4.7 ($\mathsf{rmstall}$)**.** Let τ be a quasidialogue from $\mathcal{P}$. We define the function $\mathsf{rmstall}$ recursively by cases on $\ell := |\tau|$:

CASE 1: $\ell \leq 1$. Then either $\tau = \langle\,\rangle$ or $\tau = \langle \delta \rangle$ for some doubter move δ. Both alternatives contain no believer moves, and hence zero stallings; we set

$$\mathsf{rmstall}(\tau) \overset{\mathsf{df}}{=} \tau.$$

CASE 2: $\ell = 2$. Then $\tau := \langle \delta, \beta \rangle$, so we set

$$\mathsf{rmstall}(\tau) \overset{\mathsf{df}}{=} \begin{cases} \langle \delta \rangle & \text{if } \beta = \overline{\delta}, \\ \langle \delta, \beta \rangle & \text{otherwise.} \end{cases}$$

CASE 3: $\ell \geq 3$. Then $\tau = \langle \delta, \beta, \delta' \rangle +\!\!+ \tau'$, and we need to recurse:

$$\mathsf{rmstall}(\tau) \overset{\mathsf{df}}{=} \begin{cases} \mathsf{rmstall}(\delta :: \tau') & \text{if } \beta = \overline{\delta}, \\ \langle \delta, \beta \rangle +\!\!+ \mathsf{rmstall}(\delta' :: \tau') & \text{otherwise.} \end{cases}$$

Property 4.2 (Validity of $\mathsf{rmstall}$)**.** *If τ is a quasidialogue from a program, then* $\mathsf{rmstall}(\tau)$ *is a dialogue from the same program; and if π is a quasiplay in some game, then* $\mathsf{rmstall}(\pi)$ *is a play in the same game. In addition,* $\mathsf{rmstall}$ *leaves dialogues and plays intact:*

$$\pi \textit{ dialogue or play} \implies \mathsf{rmstall}(\pi) = \pi$$

Property 4.3 ($\mathsf{rmstall}$ is $\sqsubseteq$-monotone)**.**

$$\tau \sqsubseteq \tau' \implies \mathsf{rmstall}(\tau) \sqsubseteq \mathsf{rmstall}(\tau').$$

We now define the important notion of a strategy. Roughly speaking, we want a strategy to dictate how a believer should play against a doubter. If it covers all potential doubter moves, we will call it total, and if it always leads to victory, winning. Formally, we define:

Definition 4.8 (Strategy)**.** A *strategy* σ in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ is a set of plays in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, such that:

(i) $\sigma \neq \emptyset$;

(ii) every play in σ has even length;

(iii) σ is closed under even prefixes (and therefore always contains ε):

$$\pi' \sqsubseteq_{\mathrm{e}} \pi \in \sigma \implies \pi' \in \sigma;$$

(iv) σ is deterministic: if $\pi \in \sigma$ and $\pi +\!\!+ \langle \delta \rangle$ is a play, then there exists *at most one* move β such that $\pi +\!\!+ \langle \delta, \beta \rangle \in \sigma$.

We will call a strategy *combo-free* if none of its plays contains combo moves; in symbols, $\beta \in \pi \in \sigma \implies |\beta| = 1$.

REMARK 4.5 (Strategies as posets). A strategy σ can be seen as a *poset* $(\sigma, \sqsubseteq_{\mathrm{e}})$ of even-length plays with a bottom element $\bot_\sigma = \varepsilon$.

Definition 4.9 (Total strategy)**.** A strategy σ is called *total*, if for every play $\pi \in \sigma$ and every doubter response to it δ, there is *at least one* move β such that $\pi \mathbin{+\!\!+} \langle \delta, \beta \rangle$ is also in σ.

Siding with Believer, we give the following definition:

Definition 4.10 (Winning strategy)**.** A strategy σ is called *winning*, if it is total and finite.

Remark 4.6 (Infinite plays and limits). Let us pretend awhile that dialogues (and plays) might be infinite. We should then change (among other things) the definition of strategy to demand that it is also closed under limits—or, should we? If we do so, nothing essential will change, as a correspondence between such strategies and the ones we are using here is obvious:

- starting from a strategy with infinite plays, simply remove them—all their finite, even prefixes are already included;
- starting from a strategy with only finite plays, add all limits (i.e., lubs of $\sqsubseteq_{\mathrm{e}}$-chains).

On the other hand, if we do not close under limits, we will end up distinguishing between strategies like σ and σ_∞, where:

$$\begin{aligned} \sigma &:= \{\langle \delta_0, \beta_0, \dots, \delta_n, \beta_n \rangle \mid n \in \omega\}\,, \\ \sigma_\infty &:= \sigma \cup \{\textstyle\bigsqcup \sigma\}\,. \end{aligned}$$

Such a distinction could potentially be useful for some kind of ordinal extension of games. But for reasons of elegance and simplicity we have chosen not to deal with infinite dialogues altogether, as they are not needed for the development of this game semantics of disjunctive logic programs.

Definition 4.11 (answer_σ)**.** Given a play π and a strategy σ in some game $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, the play $\mathsf{answer}_\sigma(\pi) \in \sigma$ will be:

$$\mathsf{answer}_\sigma(\pi) \overset{\mathsf{df}}{=} \begin{cases} \pi & \text{if } \pi \in \sigma, \\ \pi \mathbin{+\!\!+} \langle \beta \rangle & \text{if there exists a } \beta \text{ such that } \pi \mathbin{+\!\!+} \langle \beta \rangle \in \sigma, \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

This is a well-defined partial function because σ is deterministic, and so in the second branch, if such a β exists, it is necessarily unique. Furthermore, if σ is total, $\mathsf{answer}_\sigma(\pi)$ is undefined iff $\pi^- \notin \sigma$. Hence, if we stick with a total strategy, it will always provide us with a next move, an answer to any doubter move.

Property 4.4. *The function* answer_σ *behaves like a* closure operator*; i.e., whenever it is defined throughout the following statements, it satisfies them:*

$$\begin{aligned} \pi &\sqsubseteq \mathsf{answer}_\sigma(\pi) && \text{(extensive)}, \\ \pi \sqsubseteq \pi' \implies \mathsf{answer}_\sigma(\pi) &\sqsubseteq \mathsf{answer}_\sigma(\pi') && \text{(monotone)}, \\ \mathsf{answer}_\sigma(\mathsf{answer}_\sigma(\pi)) &= \mathsf{answer}_\sigma(\pi) && \text{(idempotent)}. \end{aligned}$$

The DLP game we have investigated is in a sense equivalent to the simplified one that we described previously. This is a consequence of the logical equivalence $\mathscr{D} \curlyvee \mathscr{E} \equiv \mathscr{D} \vee \mathscr{E}$ and of the following proposition:

Proposition 4.5. *Let $\mathscr{D}$, $\mathscr{E}$, and $\mathscr{F}$ be DLP conjunctions such that $\mathscr{F} \equiv \mathscr{D} \vee \mathscr{E}$. Then for every disjunction $F \in \mathscr{F}$ there is some disjunction $D_F \in \mathscr{D}$ and some disjunction $E_F \in \mathscr{E}$ such that $D_F \cup E_F \subseteq F$.*

Proof. Pick any disjunction F of $\mathscr{F}$. Consider the interpretation $\alpha = \mathcal{A} \setminus F$. By definition, an assignment that makes F false, must falsify $\mathscr{F}$ as well. But $\mathscr{F}$ is logically equivalent to $\mathscr{D} \vee \mathscr{E}$, which means that both $\mathscr{D}$ and $\mathscr{E}$ are false under α. Since they are both conjunctions, there is at least one element $D_F \in \mathscr{D}$ that is false, and similarly for an element $E_F \in \mathscr{E}$. ▌

We have exposed every little detail that we will need in order to develop the DLP game semantics in a formal mathematical setting. Let us do so.

4.4. Game semantics

Remember—our objective is to use games to provide denotational semantics for disjunctive logic programs; in other words, to decide which goals should succeed, i.e., which answers are correct. However, depending on the cleverness of the players involved, for the same program $\mathcal{P}$, and the same goal $\leftarrow \mathtt{G}$, Believer might win one time, yet lose another. It therefore makes no sense to determine the success of a goal by looking at a single play. Instead, we say:

Definition 4.12 (DLP game semantics)**.** Let $\mathcal{P}$ be a DLP program. A goal $\leftarrow \mathtt{G}$ *succeeds*, if Believer has a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.

This is essentially the same definition as the LP game semantics.

Remark 4.7 (Benefit of the doubt). Why do we give the benefit of the doubt to the doubter? Notice at once that if stalling was allowed, Believer would have a winning strategy of "forever stalling", for any goal. This would make every disjunction derivable! But even with stalling forbidden, things can still go wrong even without disjunctions:

► *Example 4.4.* Consider the following program

$$\mathcal{P} := \begin{Bmatrix} \mathtt{a} \leftarrow \mathtt{a}\ ,\ \mathtt{b} \\ \mathtt{b} \leftarrow \end{Bmatrix}$$

and the goal $\leftarrow \mathtt{a}$. A play in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{a})$ has to begin with Doubter doubting $\mathtt{a}$, to which Believer responds with $\mathtt{a} \leftarrow \mathtt{a}\ ,\ \mathtt{b}$. For as long as Doubter doubts $\mathtt{a}$, Believer always plays the same rule, and therefore plays will never end with a winner. Had we given the benefit of the doubt to Believer, Doubter would eventually have no choice but to switch and doubt $\mathtt{b}$, at which point Believer would win by playing the fact $\mathtt{b} \leftarrow$. This describes a winning strategy for Believer, but—this being a logic program—we most definitely do not want the goal $\leftarrow \mathtt{a}$ to succeed, because of the denotational semantics:

$$a \notin \mathrm{LHM}(\mathcal{P}) = \{b\}\,. \qquad ◄$$

Who has the benefit of the doubt is a decision similar to the closed/open world assumptions (CWA/OWA) in knowledge representation languages. Giving it to the doubter resembles CWA, while giving it to no player resembles OWA;[12] it might be desired in some cases (e.g., description logics used for the semantic web), but it is certainly not the stance we take in the world of logic programming. For more information check [33] and [28].

Remark 4.8 (Backwards compatibility). In the model-theoretic side of semantics, extensions of the language are backwards compatible: for DLP, a program without disjunctions has a unique minimal model, viz. the least Herbrand model; for LPN, the well-founded model of a program without negations is two-valued and coincides with the least Herbrand model as well. Similar compatibilities are desired and achieved in the game-theoretic side. We highlight that strictly speaking, the DLP game is not *directly* compatible with the LP game in the sense that it does not reduce to it when it is played on an LP program. The reason behind this is that in the DLP game, believers can play what we have called combo moves.[13] Nevertheless, we will prove in Lemma 6.3 that whenever there is a winning strategy in the DLP game of an LP program, then there is a winning strategy in which the believer never plays more than one rule, and is thus compatible with the LP game. In other words, the extra "combo-rule" of the DLP game is unnecessary in the absence of disjunctions.

Earlier, we explained that we will need to combine, restrict, and split plays and strategies. This is what we do next. Besides, it has been a while since we last overloaded the $\curlyvee$ and $|$ symbols.

5. Plays and strategies

In this section, we define the combination and splitting of both plays and strategies. The main point here is that these constructions preserve all the properties that we need.

