Game semantics of disjunctive logic programming

Thanos Tsouanas

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, game semantics were given to definite logic programs by Di Cosmo, Loddo, and Nicolet, and a few years later they were extended to deal with negation by Rondogiannis and Wadge. Both approaches were proven equivalent with the traditional denotational semantics. In this paper we define the 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 style of Abramsky–Jagadeesan–Malacaria and Hyland–Ong.

Keywords: logic programming, disjunctive logic programming, game semantics, logic programming semantics

2010 MSC: 03B70, 68N17, 68Q55, 91A40, 91A80

Contents

1 Introduction 2
2 Syntax 5
  2.1 Preliminaries ........................................ 6
  2.2 Disjunctive combinations .......................... 8
  2.3 Restrictions and splitting .......................... 9
3 Denotational semantics 10

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

\[ \text{this } \leftarrow \text{that}, \]

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:

\[ a \leftarrow b_1, \ldots, b_m. \] \hspace{1cm} (LP)

where the commas on the right stand for conjunctions. Denotational 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 [7] or [11] for further information. One extension is to allow negations to appear in bodies of rules:

\[ a \leftarrow b_1, \ldots, b_m, \sim c_1, \ldots, \sim c_k. \] \hspace{1cm} (LPN)

By negation, we mean negation-as-failure; the semantics we have in mind here are supplied by the well-founded model, defined in [18]. The extension in which we are interested in this article is the appearance of disjunctions in heads:

\[ a_1 \lor \cdots \lor a_n \leftarrow b_1, \ldots, b_m. \] \hspace{1cm} (DLP)
Instead of a single best model, in this case, we use a set of minimal models for the semantics, as defined in [14]. Disjunctive logic programs are extensively studied in [13]. Finally, one can consider both extensions simultaneously, by allowing both negations in bodies and disjunctions in heads:

\[ a_1 \lor \cdots \lor a_n \leftarrow b_1, \cdots, b_m, \sim c_1, \cdots, \sim c_k. \]  

(DLPN)

Satisfactory model-theoretic semantics for this extension were defined recently in [4].

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 denotational 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 associate with each program a revision operator (traditionally called \( T_P \)) and look at its fixes. In the development of game semantics, however, we will not need to use this approach. An excellent survey of fixpoint semantics for logic programming is [8].

Procedural semantics. The actual implementation of each of the above languages is 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. In this work, we do not touch this operational side of semantics; see [3] for the non-disjunctive and [13] for the disjunctive cases.

Game semantics. After the remarkable success of game semantics for functional programming languages (see [2], [1], and [10]), a game-theoretic approach to logic programming semantics was inevitable. Game semantics for the case of LP was defined in [6], and was there shown to be equivalent to the least Herbrand model semantics. Approximately a decade later, in [9], game semantics is given for LPN, equivalent to an infinite-valued refinement of the well-founded semantics (see [16]). A couple of years after that, two games to deal with DLP and DLPN were described informally in [17].

2What 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 2.2 for a solution.
In this article, game semantics for DLP is formally defined, studied, and proven correct:

**Soundness and completeness of DLP game semantics (Theorem 6E).**

The game semantics of DLP is equivalent to the minimal model semantics, i.e., on a DLP program \( \mathcal{P} \), given any disjunction \( D \),

\[
D \text{ is true wrt the DLP game semantics} \iff D \text{ is true wrt the minimal model semantics.}
\]

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 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 [6].

The history of denotational and game semantics for those four versions of logic programming is summarized in Table 1.

<table>
<thead>
<tr>
<th>Language</th>
<th>Denotational semantics</th>
<th>Game semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPN</td>
<td>well-founded model [18]</td>
<td>( \Leftrightarrow ) LPN game [9]</td>
</tr>
<tr>
<td>DLP</td>
<td>minimal-models [14]</td>
<td>( \Leftrightarrow ) this article</td>
</tr>
<tr>
<td>DLPN</td>
<td>( \infty )-valued-minimal-models [4]</td>
<td>–</td>
</tr>
</tbody>
</table>

**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 \( \Gamma \leftarrow G \) we associate a specific game \( \Gamma_{\mathcal{P}}(\Gamma \leftarrow G) \), 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. Finally, 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 into a stronger version of it (i.e., less disjunctive); (iv) use restriction to obtain splitting. We will follow these steps, again and again.

Be aware that even though the DLP game developed here is “backwards compatible” with the LP game, its formalization is drastically different from
those of both LP and 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, used for PCF and functional programming in general (see [2], [1], and [10]). Although we assume no such prior knowledge on this field, the initiated reader should hopefully feel at home.

**Notation.** The Greek letters \( \varphi, \psi, \) and \( \rho \) will stand for rules, as well as for the corresponding logic formul\( \). Programs will usually be denoted by calligraphic capital letters like \( P, Q, \) and \( R \). Lowercase letters such as \( a, d, f, p, \) etc. will always denote atoms, while uppercase such symbols will stand for sets of atoms. For example \( D \) could be the set \{\( d_1, \ldots, d_n \)\} which, as you shall shortly see, is identified with the disjunction \( d_1 \lor \cdots \lor d_n \). 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: \( D \) could stand for \( \langle D_1, \ldots, D_n \rangle \), each \( D_i \) being a set of atoms \{\( d_i^1, \ldots, d_i^k \)\}. The truth values \textit{true} and \textit{false} are written as \( T \) and \( F \) respectively; logical equivalence as \( \equiv \).

We will work with sequences a lot; we use \( \oplus \) for concatenation and \( | \cdot | \) for length. We shall write \( s \subseteq s' \) to indicate that the sequence \( s \) is a prefix of \( s' \), decorating it with an “\( _e \)” in case \( s \) is of even length: \( s \\
subseteq_e s' \) (note that \( \subseteq_e \subseteq \subseteq \)). Proper (even) prefixes will be shown as \( \sqsubset (\subseteq_e) \). \( s \mid_n \) 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 \( s^\preceq \) for this, with the convention that it leaves the empty play unaltered: \( \langle \rangle \preceq \langle \rangle \). Influenced by lists in programming languages, we use \( :: \) for the “cons” operator:

\[
x :: \langle \rangle \equiv \langle x \rangle \\
x :: \langle x_1, \ldots, x_n \rangle \equiv \langle x, x_1, \ldots, x_n \rangle.
\]

As has been just demonstrated, \( \equiv \) is used to introduce the definition of a function or symbol, while \( \vdash \) is used to “let-bind” the variables appearing on its left to the corresponding expressions that appear on its right (\( \vdash \) for 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 \( A \) 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 \( a \land (b \lor c) \land (c \lor b) \) is represented by \( \langle \{a\}, \{b,c\} \rangle \), 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 \( D \subseteq A \). A DLP conjunction is a finite sequence \( D \) of DLP disjunctions. A DLP clause is a pair \( (H ; D) \), in which the head \( H = \{a_1, \ldots, a_n\} \) is a DLP disjunction, and the body \( D = \{D_1, \ldots, D_m\} \) 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, a DLP goal. A DLP fact is a bodiless DLP clause. In logic programs, DLP rules will be written as

\[
\begin{align*}
a_1 \lor \cdots \lor a_n & \leftarrow d_1^1 \lor \cdots \lor d_{s_1}^1, \ldots, d_m^n \lor \cdots \lor d_{s_m}^m.
\end{align*}
\]

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:

\[
\begin{align*}
a_1 \lor \cdots \lor a_n & \leftarrow b_1, \ldots, b_m.
\end{align*}
\]

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.

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

\[
P := \{ \begin{array}{l}
p \leftarrow a \\
p \leftarrow b \\
a \lor b \leftarrow \end{array} \}, \quad Q := \{ e \lor p \leftarrow f \lor g , h \} , \quad R := \{ d \leftarrow f \lor h , p \leftarrow g \lor e \}.
\]

The DLP program \( P \) is proper and clean, \( Q \) is proper but not clean, and \( R \) is neither proper nor clean.