Further notational conventions. To avoid tedious repetitions in what follows, we hereby agree that $\mathcal{P}$ will always be a proper DLP program, $\leftarrow \mathtt{G}$ a goal clause, and ϕ a proper DLP rule of $\mathcal{P}$; $\mathcal{H} := (H_1, H_2)$ will be a proper partition of $H := \mathsf{head}(\phi)$, and $(\mathcal{P}_1, \mathcal{P}_2)$ the corresponding splitting of $\mathcal{P}$ with respect to ϕ over $\mathcal{H}$. Naturally we will write just ϕ_1 and ϕ_2 for the rules $\phi|_{H_1}$ and $\phi|_{H_2}$ respectively. The variable q ranges over the size of the partition: $q = 1, 2$.[14] We

[12]This would introduce strategies that are neither winning nor losing: they lead to *ties*.

[13]The reader may contrast this with the LPN game, which is directly compatible with the LP game when no negations are present: the additional rule of LPN (rôle-switch) can only be used by the believer, and only as an answer to a negative doubt. This cannot happen in an LP program.

[14]There is nothing special about the number 2 here, and everything that we develop can be stated *mutatis mutandis* for the case in which $|\mathcal{H}|$ is any larger number. However, 2 is as large as we need.

use π and π_q for dialogues or plays of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$ respectively; τ and τ_q for "quasi-". We remind the reader that we solely use β and δ for believer and doubter moves respectively, and that strategies are usually denoted by σ. In this setting, with this notation, we proceed to define combination, restriction, and splitting of both plays and strategies.

5.1. Plays

Here we show how we can combine arbitrary plays from games of the splitting of a program to create a valid play in the original program's game. The construction we use is slightly technical but hopefully the intuition behind it will be apparent to the reader, who will then have no problem appreciating and accepting the details. We will also see how to work in the opposite direction: starting from a play of a game of the combined program we can split it into two plays, valid in the games of the less disjunctive, restricted programs.

5.1.1. Combining plays

As we are about to witness, while combining plays special care must be taken because a believer move of a restricted play $\mathcal{P}_q$ might not be valid in $\mathcal{P}$:

Definition 5.1 (Forbidden move)**.** A believer move β of $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$ is called *forbidden* in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ iff it includes ϕ_q. In symbols,

$$\mathrm{Forbidden}_q(\beta) \overset{\mathsf{df}}{\iff} \phi_q \in \beta.$$

Definition 5.2 (Release)**.** Given a believer move β_i^q of $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$, we define β_i^{q*} to be β_i^q after replacing instances of the forbidden rule ϕ_q by ϕ, so that it becomes valid in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. We call β_i^{q*} the *release* of β_i^q from H_q to H.

Property 5.1. *The release of a move satisfies the following properties:*

(i) $\beta_i^{q*} = \beta_i^q \iff \phi_q \notin \beta_i^q$;

(ii) $\mathsf{body}\big(\beta_i^{q*}\big) = \mathsf{body}(\beta_i^q)$.

In order to justify believer moves in the combined play, we will need the following proposition:

Proposition 5.2. *Let β_1 and β_2 be two believer moves from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and let $\beta \coloneqq {\beta_1}^* +\!\!+ {\beta_2}^*$. Then*

$$\mathsf{head}([\,\beta\,]) = \mathsf{head}([\,\beta_1\,]) \cup \mathsf{head}([\,\beta_2\,]).$$

Proof. We compute

$$\begin{aligned}
\mathsf{head}([\,\beta_1\,]) \cup \mathsf{head}([\,\beta_2\,]) &= \Big(\bigcup\nolimits_{\psi\in\beta_1} \mathsf{head}(\psi)\Big) \cup \Big(\bigcup\nolimits_{\psi\in\beta_2} \mathsf{head}(\psi)\Big)\\
&= \bigcup\nolimits_{\psi\in\beta_1 +\!\!+ \beta_2} \mathsf{head}(\psi)\\
&= \mathsf{head}([\,\beta_1 +\!\!+ \beta_2\,])\\
&\overset{*}{=} \mathsf{head}([\,{\beta_1}^* +\!\!+ {\beta_2}^*\,])\\
&= \mathsf{head}([\,\beta\,]),
\end{aligned}$$

where the starred equality holds since

$$\mathsf{head}(\phi_1) \cup \mathsf{head}(\phi_2) = H_1 \cup H_2 = \mathsf{head}(\phi). \qquad \blacksquare$$

First we define combination for what we call synchronous plays, as it is a lot simpler. Once we have seen how to combine such plays, we proceed to give the most general definition covering arbitrary plays.

Combining synchronous plays

The three games $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$, and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ would be identical except that ϕ can only be part of believer moves of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, ϕ_1 of those of $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$, and ϕ_2 of those of $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$. Since moves are sequences, it would be delightful to simply concatenate the moves of π_1 with those of π_2 to obtain a play in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. Doubter moves are no obstacles to this plan, but believer moves can be troublesome: they may contain the rules ϕ_q, which are not allowed in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. This problem is easy to solve if *both* moves include their forbidden rules: we simply replace each of them by ϕ—a reasonable action, since $\phi \equiv \phi_1 \curlyvee \phi_2$. To deal with this case, we begin with a key definition.

Definition 5.3 (Synchronous)**.** Two quasiplays τ_1 and τ_2 in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively are called *synchronous* (or *in sync with* each other) if both Believers use their forbidden rules in the exact same turns. In symbols,

$$\text{for all } i \leq \min\{|\tau_1|, |\tau_2|\}, \qquad \phi_1 \in \beta_i^1 \iff \phi_2 \in \beta_i^2.$$

Property 5.3. *If τ_1 is in sync with τ_2, then so is any prefix of τ_1 with any prefix of τ_2.*

Let π_1 and π_2 be plays of even length in the games $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively, and suppose that the two plays are synchronous. We will describe a new play $\pi_1 \ddot{\curlyvee} \pi_2$: the *synchronous combination of π_1 and π_2 with respect to ϕ over $\mathcal{H}$.* To better understand this construction, we first present the idea informally, building the combined play turn by turn. We do so as follows: let

$$\pi_1 \coloneqq \left\langle \delta_0^1, \beta_0^1, \delta_1^1, \beta_1^1, \dots \right\rangle$$
$$\pi_2 \coloneqq \left\langle \delta_0^2, \beta_0^2, \delta_1^2, \beta_1^2, \dots \right\rangle$$

be the two given plays. We set

$$\pi_1 \ddot{\curlyvee} \pi_2 \overset{\mathsf{df}}{=} \langle \delta_0, \beta_0, \delta_1, \beta_1, \dots \rangle,$$

where the symbols are defined as follows.

The first move is essentially determined by the game: Doubter starts with

$$\delta_0 \coloneqq \delta_0^1 +\!\!+ \delta_0^2;$$

Assuming that $\phi_i \notin \beta_0^i$, Believer replies with

$$\beta_0 \coloneqq \beta_0^1 +\!\!+ \beta_0^2,$$

a valid move since $\mathsf{head}([\,\beta_0\,]) \subseteq [\,\delta_0\,]$ (Property 5.1, Proposition 5.2). Doubter must now select one occurrence from each rule-body in β_0, and the moves δ_1^1 and δ_1^2 provide just that:

$$\delta_1 \coloneqq \delta_1^1 +\!\!+\, \delta_1^2.$$

The game goes on until the turn i in which the believers of π_1 and π_2 both play their forbidden rules ϕ_1 and ϕ_2 in their justifications. At this point, it is of course Believer's turn in the combined play $\pi_1 \ddot{\curlyvee} \pi_2$. We set his move to be

$$\beta_i \coloneqq {\beta_i^1}^* +\!\!+\, {\beta_i^2}^*,$$

which is valid in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. Doubter's turn: δ_{i+1}^1 and δ_{i+1}^2 consist of occurrences from the rule-bodies of β_i^1 and β_i^2 respectively. Since releasing only affects heads (Property 5.1(ii)), all occurrences in the rule-bodies of β_i^q are occurrences in the rule-bodies of ${\beta_i^q}^*$ as well. Therefore, she can copy the selections of δ_{i+1}^1 and δ_{i+1}^2 by playing

$$\delta_{i+1} \coloneqq \delta_{i+1}^1 +\!\!+\, \delta_{i+1}^2.$$

She does so, and the game goes on until we run out of moves to combine.

Remark 5.1. We have cheated a bit, since we took for granted that the doubter of $\pi_1 \ddot{\curlyvee} \pi_2$ has in her possession *two* doubter moves δ_k^1 and δ_k^2 from π_1 and π_2 respectively. This will not hold if the plays have unequal lengths; in this case, the combined play ends as soon as the shortest play ends.

We arrive at the following definition, general enough to handle quasiplays:

Definition 5.4 (Synchronous combination)**.** Given two synchronous quasiplays

$$\begin{aligned}\tau_1 &\coloneqq \left\langle \delta_0^1, \beta_0^1, \delta_1^1, \beta_1^1, \dots \right\rangle \text{ of } \Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G}),\\ \tau_2 &\coloneqq \left\langle \delta_0^2, \beta_0^2, \delta_1^2, \beta_1^2, \dots \right\rangle \text{ of } \Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G}),\end{aligned}$$

their *synchronous combination* $\tau_1 \ddot{\curlyvee} \tau_2$ over ϕ with respect to $\mathcal{H}$ is the sequence

$$\tau_1 \ddot{\curlyvee} \tau_2 \overset{\mathsf{df}}{=} \langle \delta_0, \beta_0, \delta_1, \beta_1, \dots \rangle$$

of length $|\tau_1 \ddot{\curlyvee} \tau_2| = \min\{|\tau_1|, |\tau_2|\}$, where the symbols involved are defined by

$$\begin{aligned}\delta_i &\coloneqq \delta_i^1 +\!\!+\, \delta_i^2,\\ \beta_i &\coloneqq {\beta_i^1}^* +\!\!+\, {\beta_i^2}^*.\end{aligned}$$

Proposition 5.4 (Validity of $\ddot{\curlyvee}$)**.** *Given two synchronous quasiplays τ_1 and τ_2 of $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively, $\tau \coloneqq \tau_1 \ddot{\curlyvee} \tau_2$ is a quasiplay in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. It follows that if τ_1 and τ_2 do not stall simultaneously, τ will be a play of the game $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.*

Proof. First, by the definitions of δ_i and β_i, it is immediate that they are doubter and believer moves respectively. Since each quasiplay τ_q is valid in $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$, we know that

$$\mathsf{head}\big([\,\beta_i^1\,]\big) \subseteq [\,\delta_i^1\,]\,,$$
$$\mathsf{head}\big([\,\beta_i^2\,]\big) \subseteq [\,\delta_i^2\,]\,,$$

and so by taking unions on both sides and by using Proposition 5.2 we obtain

$$\mathsf{head}([\,\beta_i\,]) = \mathsf{head}\big([\,\beta_i^1\,]\big) \cup \mathsf{head}\big([\,\beta_i^2\,]\big) \subseteq [\,\delta_i^1\,] \cup [\,\delta_i^2\,] = [\,\delta_i^1 \mathbin{+\!\!+} \delta_i^2\,] = [\,\delta_i\,]\,,$$

which validates every believer move. Doubter moves are justified by the definition of release (which leaves bodies intact) as explained earlier in the sketchy description of play combination (p. 28). To verify that τ is indeed a quasiplay, observe that both τ_1 and τ_2 share the same goal with τ, and so $\delta_0 = \delta_0^1 \mathbin{+\!\!+} \delta_0^2$ is a correct first move for a quasidialogue from $\mathcal{P}$ to be a quasiplay of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.

For the second claim, observe that a stalling move in $\tau_1 \ddot{\curlyvee} \tau_2$ would imply the existence of two simultaneously played stalling moves, one in τ_1 and one in τ_2, against the hypothesis. ▮

As plays never stall, we get the following property as a corollary:

Corollary 5.5 (Preservation of plays)**.** *The synchronous combination of two synchronous plays is a play.*

The definition of $\ddot{\curlyvee}$ immediately yields a couple of more preservation properties:

Property 5.6 (Preservation of parity)**.** *Let τ_1 and τ_2 be two synchronous quasiplays, both of even length. Then $\tau_1 \ddot{\curlyvee} \tau_2$ will also have even length.*

Property 5.7 ($\ddot{\curlyvee}$ is monotone)**.** *Let τ_1 and τ_2 be two synchronous quasiplays. Then for any $\tau_1' \sqsubseteq \tau_1$ and any $\tau_2' \sqsubseteq \tau_2$ we have $\tau_1' \ddot{\curlyvee} \tau_2' \sqsubseteq \tau_1 \ddot{\curlyvee} \tau_2$.*

So far, so good. Not every pair of plays is synchronous though, which brings us to the next topic.

Combining arbitrary plays

Starting with two arbitrary quasiplays τ_1 and τ_2, we build two new, synchronized quasiplays $\dot{\tau}_1$ and $\dot{\tau}_2$ by inserting pairs of stall–follow moves on the turns in which one believer uses their forbidden rule but the other does not, leaving the rest of the moves intact. Now we can simply combine $\dot{\tau}_1$ with $\dot{\tau}_2$. It is exactly this idea that we exploit to extend the definition of $\tau_1 \ddot{\curlyvee} \tau_2$ to cover asynchronous plays: *synchronize first, then combine.*

Definition 5.5 (Synchronization)**.** Let τ_1 and τ_2 be two quasidialogues from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and let ℓ_1 and ℓ_2 be their lengths. We recursively define their *synchronization* $\mathsf{sync}(\tau_1, \tau_2)$ by cases depending on $\ell := \min\{\ell_1, \ell_2\}$.

Case 1: $\ell < 2$. In this case, there are no believer moves at all; and they are the only ones that can cause asynchronicity. Hence, any such pair is trivially synchronous:

$$\mathsf{sync}(\tau_1, \tau_2) \overset{\mathsf{df}}{=} (\tau_1, \tau_2)\,.$$

Case 2: $\ell \geq 2$. Then $\tau_1 \coloneqq \langle \delta_1, \beta_1 \rangle \mathbin{+\!\!+} \tau_1'$ and $\tau_2 \coloneqq \langle \delta_2, \beta_2 \rangle \mathbin{+\!\!+} \tau_2'$, and we set:

$$\mathsf{sync}(\tau_1, \tau_2) \overset{\mathsf{df}}{=} \begin{cases} \big(\langle \delta_1, \overline{\delta_1} \rangle \mathbin{+\!\!+} \mathsf{rec}_1,\ \langle \delta_2, \beta_2 \rangle \mathbin{+\!\!+} \mathsf{rec}_2\big) & \text{if (a)}, \\ \big(\langle \delta_1, \beta_1 \rangle \mathbin{+\!\!+} \mathsf{rec}_1,\ \langle \delta_2, \overline{\delta_2} \rangle \mathbin{+\!\!+} \mathsf{rec}_2\big) & \text{if (b)}, \\ \big(\langle \delta_1, \beta_1 \rangle \mathbin{+\!\!+} \mathsf{rec}_1,\ \langle \delta_2, \beta_2 \rangle \mathbin{+\!\!+} \mathsf{rec}_2\big) & \text{if (c)}, \end{cases}$$

where rec_1 and rec_2 wrap the recursive calls

$$(\mathsf{rec}_1, \mathsf{rec}_2) \coloneqq \begin{cases} \mathsf{sync}\Big(\overline{\overline{\delta_1}} :: \beta_1 :: \tau_1', \tau_2'\Big) & \text{if (a)}, \\ \mathsf{sync}\Big(\tau_1', \overline{\overline{\delta_2}} :: \beta_2 :: \tau_2'\Big) & \text{if (b)}, \\ \mathsf{sync}(\tau_1', \tau_2') & \text{if (c)}, \end{cases}$$

all according to the subcases: (a) $\phi_1 \in \beta_1$ and $\phi_2 \notin \beta_2$; (b) $\phi_1 \notin \beta_1$ and $\phi_2 \in \beta_2$; (c) otherwise.

Remark 5.2 (Dependencies). Even though we have not incorporated ϕ_1 and ϕ_2 into the symbol sync of synchronization, we stress that it does, in fact, depend on both of those rules: it uses them to determine the (a)–(c). Since we have fixed ϕ, ϕ_1, and ϕ_2 into our notation, however, we allow ourselves to simply use sync instead of an excessively precise name like $\mathsf{sync}_{\phi_1,\phi_2}$. The same is true for various symbols that we use, such as $\curlyvee$ and *.

Proposition 5.8. *Synchronization is a well-defined, total operation.*

Proof. Observe that on every recursive call of sync both of its arguments are themselves quasidialogues from $\mathcal{P}_1$ and $\mathcal{P}_2$; and so it makes sense to recurse on them. Since sync is defined by recursion, we must be careful to ensure that it terminates on every input. Note that on every recursive call of case 2 ($\ell \geq 2$), the sum of the lengths of the arguments decreases: by 2 in subcases (a)–(b), and by 4 in (c). Eventually, at least one of them will become short enough and sync will reach case 1 ($\ell < 2$), which contains no recursive calls. ▮

One can easily verify that synchronization behaves as expected:

Property 5.9 (Validity of synchronization)**.** *Let* $(\dot{\tau_1}, \dot{\tau_2}) \coloneqq \mathsf{sync}(\tau_1, \tau_2)$. *Then*

(i) *the pair* $(\dot{\tau_1}, \dot{\tau_2})$ *is synchronous;*

(ii) *if* (τ_1, τ_2) *happens to be synchronous, then* $\mathsf{sync}(\tau_1, \tau_2) = (\tau_1, \tau_2)$*;*

(iii) $\mathsf{rmstall}(\dot{\tau_q}) = \mathsf{rmstall}(\tau_q)$*;*

(iv) sync *never produces two stalling moves in the same turn.*

Proposition 5.10 (Preservation of parity)**.** *If τ_1 and τ_2 have even lengths, then so do the elements of* $\mathsf{sync}(\tau_1, \tau_2)$*.*

Proof. This is immediate by the very definition of sync, which generates its output $\mathsf{sync}(\tau_1, \tau_2)$ incrementally by pairs of moves, while consuming pairs of moves from its inputs τ_1 and τ_2. Thus, if both of its inputs are of even length, the same will be true for its outputs. ∎

Proposition 5.11 (sync is monotone)**.** *Let τ_1' and τ_2' be two quasidialogues, and let $\tau_1 \sqsubseteq \tau_1'$ and $\tau_2 \sqsubseteq \tau_2'$. Then $\mathsf{sync}(\tau_1, \tau_2) \sqsubseteq \mathsf{sync}(\tau_1', \tau_2')$. Moreover, if $\tau_1 \sqsubset \tau_1'$ and $\tau_2 \sqsubset \tau_2'$, then $\mathsf{sync}(\tau_1, \tau_2) \sqsubset \mathsf{sync}(\tau_1', \tau_2')$.*

Proof. Just like in the previous proof, we only need to observe how synchronization really works. In fact, the construction of $\mathsf{sync}(\tau_1, \tau_2)$ must end with a call to case 1 of its definition, with either $\ell = 0$ or $\ell = 1$. In either case, extending any of τ_1 or τ_2 results in an extension of the corresponding synchronized quasidialogue. ∎

We know how to combine synchronous plays; and we know how to synchronize asynchronous plays. Compose the two and behold: a method to combine arbitrary plays. Just as in the synchronous case, we state the definition in its most general form, which is able to handle quasidialogues.

Definition 5.6 (Combination of quasidialogues)**.** Let τ_1 and τ_2 be two quasidialogues from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. We define their *combination* $\tau_1 \curlyvee \tau_2$ by composition:

$$\curlyvee \overset{\mathsf{df}}{=} \ddot{\curlyvee} \circ \mathsf{sync}.$$

In other words,

$$\tau_1 \curlyvee \tau_2 \overset{\mathsf{df}}{=} \dot{\tau_1} \ddot{\curlyvee} \dot{\tau_2},$$

where $(\dot{\tau_1}, \dot{\tau_2}) := \mathsf{sync}(\tau_1, \tau_2)$.

Remark 5.3. By Property 5.9(ii), $\curlyvee$ is compatible with $\ddot{\curlyvee}$, in the sense that it yields the same output in case its input is synchronous.

Remark 5.4 (End of $\tau_1 \curlyvee \tau_2$). According to the definitions of synchronization and syncronous combination, the combined quasidialogue $\tau_1 \curlyvee \tau_2$ ends exactly when we reach the final move of either τ_1 or τ_2 (whichever comes first).

By defining the general combination as a composition of sync and $\ddot{\curlyvee}$, we readily get their composable properties as corollaries:

Corollary 5.12 (Validity of combination)**.** *Given two (quasi)dialogues τ_1 and τ_2 from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, their disjunctive combination $\tau_1 \curlyvee \tau_2$ is a (quasi)-dialogue from $\mathcal{P}$.*

Corollary 5.13 (Preservation of plays)**.** *If τ_1 and τ_2 are two (quasi)plays in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively, then $\tau_1 \curlyvee \tau_2$ is a (quasi)play in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.*

Corollary 5.14 (Preservation of parity). *If π_1 and π_2 have even length, then so does $\pi_1 \curlyvee \pi_2$.*

Corollary 5.15 ($\curlyvee$ is monotone). *Let π_1' and π_2' be two dialogues, and let $\pi_1 \sqsubseteq \pi_1'$ and $\pi_2 \sqsubseteq \pi_2'$. Then $\pi_1 \curlyvee \pi_2 \sqsubseteq \pi_1' \curlyvee \pi_2'$. Moreover, if $\pi_1 \sqsubset \pi_1'$ and $\pi_2 \sqsubset \pi_2'$, then $\pi_1 \curlyvee \pi_2 \sqsubset \pi_1' \curlyvee \pi_2'$.*