**Definition 2.2.** On the set of rules we define two operators head and body as the projections

\[
\begin{align*}
\text{head}((H ; D)) & \triangleq H \\
\text{body}((H ; D)) & \triangleq D.
\end{align*}
\]

**Remark 2.1** (Why propositional?). Following a common practice in the field of semantics for logic programs, we have restricted our study to propositional, but possibly countably infinite programs. Thus, we avoid variables and function
symbols at the cost of finiteness, but this is a fair bargain, as no difficulties arise on the semantics side. To see how we end up with infinite programs, start from a non-propositional, finite program, containing at least one function symbol, and replace each of its rules by all of its ground instantiations. What you get is a countably infinite program with equivalent semantics; see [8] for a relevant discussion.

Remark 2.2 (Disjunctions in bodies). Following [13], a disjunctive clause, is a universally quantified logic formula like

$$\forall (L_1 \lor \cdots \lor L_k),$$

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

$$\forall (a_1 \lor \cdots \lor a_n \lor \neg b_1 \lor \cdots \lor \neg b_m),$$

or, equivalently (by De Morgan) to

$$\forall (a_1 \lor \cdots \lor a_n \lor \neg (b_1 \land \cdots \land b_m)),$$

which is equivalent to the implication

$$\forall (a_1 \lor \cdots \lor a_n \leftarrow (b_1 \land \cdots \land b_m)),$$

and so we adopt it as the logic programming notation of a clause, writing commas instead of $\land$. 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 2.3 ($\hat{P}$ and $\hat{\phi}$). Let $P$ be a general DLP program. Then $\hat{P}$ is the DLP program that results if we replace every unclean rule $\phi = (H, \langle D_1, \ldots, D_n \rangle)$ of $P$ by all rules in

$$\hat{\phi} \equiv \{(H, C) \mid C \in D_1 \times \cdots \times D_n\}.$$

We call $\hat{P}$ the clean version of $P$.

Property 2.1. $\hat{P}$ is clean and logically equivalent to $P$.

This is a simple case of the most general Lloyd–Topor transformation, which transforms logic programs containing arbitrary formulae in their bodies into normal ones. See [12], [11, Chapter 4] or [13, pp. 188–189] for more details.

Example 2.2. Here is the desugaring of $Q$ from the previous example:

$$Q \Rightarrow \begin{cases} e \lor p \leftarrow f \lor g, h \\ p \lor q \leftarrow g, e \lor r \end{cases} \quad \mapsto \quad \hat{Q} \Rightarrow \begin{cases} e \lor p \leftarrow f, h \\ e \lor p \leftarrow g, h \\ p \lor q \leftarrow g, e \\ p \lor q \leftarrow g, r \end{cases}$$
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 both 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 to readability.

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

**Definition 2.4** (Combination of disjunctions). The (disjunctive) combination of two DLP disjunctions is simply their union:

\[ D \sqcup E \equiv D \cup E. \]

**Definition 2.5** (Combination of conjunctions). Given two DLP conjunctions \( \mathcal{D} = \langle D_1, \ldots, D_n \rangle \) and \( \mathcal{E} = \langle E_1, \ldots, E_m \rangle \), their combination is another DLP conjunction, logically equivalent to \( \mathcal{D} \sqcap \mathcal{E} \), and defined by the following equation:

\[ \mathcal{D} \sqcap \mathcal{E} \equiv \langle D_1 \sqcap E_1, \ldots, D_1 \sqcap E_m, \ldots, D_n \sqcap E_1, \ldots, D_n \sqcap E_m \rangle. \]

Notice that the sequence on the right is empty iff any of \( \mathcal{D} \) or \( \mathcal{E} \) is empty.