Remark 5.5 (Losing is not preserved). Even though Doubter may be victorious in two plays, in the combined version she might not be so. As a counterexample, consider the following program and splitting, in which Doubter wins both π_1 and π_2 (i.e., Believer cannot play a valid response to either plays) and yet she does not win in the combined play:

$$\mathcal{P} := \left\{ \begin{array}{r} \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{a} \vee \mathtt{b} \leftarrow \phantom{\mathtt{a}} \\ \mathtt{c} \vee \mathtt{d} \leftarrow \phantom{\mathtt{a}} \end{array} \right\} \rightsquigarrow \left(\mathcal{P}_1 := \left\{ \begin{array}{r} \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{a} \vee \mathtt{b} \leftarrow \phantom{\mathtt{a}} \\ \mathtt{c} \leftarrow \phantom{\mathtt{a}} \end{array} \right\}, \ \mathcal{P}_2 := \left\{ \begin{array}{r} \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{a} \vee \mathtt{b} \leftarrow \phantom{\mathtt{a}} \\ \mathtt{d} \leftarrow \phantom{\mathtt{a}} \end{array} \right\} \right)$$

and the plays

$$\underbrace{\left| \begin{array}{rr} \mathsf{goal}: & \leftarrow \underline{\mathtt{p}} \\ \beta_0^1: & \mathtt{p} \leftarrow \underline{\mathtt{a}} \end{array} \right|}_{\pi_1} \curlyvee \underbrace{\left| \begin{array}{rr} \mathsf{goal}: & \leftarrow \underline{\mathtt{p}} \\ \beta_0^2: & \mathtt{p} \leftarrow \underline{\mathtt{b}} \end{array} \right|}_{\pi_2} = \underbrace{\left| \begin{array}{rr} \mathsf{goal}: & \leftarrow \underline{\mathtt{p}} \\ \beta_0: & \mathtt{p} \leftarrow \underline{\mathtt{a}} \\ & \mathtt{p} \leftarrow \underline{\mathtt{b}} \\ [\beta_0]: & \mathtt{p} \leftarrow \underline{\mathtt{a} \vee \mathtt{b}} \end{array} \right|}_{\pi}.$$

Notice that Believer cannot move in neither of the starting plays, but he can certainly move in their combination by playing the rule $\mathtt{a} \vee \mathtt{b} \leftarrow$. However, as we will later see, such misbehaviors are evaded if the plays π_q come from a strategy splitting; the impatient can read Corollary 5.23 (p. 39).

Remark 5.6 (Combining goals). When combining plays, their common goal $\leftarrow \mathtt{G}$ does not really have to be all that common. We can combine a play in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G}_1)$ with a play in $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G}_2)$ to get a new play in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, where $\mathtt{G} := \mathtt{G}_1 \vee \mathtt{G}_2$. To do so, we first extract plays in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ by altering only the first move in each play so that it is restricted to $\mathtt{G}_q$; then we combine. But we will not need to do such a thing in this work.

We have seen how to combine plays; now it is time to restrict and split them.

5.1.2. Restricting and splitting plays

Definition 5.7 (Play restriction). Suppose that π is a play and consider the sequence obtained by restricting every believer move in π that contains ϕ to an identical move in which ϕ has been restricted to H_q. We denote this sequence by $\pi|^{\phi}_{H_q}$ and call it the *restriction* of the play π to H_q with respect to ϕ.

Theorem 5A. *For any play π of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, its restriction $\pi_q \coloneqq \pi|^{\phi}_{H_q}$ is a valid play in $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$.*

Proof. Note that the only alteration performed leaves all bodies intact, so that every doubter move remains valid. The only case where a head is altered is when a believer move β of π includes the forbidden rule ϕ. Denoting by β^q the corresponding believer move of π_q, it is evident that $\mathsf{head}([\beta^q]) \subseteq \mathsf{head}([\beta])$ so that in both plays, every believer move is valid as well. ▮

Naturally we define splitting as a pair of restrictions:

Definition 5.8 (Play splitting)**.** Given a play π of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, the *play splitting of π with respect to ϕ over $\mathcal{H}$* is the pair

$$\pi|^{\phi}_{\mathcal{H}} \overset{\text{df}}{=} \left(\pi|^{\phi}_{H_1}, \pi|^{\phi}_{H_2}\right).$$

Thanks to Theorem 5A, these plays are indeed valid in the corresponding games.

After all this work on plays, strategies are next; but we have done most of the hard work in this section, so that the following one will be a breeze.

5.2. Strategies

In the previous section we defined combination, restriction, and splitting for plays in a given game, and proved the correctness of these definitions. Now we will do the same for strategies, keeping the same notation that we agreed upon. The combination and the splitting of strategies lie at the hearts of our proofs of completeness and soundness respectively.

5.2.1. Combining strategies

Combination of plays can be extended to strategies in a straightforward way:

Definition 5.9 (Combination of strategies)**.** Given two strategies σ_1 and σ_2 in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively, we define their *combination*

$$\sigma_1 \curlyvee \sigma_2 \overset{\text{df}}{=} \{\pi_1 \curlyvee \pi_2 \mid \pi_1 \in \sigma_1 \text{ and } \pi_2 \in \sigma_2\}.$$

Definition 5.10. Given a play $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$, we call the elements of the set

$$\mathbf{C}(\pi) \overset{\text{df}}{=} \{(\pi_1, \pi_2) \in \sigma_1 \times \sigma_2 \mid \pi_1 \curlyvee \pi_2 = \pi\}$$

the *creators* of π from σ_1 and σ_2. Equipped with the product order induced by either $\sqsubseteq$ or $\sqsubseteq_{\mathrm{e}}$, the set $\mathbf{C}(\pi)$ becomes a poset; and as we are about to see, it always has a least element: a pair which we naturally call the *shortest creators* of π from σ_1 and σ_2. Note that the two orderings $\sqsubseteq$ and $\sqsubseteq_{\mathrm{e}}$ coincide in this poset since all plays involved come from strategies, and therefore are of even length.

We will now prove an important "decomposition" property, which essentially allows us to reverse play-combination in case the two plays come from appropriate strategies.

Lemma 5.16 (Reversibility of combination)**.** *Given a play $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$, we can extract in a unique way, two synchronized quasiplays τ_1 and τ_2, each of length $|\pi|$, such that $\tau_1 \ddot{\curlyvee} \tau_2 = \pi$ and both τ_q* agree with *σ_q, i.e.,* $\mathsf{rmstall}(\tau_q) \in \sigma_q$.

Proof. Corollary 5.14 guarantees that $|\pi|$ is even, which allows us to prove the lemma by induction on $\ell \coloneqq |\pi| \,/\, 2$:

Base: $(\ell = 0)$. Trivially, $(\langle\,\rangle, \langle\,\rangle)$ is the pair that we seek.

Induction step: $(\ell = n + 1)$. Let

$$\pi \coloneqq \langle \delta_0, \beta_0, \ldots, \delta_n, \beta_n \rangle .$$

By the induction hypothesis, we know that there are two unique, synchronized quasiplays

$$\tau_1' \coloneqq \left\langle \delta_0^1, \beta_0^1, \ldots, \delta_{n-1}^1, \beta_{n-1}^1 \right\rangle$$
$$\tau_2' \coloneqq \left\langle \delta_0^2, \beta_0^2, \ldots, \delta_{n-1}^2, \beta_{n-1}^2 \right\rangle$$

which combine into π^- and agree with the corresponding strategies. Notice that if (τ_1, τ_2) satisfies the requested properties for π, then (τ_1^-, τ_2^-) satisfies them for π^-, so that $\tau_1^- = \tau_1'$ and $\tau_2^- = \tau_2'$. This guarantees that the following construction is the only one possible. We need to determine δ_n^q and β_n^q. Since π is a play, the doubter move δ_n is either the first move $(n = 0)$, or it consists of doubts from the bodies of the last believer moves in τ_1' and τ_2' $(n > 0)$. In the first case, we set $\delta_n^q \coloneqq \delta_n$, while in the second one, we split it in a unique way in two parts,

$$\delta_n \coloneqq \delta_n^1 + \delta_n^2,$$

such that δ_n^q is a valid doubter response to τ_q'. We now use these δ_n^q to obtain the corresponding believer moves $\beta_n^{q\,*}$. For this we appeal to the fact that σ_q (being deterministic) can have at most one next-move for the play $\mathsf{rmstall}\big(\tau_q' \mathbin{+\!\!+} \langle \delta_n^q \rangle\big)$, and we know that it has at least one since $\pi \in \sigma_1 \curlyvee \sigma_2$:

$$\beta_n^q \coloneqq \big(\mathsf{answer}_{\sigma_q} \circ \mathsf{rmstall}\big)\,\big(\tau_q' \mathbin{+\!\!+} \langle \delta_n^q \rangle\big).$$

Note that as this is an answer from σ_q, the quasiplay $\tau_q' \mathbin{+\!\!+} \langle \delta_n^q, \beta_n^q \rangle$ remains in agreement with it. ∎

Definition 5.11 (Projections of plays)**.** We call the quasiplay τ_q of the above lemma the *projection of π on σ_q* and denote it by $\mathsf{proj}_{\sigma_q}(\pi)$.

By their construction, projections satisfy the following property:

Property 5.17. *Given any $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$,*

$$\mathsf{proj}_{\sigma_q}\big(\pi^-\big) = \Big(\mathsf{proj}_{\sigma_q}(\pi)\Big)^- .$$

In other words, denoting by T_q the sets of quasiplays in $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$, the diagram in Figure 1 commutes.

$$\begin{tikzcd}
\mathrm{T}_1 \arrow[d, "\overline{\ }"] & \sigma_1 \curlyvee \sigma_2 \arrow[l, "\mathsf{proj}_{\sigma_1}"'] \arrow[r, "\mathsf{proj}_{\sigma_2}"] \arrow[d, "\overline{\ }"] & \mathrm{T}_2 \arrow[d, "\overline{\ }"] \\
\mathrm{T}_1 & \sigma_1 \curlyvee \sigma_2 \arrow[l, "\mathsf{proj}_{\sigma_1}"] \arrow[r, "\mathsf{proj}_{\sigma_2}"'] & \mathrm{T}_2
\end{tikzcd}$$

Figure 1: Commutative diagram for Property 5.17.

A link between projections and shortest creators is revealed: starting with π as above, first use Lemma 5.16 to obtain the quasiplay projections of π. Then, using $\mathsf{rmstall}$, remove all existing stall–follow moves; and finally, in case the quasiplay ended in a stall move, appeal to the strategy's totality to append the appropriate answer from σ_q. Following these steps, one actually obtains the shortest creators of π from σ_1 and σ_2. This will be clarified shortly in a lemma—but first, a definition that we will need:

Definition 5.12. Let $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$. We define the function $\mathsf{cr}_{\sigma_q} : \sigma \rightharpoonup \sigma_q$ as the composition

$$\mathsf{cr}_{\sigma_q} \overset{\mathsf{df}}{=} \mathsf{answer}_{\sigma_q} \circ \mathsf{rmstall} \circ \mathsf{proj}_{\sigma_q},$$

and simply write

$$\mathsf{cr}(\pi) \overset{\mathsf{df}}{=} (\mathsf{cr}_{\sigma_1}(\pi), \mathsf{cr}_{\sigma_2}(\pi))$$

for the function $\mathsf{cr} : \sigma \rightharpoonup \sigma_1 \times \sigma_2$. Both functions will turn out to be total.

Lemma 5.18. *Let $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$. Then*

$$\mathsf{cr}(\pi) = \min \mathbf{C}(\pi).$$

Proof. Let $\pi \coloneqq \langle \delta_0, \beta_0, \ldots, \delta_n, \beta_n \rangle \in \sigma_1 \curlyvee \sigma_2$ and pick any pair of creators $(\pi_1, \pi_2) \in \mathbf{C}(\pi)$, so that $\pi_1 \curlyvee \pi_2 = \pi$. We will show that $\mathsf{cr}(\pi) \sqsubseteq_{\mathrm{e}} (\pi_1, \pi_2)$. Set $(\tau_1, \tau_2) \coloneqq \mathsf{sync}(\pi_1, \pi_2)$, and write

$$\begin{aligned}
\tau_1 &\coloneqq \left\langle \delta_0^1, \beta_0^1, \delta_1^1, \beta_1^1, \ldots, \delta_{n_1}^1, \beta_{n_1}^1 \right\rangle \\
\tau_2 &\coloneqq \left\langle \delta_0^2, \beta_0^2, \delta_1^2, \beta_1^2, \ldots, \delta_{n_2}^2, \beta_{n_2}^2 \right\rangle.
\end{aligned}$$

By definition, $\pi = \tau_1 \ddot{\curlyvee} \tau_2$, so that

$$\pi = \left\langle \underbrace{\delta_0^1 + \delta_0^2}_{\delta_0}, \underbrace{\beta_0^{1^*} + \beta_0^{2^*}}_{\beta_0}, \underbrace{\delta_1^1 + \delta_1^2}_{\delta_1}, \underbrace{\beta_1^{1^*} + \beta_1^{2^*}}_{\beta_1}, \ldots, \underbrace{\delta_n^1 + \delta_n^2}_{\delta_n}, \underbrace{\beta_n^{1^*} + \beta_n^{2^*}}_{\beta_n} \right\rangle,$$

where $n \coloneqq \min \{n_1, n_2\}$. Now, using the monotonicity of $\mathsf{rmstall}$ and $\mathsf{answer}_{\sigma_q}$

(Properties 4.3 and 4.4), we reason:

$$\begin{aligned}
\mathsf{proj}_{\sigma_q}(\pi) \sqsubseteq_{\mathrm{e}} \tau_q &\implies \mathsf{rmstall}\big(\mathsf{proj}_{\sigma_q}(\pi)\big) \sqsubseteq_{\mathrm{e}} \mathsf{rmstall}(\tau_q) \\
&\implies \mathsf{answer}_{\sigma_q}\Big(\mathsf{rmstall}\big(\mathsf{proj}_{\sigma_q}(\pi)\big)\Big) \sqsubseteq_{\mathrm{e}} \mathsf{answer}_{\sigma_q}(\mathsf{rmstall}(\tau_q)) \\
&\implies \mathsf{cr}(\pi) \sqsubseteq_{\mathrm{e}} \mathsf{answer}_{\sigma_q}(\mathsf{rmstall}(\tau_q)).
\end{aligned}$$

Notice that τ_q cannot end in a stalling move, and so $\mathsf{rmstall}(\tau_q)$ will have even length. According to Definition 4.11, we can then expect that

$$\mathsf{answer}_{\sigma_q}(\mathsf{rmstall}(\tau_q)) = \mathsf{rmstall}(\tau_q),$$

so that by using Property 5.9(iii) we derive

$$\mathsf{cr}(\pi) \sqsubseteq_{\mathrm{e}} \mathsf{rmstall}(\tau_q) = \pi_q,$$

which is what we wanted to prove. ∎

We now relate the shortest creators of a play with those of its even prefixes.

Proposition 5.19. *Let $\varepsilon \neq \pi \in \sigma := \sigma_1 \curlyvee \sigma_2$, and let $\pi_q := \mathsf{cr}_{\sigma_q}(\pi)$. Then*

$$\mathsf{cr}\big(\pi^-\big) = \begin{cases} \big(\pi_1^-, \pi_2\,\big) & \text{if (a)}, \\ \big(\pi_1\,, \pi_2^-\big) & \text{if (b)}, \\ \big(\pi_1^-, \pi_2^-\big) & \text{if (c)}, \end{cases}$$

depending on whether (a) *the penultimate believer move of* $\mathsf{proj}_{\sigma_2}(\pi)$ *is a stalling,* (b) *the penultimate believer move of* $\mathsf{proj}_{\sigma_1}(\pi)$ *is a stalling, or* (c) *otherwise.*

Proof. This is a consequence of Property 5.17. Perhaps the weird-looking conditions need further explanation. The last believer move of π_q is needed to form the last move of π^- iff the penultimate move of $\mathsf{proj}_{\sigma_q}(\pi)$ is a stalling. This holds because once we project a play to σ_q, stalling moves might appear to the quasiplay projections. If the penultimate move of $\mathsf{proj}_{\sigma_q}(\pi)$ is such a stalling, once we delete the last move from π_q to obtain π_q^-, and use $\mathsf{rmstall}$ to remove all stall–follow moves, the resulting play will be of odd length. $\mathsf{answer}_{\sigma_q}(\tau)$ will then reproduce the move we deleted, thus bringing us back to π_q.

Since $\curlyvee$ never concatenates two stalling moves, $\mathsf{proj}_{\sigma_1}(\pi)$ and $\mathsf{proj}_{\sigma_2}(\pi)$ will never stall simultaneously; this proves that the conditions (a)–(c) are mutually exclusive. ∎

This proposition might become clearer after inspecting the three commutative diagrams of Figure 2. There, the stated equality is represented by the commutativity of an appropriate diagram, one for each case (a)–(c).

Corollary 5.20. *Let $\varepsilon \neq \pi \in \sigma := \sigma_1 \curlyvee \sigma_2$. Then*

(i) $\min \mathbf{C}(\pi^-) \sqsubset_{\mathrm{e}} \min \mathbf{C}(\pi)$;

$$\begin{tikzcd}
\sigma_1 \arrow[d, "-"] & \sigma_1 \curlyvee \sigma_2 \arrow[l, "\mathsf{cr}_{\sigma_1}"'] \arrow[d, "-"] \arrow[rd, bend left, "\mathsf{cr}_{\sigma_2}"] & \\
\sigma_1 & \sigma_1 \curlyvee \sigma_2 \arrow[l, "\mathsf{cr}_{\sigma_1}"] \arrow[r, "\mathsf{cr}_{\sigma_2}"'] & \sigma_2
\end{tikzcd}$$

Figure 2a: The penultimate believer move of $\mathsf{proj}_{\sigma_2}(\pi)$ is a stalling.

$$\begin{tikzcd}
 & \sigma_1 \curlyvee \sigma_2 \arrow[ld, bend right, "\mathsf{cr}_{\sigma_1}"'] \arrow[r, "\mathsf{cr}_{\sigma_2}"] \arrow[d, "-"] & \sigma_2 \arrow[d, "-"] \\
\sigma_1 & \sigma_1 \curlyvee \sigma_2 \arrow[l, "\mathsf{cr}_{\sigma_1}"] \arrow[r, "\mathsf{cr}_{\sigma_2}"'] & \sigma_2
\end{tikzcd}$$

Figure 2b: The penultimate believer move of $\mathsf{proj}_{\sigma_1}(\pi)$ is a stalling.

(ii) $\min \mathbf{C}(\pi') \sqsubseteq_{\mathrm{e}} \min \mathbf{C}(\pi)$, *for any* $\pi' \sqsubseteq_{\mathrm{e}} \pi$.

Theorem 5B. *The set* $\sigma := \sigma_1 \curlyvee \sigma_2$, *is a strategy in* $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.

Proof. Foremost it is indeed a set of plays in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ thanks to Corollary 5.12.

Non-empty. Since σ_1 and σ_2 are strategies, they are both non-empty, and so there is at least one play in σ (obtained by combination).

Even-length. This is a direct application of Corollary 5.14.

Even-prefix-closed. This is immediate by Corollary 5.20, since both σ_q are strategies and therefore closed under even prefixes.

Deterministic. Towards a contradiction, assume that π and $\widetilde{\pi}$ are plays of even length in σ that differ only in the last believer move:

$$\pi := \langle \delta_0, \beta_0, \ldots, \delta_n, \beta_n \rangle$$
$$\widetilde{\pi} := \left\langle \delta_0, \beta_0, \ldots, \delta_n, \widetilde{\beta_n} \right\rangle .$$

For each of them, use Lemma 5.16 to extract its two quasiplay projections on σ_1 and σ_2; (τ_1, τ_2) from π and $(\widetilde{\tau_1}, \widetilde{\tau_2})$ from $\widetilde{\pi}$.

$$\tau_1 := \left\langle \delta_0^1, \beta_0^1, \ldots, \delta_n^1, \beta_n^1 \right\rangle \qquad \widetilde{\tau_1} := \left\langle \delta_0^1, \beta_0^1, \ldots, \delta_n^1, \widetilde{\beta_n^1} \right\rangle$$
$$\tau_2 := \left\langle \delta_0^2, \beta_0^2, \ldots, \delta_n^2, \beta_n^2 \right\rangle \qquad \widetilde{\tau_2} := \left\langle \delta_0^2, \beta_0^2, \ldots, \delta_n^2, \widetilde{\beta_n^2} \right\rangle ,$$

so that

$$\pi = \left\langle \delta_0^1 \mathbin{+\!\!+} \delta_0^2, {\beta_0^1}^* \mathbin{+\!\!+} {\beta_0^2}^*, \ldots, \delta_n^1 \mathbin{+\!\!+} \delta_n^2, {\beta_n^1}^* \mathbin{+\!\!+} {\beta_n^2}^* \right\rangle$$
$$\widetilde{\pi} = \left\langle \delta_0^1 \mathbin{+\!\!+} \delta_0^2, {\beta_0^1}^* \mathbin{+\!\!+} {\beta_0^2}^*, \ldots, \delta_n^1 \mathbin{+\!\!+} \delta_n^2, \widetilde{\beta_n^1}^* \mathbin{+\!\!+} \widetilde{\beta_n^2}^* \right\rangle .$$

$$\begin{tikzcd}
\sigma_1 \arrow[d, "-"] & \sigma_1 \curlyvee \sigma_2 \arrow[l, "\mathsf{cr}_{\sigma_1}"'] \arrow[r, "\mathsf{cr}_{\sigma_2}"] \arrow[d, "-"] & \sigma_2 \arrow[d, "-"] \\
\sigma_1 & \sigma_1 \curlyvee \sigma_2 \arrow[l, "\mathsf{cr}_{\sigma_1}"] \arrow[r, "\mathsf{cr}_{\sigma_2}"'] & \sigma_2
\end{tikzcd}$$

Figure 2c: Otherwise.