**Definition 2.6** (Combination of rules). The combination of two DLP rules \( \phi_1 = (H_1, \mathcal{D}_1) \) and \( \phi_2 = (H_2, \mathcal{D}_2) \) is the rule defined by

\[ \phi_1 \sqcap \phi_2 \equiv (H_1 \sqcap H_2, \mathcal{D}_1 \sqcap \mathcal{D}_2). \]

**Remark 2.3.** It follows that \( \phi_1 \sqcap \phi_2 \) will not be a clean rule in general, unless one of the \( \phi_i \)'s is a fact. This is the only construction that won’t preserve cleanliness. But worry not, as we only use it to extract its head (which causes no trouble), not to create program rules.

**Example 2.3.** Combining the following two rules

\[
\begin{align*}
p \lor q & \leftarrow a \lor b, c \quad \text{i.e.,} \quad \langle \{p, q\}, \langle \{a, b\}, \{c\} \rangle \rangle \\
p \lor r & \leftarrow d, e \quad \text{i.e.,} \quad \langle \{p, r\}, \langle \{d\}, \{e\} \rangle \rangle
\end{align*}
\]
we obtain
\[ p \lor q \lor r \iff a \lor b \lor d , a \lor b \lor e , c \lor d , c \lor e , \]
i.e., \( \{ p, q, r \}, \{ a, b, d \}, \{ a, b, e \}, \{ c, d \}, \{ c, e \} \).

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 2.7 (Combining sequences).** Given a sequence \( \langle T_1, \ldots, T_n \rangle \) of DLP disjunctions, conjunctions, or rules, we set
\[
[\langle T_1, \ldots, T_n \rangle] \overset{\text{def}}{=} T_1 \vee \cdots \vee T_n,
\]
where \( \vee \) is understood to associate to the left, and \( \emptyset, \langle \emptyset \rangle \), or \( \langle \emptyset, \langle \emptyset \rangle \rangle \) is its unit, depending on whether we are combining DLP disjunctions, conjunctions, or rules respectively.

An alternative presentation of the same definition uses recursion:
\[
[\langle \rangle] \overset{\text{def}}{=} \begin{cases} \emptyset & \text{(DLP disjunctions)} \\ \langle \emptyset \rangle & \text{(DLP conjunctions)} \\ \langle \emptyset, \langle \emptyset \rangle \rangle & \text{(DLP rules)} \end{cases}
\]
\[
[\langle T_1, \ldots, T_{n+1} \rangle] \overset{\text{def}}{=} [\langle T_1, \ldots, T_n \rangle] \vee T_{n+1}.
\]

### 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 2.8 (Rule restriction).** For any given rule \( \phi \) and any set of atoms \( A \), we define the *restriction* of \( \phi \) to \( A \) by
\[
\phi|_A \overset{\text{def}}{=} (\text{head}(\phi) \cap A, \text{body}(\phi)).
\]
It follows that a rule’s body is unaffected by restriction: \( \text{body}(\phi) = \text{body}(\phi|_A) \).

**Definition 2.9 (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 \( \phi \) of \( \mathcal{P} \) to \( A \):
\[
\mathcal{P}|_A^\phi \overset{\text{def}}{=} (\mathcal{P} \setminus \{ \phi \}) \cup \{ \phi|_A \}.
\]
In words, \( \mathcal{P}|_A^\phi \) is the program which is identical to \( \mathcal{P} \), with the exception that the rule \( \phi \) has been replaced by \( \phi|_A \).
Example 2.4. Let $\mathcal{P}$ be the proper DLP program

$$
\mathcal{P} := \left\{ p \lor q \lor r \leftarrow f, g, \\
p \lor q \lor r \leftarrow a, c, \\
f \leftarrow \right\},
$$

and let $\phi := p \lor q \lor r \leftarrow a, c$. Here are two restrictions of $\mathcal{P}$ wrt $\phi$:

$$
\mathcal{P}_{\{p,q\}}^{\phi} := \left\{ p \lor q \lor r \leftarrow f, g, \\
p \lor q \leftarrow a, c, \\
f \leftarrow \right\}, \quad \mathcal{P}_{\{r\}}^{\phi} := \left\{ p \lor q \lor r \leftarrow f, g, \\
r \leftarrow a, c, \\
f \leftarrow \right\}. 
$$