Now, which of the four statements $\phi \in {\beta_n^q}^*$ and $\phi \in \widetilde{\beta_n^q}^*$ hold? By a tedious and trivial inspection of all 16 different cases that arise, one can confirm that every case leads to a contradiction, by obtaining *two* plays of the *same* strategy σ_q, differing only in their final (believer) move. This is of course absurd because strategies are deterministic. ▌

Proposition 5.21 (Preservation of totality)**.** *The combined strategy $\sigma_1 \curlyvee \sigma_2$ is total if both σ_1 and σ_2 are total.*

Proof. Suppose that we are given a play $\langle \delta_0, \beta_0, \ldots, \delta_n, \beta_n, \delta_{n+1} \rangle$ such that the immediate prefix

$$\pi := \langle \delta_0, \beta_0, \ldots, \delta_n, \beta_n \rangle \in \sigma.$$

We seek a believer move β_{n+1} such that $\pi \mathbin{+\!\!+} \langle \delta_{n+1}, \beta_{n+1} \rangle \in \sigma$. Use Lemma 5.16 to extract the quasiplay projections (τ_1, τ_2) of π on σ_1 and σ_2, so that

$$\pi = \tau_1 \mathbin{\ddot{\curlyvee}} \tau_2 := \left\langle \delta_0^1 \mathbin{+\!\!+} \delta_0^2, {\beta_0^1}^* \mathbin{+\!\!+} {\beta_0^2}^*, \ldots, \delta_n^1 \mathbin{+\!\!+} \delta_n^2, {\beta_n^1}^* \mathbin{+\!\!+} {\beta_n^2}^* \right\rangle .$$

Since δ_{n+1} is a valid next-move for π, it consists of doubts from the bodies of β_n^1 and β_n^2 (Property 5.1(ii)), so that it can be unambiguously split into two sequences of such doubts $\delta_{n+1} := \delta_{n+1}^1 \mathbin{+\!\!+} \delta_{n+1}^2$. By the totality of σ_q, there must be at least one believer move β_{n+1}^q satisfying

$$\pi_q^+ := \mathsf{rmstall}(\tau_q) \mathbin{+\!\!+} \left\langle \delta_{n+1}^q, \beta_{n+1}^q \right\rangle \in \sigma_q.$$

Easily now, $\pi_1^+ \curlyvee \pi_2^+$ contains the believer move that we need. ▌

Proposition 5.22 (Preservation of finiteness)**.** *The combined strategy $\sigma_1 \curlyvee \sigma_2$ is finite if both σ_1 and σ_2 are finite.*

Proof. By the definition of strategy combination, $|\sigma|$ is bounded by $|\sigma_1 \times \sigma_2|$, which is finite since both σ_q are finite. ▌

Remembering that total + finite = winning, we arrive at the following corollary:

Corollary 5.23 (Preservation of winning)**.** *The combined strategy $\sigma_1 \curlyvee \sigma_2$ is winning if both σ_1 and σ_2 are winning.*

5.2.2. Restricting and splitting strategies

Definition 5.13 (Strategy restriction)**.** Let σ be a strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. Then the *restriction* of σ with respect to ϕ on H_q, is the set of plays defined by:

$$\sigma|^{\phi}_{H_q} \overset{\mathsf{df}}{=} \left\{ \pi|^{\phi}_{H_q} \;\middle|\; \pi \in \sigma \right\}.$$

Theorem 5C. *The set of plays $\sigma_q := \sigma|^{\phi}_{H_q}$ is a valid strategy in $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$.*

Proof. *Non-empty.* This is trivial, since $\sigma \neq \emptyset$.

Even-length. It is obvious that restriction leaves lengths of plays intact, so that every member of σ_q has even length.

Even-prefix-closed. Let $\pi'_q \sqsubseteq \pi_q \in \sigma_q$, with $\ell := |\pi'_q|$ even. We need $\pi'_q \in \sigma_q$. Since $\pi_q \in \sigma_q$, there is a $\pi \in \sigma$ such that $\pi_q = \pi|^{\phi}_{H_q}$. We now compute:

$$\pi'_q = \pi_q\restriction_{\ell} = \left(\pi|^{\phi}_{H_q}\right)\restriction_{\ell} = \underbrace{(\pi\restriction_{\ell})}_{\in\sigma}|^{\phi}_{H_q},$$

which shows that $\pi'_q \in \sigma_q$ as was desired.

Deterministic. Suppose that we have the following plays in σ_q:

$$\pi_q := \langle \delta^q_0, \beta^q_0, \delta^q_1, \beta^q_1, \ldots, \delta^q_k, \beta^q_k \rangle,$$
$$\widetilde{\pi_q} := \left\langle \delta^q_0, \beta^q_0, \delta^q_1, \beta^q_1, \ldots, \delta^q_k, \widetilde{\beta^q_k} \right\rangle,$$

so that there are plays

$$\pi := \langle \delta_0, \beta_0, \delta_1, \beta_1, \ldots, \delta_k, \beta_k \rangle,$$
$$\widetilde{\pi} := \left\langle \widetilde{\delta_0}, \widetilde{\beta_0}, \widetilde{\delta_1}, \widetilde{\beta_1}, \ldots, \widetilde{\delta_k}, \widetilde{\beta_k} \right\rangle,$$

in σ, such that

$$\pi_q = \pi|^{\phi}_{H_q} \qquad \text{and} \qquad \widetilde{\pi_q} = \widetilde{\pi}|^{\phi}_{H_q}.$$

Observe that since play-splitting leaves all doubter moves unaltered, we have that $\delta_i = \widetilde{\delta_i}\,(= \delta^q_i)$ for all $i = 0, \ldots, k$. By finite induction on i we show the corresponding equalities for the believer moves. Assume (the inductive hypothesis) that $\beta_j = \widetilde{\beta_j}$ holds for $0 \leq j < i \leq k$. Now look at the plays $\pi\restriction_{2i+2}$ and $\widetilde{\pi}\restriction_{2i+2}$ which both belong in σ since it is prefix-closed. Then $\beta_i = \widetilde{\beta_i}$ since both plays belong in the same (deterministic) strategy σ. This essentially yields $\pi = \widetilde{\pi}$ and consequently $\pi_q = \widetilde{\pi_q}$, which establishes the determinacy of σ_q. ▮

Proposition 5.24 (Preservation of totality)**.** *If σ is total, then so is $\sigma|^{\phi}_{H_q}$.*

Proof. We are given a play $\pi_q \in \sigma_q$ and a doubter move δ such that $\pi_q +\!\!+\, \delta$ is valid in $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$, and we seek a believer next-move for it. By the hypothesis, there exists some $\pi \in \sigma$, such that $\pi_q = \pi|^{\phi}_{H_q}$. Since bodies are left intact by restriction, $\pi +\!\!+\, \delta$ is valid in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. Hence, by the totality of σ, there is a believer move β such that $\pi +\!\!+\, \langle \delta, \beta \rangle \in \sigma$. This means that $(\pi +\!\!+\, \langle \delta, \beta \rangle)|^{\phi}_{H_q} \in \sigma_q$, and its last move is the believer move that we sought. ▮

Proposition 5.25 (Preservation of finiteness)**.** *If σ is finite, then so is $\sigma|^{\phi}_{H_q}$.*

Proof. By definition, every play in $\sigma|^{\phi}_{H_q}$ is created by restricting a play of σ. This grants finiteness, as $\sigma|^{\phi}_{H_q}$ is nothing more than the image of a function $(\pi \mapsto \pi|^{\phi}_{H_q})$ whose domain is the finite set σ. ▮

Corollary 5.26 (Preservation of winning)**.** *If σ is winning, then so is $\sigma|^{\phi}_{H_q}$.*

For one last time, we use restriction to obtain splitting:

Definition 5.14 (Strategy splitting)**.** Let σ be a strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. The *splitting of σ with respect to ϕ over $\mathcal{H}$* is the pair

$$\sigma|^{\phi}_{\mathcal{H}} \overset{\text{df}}{=} \left(\sigma|^{\phi}_{H_1}, \sigma|^{\phi}_{H_2}\right).$$

This completes our game-theoretic weaponry for DLP programs. We have finally reached the point that we can put all those pieces together to prove that the DLP game semantics is sound and complete with respect to the minimal model semantics.

6. Soundness and completeness

Before proceeding to prove soundness and completeness of the DLP game semantics, we state some known results for the case of LP.

Theorem 6A (Di Cosmo–Loddo–Nicolet)**.** *The LP game semantics is sound and complete with respect to SLD resolution.*

Theorem 6B (Clark)**.** *SLD resolution is sound and complete with respect to the least Herbrand model semantics.*

The first of these is proved in [8], while the second one is due to [7] (and can also be found in [20, Theorems 7.1 and 8.6]). Putting the above two theorems together, we arrive at the correctness of the LP game semantics:

Corollary 6.1 (Soundness and completeness of the LP game semantics)**.** *Let $\mathcal{P}$ be an LP program, and $\leftarrow \mathtt{p}$ a goal. Then, there is a winning strategy in the associated LP game iff $\mathtt{p}$ belongs to the least Herbrand model of $\mathcal{P}$.*

To prove the equivalence that was promised in the introduction we need the following lemma, which relates the models of a splitting of a program with those of the original program.

Lemma 6.2 (Inclusions)**.** *Let $(\mathcal{P}_1, \mathcal{P}_2)$ be the splitting of $\mathcal{P}$ with respect to ϕ over (H_1, H_2). Then*

$$\mathrm{MM}(\mathcal{P}) \subseteq \mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P}).$$

PROOF. Let $\phi_1 := \phi|_{H_1}$ and $\phi_2 := \phi|_{H_2}$. For the first inclusion, let $S \in \mathrm{MM}(\mathcal{P})$, and suppose $S \notin \mathrm{MM}(\mathcal{P}_1)$. We need $S \in \mathrm{MM}(\mathcal{P}_2)$. There are two ways in which S can fail to be in $\mathrm{MM}(\mathcal{P}_1)$: either it is a model but not a minimal one, or it is not even a model to begin with.
CASE 1: *S is a non-minimal model of $\mathcal{P}_1$.* There exists then, a proper submodel $S_0 \subsetneq S$ of $\mathcal{P}_1$, with $S_0 \models \mathcal{P}_1$. By definition, this would also be a model of $\mathcal{P}$, and therefore S would not be minimal in $\mathcal{P}$, which is a contradiction, so that this case can never arise.
CASE 2: *S is not a model of $\mathcal{P}_1$.*

$$\begin{aligned} S \in \mathrm{MM}(\mathcal{P}) &\implies S \in \mathrm{HM}(\mathcal{P}) \\ &\implies S \models \psi \text{ for all } \psi \in \mathcal{P} \\ &\implies S \models \psi \text{ for all } \psi \in \mathcal{P}_1 \setminus \{\phi_1\} \qquad (\mathcal{P}_1 \setminus \{\phi_1\} \subset \mathcal{P}) \\ &\implies S \not\models \phi_1 \qquad \text{(by case hypothesis)} \end{aligned}$$

Since (ϕ_1, ϕ_2) is the splitting of ϕ over (H_1, H_2), S is forced to satisfy ϕ_2, and therefore satisfies every element of $\mathcal{P}_2$, so that $S \in \mathrm{HM}(\mathcal{P}_2)$.

It remains to show that it is minimal in $\mathcal{P}_2$. But if it was not, we would arrive at the same contradiction as in case 1; therefore, $S \in \mathrm{MM}(\mathcal{P}_2)$. We have proved the inclusion

$$\mathrm{MM}(\mathcal{P}) \subseteq \mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2). \tag{1}$$

Obviously now, $\mathrm{MM}(\mathcal{P}_1) \subseteq \mathrm{HM}(\mathcal{P}_1)$ and $\mathrm{MM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P}_2)$, so by taking unions on both sides we obtain

$$\mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P}_1) \cup \mathrm{HM}(\mathcal{P}_2). \tag{2}$$

As $\mathcal{P}$ is a weaker program than its restrictions, we also have the inclusions $\mathrm{HM}(\mathcal{P}_1) \subseteq \mathrm{HM}(\mathcal{P})$ and $\mathrm{HM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P})$. Hence, by taking unions for one last time,

$$\mathrm{HM}(\mathcal{P}_1) \cup \mathrm{HM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P}). \tag{3}$$

Putting (1)–(3) together:

$$\mathrm{MM}(\mathcal{P}) \subseteq \mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P}_1) \cup \mathrm{HM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P}). \qquad \blacksquare$$

REMARK 6.1. One might be tempted to believe that some of these inclusions are actually equalities, but none of them holds in general, as the following example demonstrates: consider the DLP program $\mathcal{P}$ and its splitting

$$\mathcal{P} := \left\{ \begin{array}{r} \mathtt{b} \leftarrow \\ \mathtt{a} \lor \mathtt{b} \lor \mathtt{c} \leftarrow \end{array} \right\} \rightsquigarrow \left(\mathcal{P}_1 := \left\{ \begin{array}{r} \mathtt{b} \leftarrow \\ \mathtt{a} \leftarrow \end{array} \right\}, \quad \mathcal{P}_2 := \left\{ \begin{array}{r} \mathtt{b} \leftarrow \\ \mathtt{b} \lor \mathtt{c} \leftarrow \end{array} \right\} \right).$$

Then the corresponding sets of Herbrand models and minimal models are

$$\begin{aligned} \mathrm{HM}(\mathcal{P}) &= \{\{b\}, \{b,a\}, \{b,c\}, \{b,a,c\}\}, & \mathrm{MM}(\mathcal{P}) &= \{\{b\}\}, \\ \mathrm{HM}(\mathcal{P}_1) &= \{\{a,b\}\}, & \mathrm{MM}(\mathcal{P}_1) &= \{\{a,b\}\}, \\ \mathrm{HM}(\mathcal{P}_2) &= \{\{b\}, \{b,c\}\}, & \mathrm{MM}(\mathcal{P}_2) &= \{\{b\}\}, \end{aligned}$$

so that in this example all inclusions are proper:

$$\mathrm{MM}(\mathcal{P}) \subsetneq \mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2) \subsetneq \mathrm{HM}(\mathcal{P}_1) \cup \mathrm{HM}(\mathcal{P}_2) \subsetneq \mathrm{HM}(\mathcal{P}).$$

Theorem 6C (Soundness of the finite, clean DLP game semantics)**.** *Let $\mathcal{P}$ be a finite, clean DLP program, and $\leftarrow \mathtt{G}$ a goal. If there is a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, then G is true in every minimal model of $\mathcal{P}$.*

Proof is by induction on the number $N(\mathcal{P})$ of disjunction symbols ($\vee$) that appear in the heads of $\mathcal{P}$.

Base. Let σ be a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. We claim that *there exists a combo-free winning strategy σ' in the same game.* If so, observe that the head of the first believer move of every play of σ' must be one disjunct $\mathtt{g}$ of $\mathtt{G}$. Hence by altering the first (doubter) moves of its plays to correctly doubt $\mathtt{g}$, we end up with a combo-free, winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{g})$. This is now compatible with the LP game and we can use the soundness of the LP game semantics (Corollary 6.1) to obtain $g \in \mathrm{LHM}(\mathcal{P})$. But this means that G is true in $\mathrm{LHM}(\mathcal{P})$, the only minimal model of $\mathcal{P}$.
Proof of the claim. If σ contains no combo moves, we are done. Otherwise, pick a maximal play from σ such that it contains at least one combo move and look at the last one, β. We show that we can safely replace β by a non-combo move that contains exactly one of its rules, and still win the game. Towards a contradiction, suppose that no such rule exists. Then every rule in β contains a "bad" atom such that when doubted by the doubter, we cannot win. But this contradicts the fact that σ is winning, as it contains no answer for a doubter move that doubts exactly one bad atom from each rule of β.

Induction step. Since $\mathcal{P}$ is a proper DLP program, we can pick a proper DLP rule $\phi \in \mathcal{P}$, and split $\mathcal{P}$ with respect to it over some proper partition (H_1, H_2) of $\mathsf{head}(\phi)$ to get $(\mathcal{P}_1, \mathcal{P}_2)$. Notice that $\max\{N(\mathcal{P}_1), N(\mathcal{P}_2)\} < N(\mathcal{P})$, which allows us to use the induction hypothesis for both programs $\mathcal{P}_q$.

Let σ be a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, and split it likewise to derive two strategies σ_1 and σ_2 for $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively (Theorem 5C). Since σ is winning, σ_1 and σ_2 are also winning (Corollary 5.26), and so by the induction hypothesis we know that G must be true in every minimal model of $\mathcal{P}_1$ and in every minimal model of $\mathcal{P}_2$. In other words, G is true in the union $\mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2)$; so by Lemma 6.2, it is true in every element of $\mathrm{MM}(\mathcal{P})$. ▮

Theorem 6D (Completeness of the finite, clean DLP game semantics)**.** *Let $\mathcal{P}$ be a finite, clean DLP program, and $\leftarrow \mathtt{G}$ a goal. If G is true in every minimal model of $\mathcal{P}$, then there is a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.*

Proof is again by induction on $N(\mathcal{P})$:

Base. Since $\mathrm{LHM}(\mathcal{P}) \models G$, there is at least one $g \in G$ with $g \in \mathrm{LHM}(\mathcal{P})$. Using the completeness of the LP game semantics (Corollary 6.1), we obtain a winning strategy σ in the LP game for $\mathcal{P}$ with the goal $\leftarrow \mathtt{g}$. Without any modification, we can consider this as a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{g})$. It remains to correct all first (doubter) moves by including all extra doubts from $\mathtt{G}$ in them. In this way, we have a winning strategy σ' in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.

Induction step. Again, pick a proper DLP rule $\phi \in \mathcal{P}$, and split $\mathcal{P}$ to get $(\mathcal{P}_1, \mathcal{P}_2)$. We know that G is true in every minimal model of $\mathcal{P}$. Therefore, it is also true in every model of $\mathcal{P}$ (because a non-minimal model can only make more formulæ true than a minimal one, not less). Using Lemma 6.2 again, $\mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P})$, so that G is true in every minimal model of $\mathcal{P}_1$ and in every minimal model of $\mathcal{P}_2$. By the induction hypothesis, there are two winning strategies σ_1 and σ_2 in the games $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively. Using Theorem 5B we can combine them to get a new strategy $\sigma_1 \curlyvee \sigma_2$ for $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, which is winning by Corollary 5.23. ∎

In order to generalize these results to general DLP programs we need the following key lemma which connects games on a general DLP program $\mathcal{P}$ with its clean version, $\widehat{\mathcal{P}}$.

Lemma 6.3. *Let $\mathcal{P}$ be a general DLP program, and $\leftarrow \mathtt{G}$ a DLP goal. Then, there is a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ iff there is a winning strategy in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$.*

Proof. "⇒": We are given a winning strategy σ for $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. To win in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$, we move as follows: suppose that the believer following σ plays

$$\beta := \langle \psi_1, \dots, \psi_n \rangle .$$

Now, we might not be able to play the same move, because some of the ψ_i's may be unclean, and therefore not available in $\widehat{\mathcal{P}}$. But, since each $\rho \in \mathcal{P}$ gives rise to a sequence of clean rules $\widehat{\rho}$ (actually a set, but we can fix an ordering and get a sequence out of it), we play:

$$\widehat{\beta} := \widehat{\psi_1} +\!\!+ \cdots +\!\!+ \widehat{\psi_n}.$$

This describes a winning strategy $\widehat{\sigma}$ in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$. We must verify two things: (i) that $\widehat{\beta}$ is indeed a valid move, and (ii) that we know how to win from any next doubter move $\widehat{\delta}$. (i) is trivial: it is only the heads that affect the validity of believer moves. Regarding (ii), we make the following claim: *for every $\psi_i \in \beta$ and for any doubts $\widehat{\delta_i}$ on $\widehat{\psi_i}$, there is a disjunction $D^i \in \mathsf{body}(\psi_i)$ such that $D^i \subseteq \left[\widehat{\delta_i}\right]$.* Then, our move in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$ against a doubter move consisting of such doubts

$$\widehat{\delta} := \widehat{\delta_1} +\!\!+ \cdots +\!\!+ \widehat{\delta_n}$$

is σ's move for the doubts $\delta := \left\langle D^1, \dots, D^n \right\rangle$, valid thanks to the claim and the trivial observation that

$$\left[\widehat{\delta}\right] = \left[\widehat{\delta_1} +\!\!+ \cdots +\!\!+ \widehat{\delta_n}\right] = \left[\widehat{\delta_1}\right] \cup \cdots \cup \left[\widehat{\delta_n}\right].$$

Proof of the claim. Assume otherwise: there is a rule $\psi_i \in \beta$ of the form

$$\psi_i := \mathtt{E}_i \leftarrow \mathtt{D}^i_1 \ , \ \cdots \ , \ \mathtt{D}^i_{k_i},$$

and a $\widehat{\delta_i}$ on $\widehat{\psi_i}$ such that for all $D^i_j \in \mathsf{body}(\psi_i)$, $D^i_j \not\subseteq \left[\,\widehat{\delta_i}\,\right]$, i.e., there is a $d^i_j \in D^i_j$ with $d^i_j \notin \left[\,\widehat{\delta_i}\,\right]$. But this implies that $\widehat{\delta}$ did not doubt anything from the body of the rule

$$\mathtt{E}_i \leftarrow \mathtt{d}^i_1 \ , \ \cdots \ , \ \mathtt{d}^i_{k_i} \in \widehat{\psi_i},$$

which is impossible.

"$\Leftarrow$": In this direction we are given a strategy σ in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$. Again, to win in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, we follow σ for as long as it does not instruct us to include transformed rules, i.e., rules that do not exist in $\mathcal{P}$. Suppose now that $\beta \coloneqq \langle \psi_1, \dots, \psi_n \rangle$ is the move that σ would play, where some of the ψ_i's are transformed. We then play

$$\begin{aligned} \beta^{\vee} &\coloneqq \langle \psi^{\vee}_1, \dots, \psi^{\vee}_n \rangle \,, \\ \text{where} \quad \psi^{\vee}_i &\coloneqq \text{the first } \rho \in \mathcal{P} \text{ such that } \psi_i \in \widehat{\rho}. \end{aligned}$$

And this describes a winning strategy $\sigma^{\vee}$ in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. We must verify the same claims (i)–(ii) as above. (i) is trivial for the same reason. For (ii), let $\delta^{\vee} \coloneqq \langle D_1, \dots, D_n \rangle$ be such a doubter response to $\beta^{\vee}$ so that $D_i \in \mathsf{body}(\psi^{\vee}_i)$. This translates easily to a doubt δ on β, namely

$$\delta \coloneqq \langle \{d_1\}, \dots, \{d_n\} \rangle \,,$$

where $d_i \in D_i$ and $\{d_i\} \in \mathsf{body}(\psi_i)$, so that

$$[\,\delta\,] = \{d_1, \dots, d_n\} \subseteq D_1 \cup \cdots \cup D_n = [\,\delta^{\vee}\,] \,.$$

This allows us to copycat the move that σ would play in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$ against δ. We go on in this manner until the winning strategy σ plays a fact ψ among its rules; then $\psi^{\vee}$ will also be a fact. ▮

Theorem 6E (Soundness and completeness for finite DLP)**.** *Let $\mathcal{P}$ be a finite, general DLP program, and $\leftarrow \mathtt{G}$ a DLP goal. Then, there exists a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ iff G is true in every minimal model of $\mathcal{P}$.*

PROOF. We have proved everything we need:

there is a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$	$\iff$	there is a winning strategy in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$	(Lemma 6.3)
	$\iff$	G is true on every minimal model of $\widehat{\mathcal{P}}$	(Theorem 6C $\Rightarrow$) (Theorem 6D $\Leftarrow$)
	$\iff$	G is true on every minimal model of $\mathcal{P}$.	(Property 2.1) ▮

We now drop the heavy restriction of finiteness. This has the remarkable consequence that we can handle $\mathsf{ground}(\mathcal{P})$ for any first-order DLP program $\mathcal{P}$ (see also Remark 3.2).