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

**Definition 2.10** (Proper partition). Suppose that $H$ is a set of atoms. Then a tuple $H := (H_1, \ldots, H_n)$ is a proper partition of $H$ iff

(i) $\emptyset \neq H_i \subseteq H$ 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, splitting wrt some partition $H$ is trivial:

**Definition 2.11** (Splitting a program). Let $\phi$ be a proper DLP rule, and let $H := (H_1, \ldots, H_n)$ be a proper partition of its head. Then the splitting of $\mathcal{P}$ wrt $\phi$ over $H$ is the tuple

$$
(\mathcal{P}_{H_1}^{\phi}, \ldots, \mathcal{P}_{H_n}^{\phi}).
$$

Example 2.5. The pair $(\mathcal{P}^{\phi}_{\{p,q\}}, \mathcal{P}^{\phi}_{\{r\}})$ of Example 2.4 is the splitting of $\mathcal{P}$ wrt $\phi$ over $H := (\{p,q\}, \{r\})$.

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

3. Denotational semantics

In this section we define the denotational or declarative semantics for disjunctive logic programs: we present the minimal model semantics of Minker (see [14] or [13]), which is the de facto denotational semantics for disjunctive logic programs, and the criterion for soundness and completeness for 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 [11] for more information.
**Denotational semantics of LP—The least Herbrand model**

Given a propositional logic program \( \mathcal{P} \), we define its Herbrand base \( \text{HB}(\mathcal{P}) \) to be the set of all atoms that appear in \( \mathcal{P} \). By a Herbrand interpretation \( \mathcal{I} \) of \( \mathcal{P} \), we mean any assignment of truth values \( (\top, \bot) \) to the elements of the Herbrand base. We say that \( \mathcal{I} \) satisfies a rule \( p \leftarrow a_1, \ldots, a_n \), if it satisfies the logic formula \( a_1 \land \cdots \land a_n \rightarrow p \). 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 \( \mathcal{I} \),

\[
a \in \mathcal{I} \iff \mathcal{I}(a) = \top.
\]

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 \( \top \) to every element of the Herbrand base). Therefore the family of all Herbrand models \( \text{HM}(\mathcal{P}) \) is always non-empty, which allows us to define the least Herbrand model as the intersection of this family:

\[
\text{LHM}(\mathcal{P}) \equiv \bigcap \text{HM}(\mathcal{P}).
\]

We use it to provide the declarative semantics of \( \mathcal{P} \):

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

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

**Denotational semantics of DLP—Minimal model semantics**

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

\[
\mathcal{P} := \left\{ \begin{array}{c} p \leftarrow a \\ p \leftarrow b \\ a \lor b \leftarrow \end{array} \right. \right..
\]

First, we compute its Herbrand models:

\( \{a, p\}, \{b, p\}, \{a, b, p\} \).

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 miserably in the presence of disjunctions:

\[
\bigcap \text{HM}(\mathcal{P}) = \bigcap \{\{a, p\}, \{b, p\}, \{a, b, p\}\} = \{p\}.
\]

11
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 of minimal models, viz. \( \{\{a, p\}, \{b, p\}\} \), to obtain semantics for \( \mathcal{P} \).

Denote by \( \text{MM}(\mathcal{P}) \) the set of its minimal models. By the definition that follows, the set \( \text{MM}(\mathcal{P}) \) provides 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 model of \( \mathcal{P} \).