Theorem 6F (Soundness and completeness for general DLP)**.** *Let $\mathcal{P}$ be a general DLP program, and $\leftarrow \mathtt{G}$ a disjunctive goal. Then, there exists a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ iff G is true in every minimal model of $\mathcal{P}$.*

Proof. *Completeness.* Assume G is true in every minimal model of $\mathcal{P}$, and therefore also in every model of $\mathcal{P}$, in symbols $\mathcal{P} \models G$. By the compactness theorem of propositional calculus, there exists a finite subset $\mathcal{P}_G \subseteq \mathcal{P}$ such that $\mathcal{P}_G \models G$. Specifically, G is true in every minimal model of $\mathcal{P}_G$, which allows us to use Theorem 6E to obtain a winning strategy σ_G in $\Gamma_{\mathcal{P}_G}(\leftarrow \mathtt{G})$. Observe now that this very strategy is still winning in the "bigger" game $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, since the believer will never pick any of the additional rules, and therefore the course of the game cannot change.

Soundness. Suppose we have a winning strategy σ for $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. This can only use a finite number of rules from $\mathcal{P}$, which form a finite subset $\mathcal{P}_\sigma \subseteq \mathcal{P}$. Obviously, σ is still winning in $\Gamma_{\mathcal{P}_\sigma}(\leftarrow \mathtt{G})$. This means that G is true in every minimal model of $\mathcal{P}_\sigma$ (by Theorem 6E again), and therefore in every model of $\mathcal{P}_\sigma$. Since there are no negations involved, G remains true in every model of the superset $\mathcal{P}$; specifically in every minimal one. ▮

Corollary 6.4 (Soundness and completeness for first-order DLP)**.** *Let $\mathcal{P}$ be a first-order DLP program, and C a positive ground clause. Then $\mathcal{P} \models C$ iff there exists a winning strategy in $\Gamma_{\mathsf{ground}(\mathcal{P})}(\leftarrow \mathtt{C})$.*

Proof.

$$\begin{aligned}
\mathcal{P} \models C &\iff C \text{ is true in every m.m. of } \mathcal{P} && \text{(Theorem 3B)}\\
&\iff C \text{ is true in every m.m. of } \mathsf{ground}(\mathcal{P}) && \text{(Property 3.2)}\\
&\iff \text{there is a winning } \sigma \text{ in } \Gamma_{\mathsf{ground}(\mathcal{P})}(\leftarrow \mathtt{C}) && \text{(Theorem 6F)}
\end{aligned}$$ ▮

7. Conclusion

In this article, we defined a game semantics for disjunctive logic programs and proved it equivalent to the standard denotational semantics. In order to accomplish this, we developed a novel presentation of DLP programs, studied its syntax and semantics, and defined the DLP game in terms of it. At this point, two directions of further investigation seem promising.

DLPN. The obvious next step is to unite the two games DLP and LPN to form a game for the case of DLPN, and prove its correctness by establishing the equivalence with the infinite-valued minimal model semantics of [6]. Expressed in the colloquial style of the equation in p. 18,

$$\begin{aligned}
\text{DLP game } &= \text{LP game} + \text{implicit rules} + \text{combo moves}\\
\text{LPN game } &= \text{LP game} + \text{rôle-switch}\\
\text{DLPN game } &\overset{?}{=} \text{LP game} + \text{rôle-switch} + \text{implicit rules} + \text{combo moves.}
\end{aligned}$$

In this way, we will have a first, truly uniform approach to semantics, one that is able to deal with all four main versions of logic programming.

Connections through games. As was mentioned in the introduction, the definitions, the statements, and in general the whole development, has been mainly influenced by games from the functional programming world. Once all four versions of logic programming have been dealt with in a uniform way via games, a fruitful connection with the functional games might be possible. In this way, games could help in building a long-desired bridge between the "sister paradigms" of declarative programming: logic and functional.

Acknowledgments

Foremost I am indebted to Olivier Laurent for his constructive guidance, patience, and support. Without his countless suggestions, I would still be struggling to prove much harder albeit equivalent claims. I would also like to thank Panos Rondogiannis and John Power for many helpful comments and enlightening conversations.

References

[1] S. Abramsky, R. Jagadeesan, P. Malacaria, Full abstraction for PCF, Information and Computation 163 (2000) 409–470.

[2] S. Abramsky, G. McCusker, Game semantics, in: H. Schwichtenberg, U. Berger (Eds.), Computational Logic: Proceedings of the 1997 Marktoberdorf Summer School, Springer-Verlag, 1999, pp. 1–56.

[3] J.M. Andreoli, Logic programming with focusing proofs in linear logic, Journal of Logic and Computation 2 (1992) 297–347.

[4] J.M. Andreoli, R. Pareschi, Logic programming with sequent systems, in: P. Schroeder-Heister (Ed.), Extensions of Logic Programming, volume 475 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1991, pp. 1–30.

[5] K.R. Apt, Logic programming, in: Handbook of theoretical computer science (vol. B): formal models and semantics, MIT Press, Cambridge, MA, USA, 1990, pp. 493–574.

[6] P. Cabalar, D. Pearce, P. Rondogiannis, W. Wadge, A purely model-theoretic semantics for disjunctive logic programs with negation, in: C. Baral, G. Brewka, J. Schlipf (Eds.), Logic Programming and Nonmonotonic Reasoning, volume 4483 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 44–57.

[7] K. Clark, Predicate Logic as a Computational Formalism, Ph.D. thesis, Queen Mary, University of London, 1980.

[8] R. Di Cosmo, J.V. Loddo, S. Nicolet, A game semantics foundation for logic programming, in: C. Palamidessi, H. Glaser, K. Meinke (Eds.), Principles of Declarative Programming, volume 1490 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1998, pp. 355–373. 10.1007/BFb0056626.

[9] M.H. van Emden, Quantitative deduction and its fixpoint theory, Journal of Logic Programming 3 (1986) 37–53.

[10] M.H. van Emden, R.A. Kowalski, The semantics of predicate logic as a programming language, Journal of the ACM 23 (1976) 569–574.

[11] M. Fitting, Fixpoint semantics for logic programming—a survey, Theoretical Computer Science 278 (1999) 25–51.

[12] C. Galanaki, P. Rondogiannis, W.W. Wadge, An infinite-game semantics for well-founded negation in logic programming, Annals of Pure and Applied Logic 151 (2008) 70–88. First Games for Logic and Programming Languages Workshop.

[13] M. Gelfond, Answer sets, in: F. van Harmelen, V. Lifschitz, B. Porter (Eds.), Handbook of Knowledge Representation, Elsevier, 2008.

[14] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, MIT Press, 1988, pp. 1070–1080.

[15] W. Hodges, Logic and games, in: E.N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, 2009, spring 2009 edition.

[16] J.M.E. Hyland, C.H. Ong, On full abstraction for PCF: I. Models, observables and the full abstraction problem; II. Dialogue games and innocent strategies; III. A fully abstract and universal game model, Information and Computation 163 (2000) 285–408.

[17] E. Komendantskaya, G. McCusker, J. Power, Coalgebraic semantics for parallel derivation strategies in logic programming, in: M. Johnson, D. Pavlovic (Eds.), AMAST, volume 6486 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 111–127.

[18] E. Komendantskaya, J. Power, Coalgebraic semantics for derivations in logic programming, in: Proceedings of the 4th international conference on Algebra and coalgebra in computer science, CALCO'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 268–282.

[19] V. Kountouriotis, C. Nomikos, P. Rondogiannis, A game-theoretic characterization of boolean grammars, Theoretical Computer Science 412 (2011) 1169–1183.

[20] J.W. Lloyd, Foundations of Logic Programming, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1987.

[21] J.W. Lloyd, R.W. Topor, Making Prolog more expressive, Journal of Logic Programming 3 (1984) 225–240.

[22] J. Lobo, J. Minker, A. Rajasekar, Foundations of disjunctive logic programming, MIT Press, Cambridge, MA, USA, 1992.

[23] J.V. Loddo, R.D. Cosmo, Playing logic programs with the alpha-beta algorithm, in: M. Parigot, A. Voronkov (Eds.), LPAR, volume 1955 of *Lecture Notes in Computer Science*, Springer, 2000, pp. 207–224.

[24] P. Lorenzen, Ein dialogisches konstruktivitätskriterium, in: Infinitistic Methods: Proceedings of the Symposium on Foundations of Mathematics, Warsaw, 2–9 September 1959, Pergamon Press, 1961, pp. 193–200.

[25] D. Miller, G. Nadathur, Programming with Higher-Order Logic, Cambridge University Press, 2012.

[26] D. Miller, G. Nadathur, F. Pfenning, A. Scedrov, Uniform proofs as a foundation for logic programming, Annals of Pure and Applied Logic 51 (1991) 125–157.

[27] D. Miller, A. Saurin, A game semantics for proof search: Preliminary results, Electronic Notes in Theoretical Computer Science 155 (2006) 543–563.

[28] J. Minker, On indefinite databases and the closed world assumption, volume 138 of *Lecture Notes in Computer Science*, Springer-Verlag, 1982.

[29] J. Needham, M.D. Vos, A games semantics of ASP, in: V. Dahl, I. Niemelä (Eds.), Logic Programming, volume 4670 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2007, pp. 460–461.

[30] H. Nickau, Hereditarily sequential functionals, in: A. Nerode, Y. Matiyasevich (Eds.), LFCS, volume 813 of *Lecture Notes in Computer Science*, Springer, 1994, pp. 253–264.

[31] C. Nomikos, P. Rondogiannis, A game semantics for intensional logic programming, Presented in Games for Logic and Programming Languages (GaLoP) VII, Dubrovnik, Croatia, 2012.

[32] D. Pym, E. Ritter, A games semantics for reductive logic and proof-search, in: D.R. Ghica, G. McCusker (Eds.), Games for Logic and Programming Languages (GALOP 2005), University of Edinburgh, 2–3 April 2005, pp. 107–123.

[33] R. Reiter, On closed world data bases, Logic and Databases (1978) 55–76.

[34] P. Rondogiannis, W.W. Wadge, Minimum model semantics for logic programs with negation-as-failure, ACM Trans. Comput. Logic 6 (2005) 441–467.

[35] Th. Tsouanas, Semantic approaches to Logic Programming, Master's thesis, 2010.

[36] J. Väänänen, Models and Games, Cambridge series in advanced mathematics, Cambridge University Press, 2011.

[37] A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (1991) 619–649.

Index