**Example 3.1.** Consider the DLP program

\[
\mathcal{Q} := \begin{cases} 
  p \leftarrow a \\
  p \leftarrow b \\
  b \leftarrow c \\
  a \lor c \leftarrow 
\end{cases}
\]

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

\[
\text{HM}(\mathcal{Q}) = \{ \{a, p\}, \{a, b, p\}, \{c, b, p\}, \{a, b, c, p\} \},
\]

\[
\text{MM}(\mathcal{Q}) = \{ \{a, p\}, \{c, b, p\} \}.
\]

Under the minimal model semantics, \( p \) and \( a \lor c \) are both \( T \), as

\[
\{a, p\} \models p \quad \text{and} \quad \{a, p\} \models a \lor c
\]

\[
\{c, b, p\} \models p \quad \text{and} \quad \{c, b, p\} \models a \lor c,
\]

while both \( a \) and \( b \lor c \) are \( F \) because of

\[
\{c, b, p\} \not\models a \quad \text{and} \quad \{a, p\} \not\models b \lor c.
\]

**Remark 3.1.** In [13], 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.

Readily, the definition of rule restriction implies the following property:

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

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 [13], whose treatment far exceeds our needs.
4. Games

Game semantics made their debut on the logic programming scene with [6], in which a game was defined for the case of LP. This game—or, to be fair, its propositional version—was extended in [9] to cover negation. In this section we extend it in the other direction, defining games for disjunctive logic programs. First we outline 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 proceed to make things formal.3

4.1. An outline of the propositional LP game

The general picture, common in all such games for logic programming, is that two players (Doubter vs Believer) argue over whether a given goal \( p \) succeeds or not. 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 example, 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 \( \beta \) and doubter moves by \( \delta \).

4.2. The simplified DLP game

This game extends the one explained 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 use more than one rule to support his belief. What is more, he can use not

---

3This simplified game is based on a corrected version of the DLP game described in [17].
4We 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.
only rules from the given program, but also implicit rules, i.e., rules of the form \( a \leftarrow a \). Thanks to rule combination, he can disjunctively combine his selection of rules into one, say \( G \leftarrow D_1, \ldots, D_n \), and play it against his opponent:

Believer: “\( G \) because \( D_1, D_2, \ldots, D_n \).”

and it is now Doubter’s turn to choose a disjunction \( D_i \) to doubt:

Doubter: “Why \( D_i \)?”

And the game goes on.

To sum up, Believer is constantly challenged by Doubter to justify why he believes some disjunction \( \delta \). He does that by \( \gamma \)-combining a sequence of DLP rules to form a single rule \( \beta \), such that the head of \( \beta \) is a subset of \( \delta \). Doubter must then select which disjunction from the body of \( \beta \) she doubts, and so on.

The somewhat arbitrary decision to allow the believer to include implicit rules not from the program is backed up by the following example:

\[ \text{Example 4.1. Consider the DLP program} \]

\[ p \leftarrow a \]
\[ p \leftarrow b \]
\[ b \leftarrow c \]
\[ a \lor c \leftarrow \]

For the goal \( p \), two plays in this game could look like this:

\[ \pi_1 \]

\[ \pi_2 \]

In both plays, Believer justifies the move \( B_0 \) by combining the first two program rules. Then, in \( \pi_1 \), he combines the program rule \( b \leftarrow c \) with the implicit rule \( a \leftarrow a \), while in \( \pi_2 \), he only uses program rules. You can easily verify that without implicit rules it is impossible for him to win this game.

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

\[ \text{Example 4.1. Consider the DLP program} \]

\[ p \leftarrow a \]
\[ p \leftarrow b \]
\[ b \leftarrow c \]
\[ a \lor c \leftarrow \]

For the goal \( p \), two plays in this game could look like this:

\[ \pi_1 \]

\[ \pi_2 \]

In both plays, Believer justifies the move \( B_0 \) by combining the first two program rules. Then, in \( \pi_1 \), he combines the program rule \( b \leftarrow c \) with the implicit rule \( a \leftarrow a \), while in \( \pi_2 \), he only uses program rules. You can easily verify that without implicit rules it is impossible for him to win this game.

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

---

5The need for implicit rules was Olivier Laurent’s objection to the correctness of the DLP game as I describe it in [17], according to which the believer can only combine program rules. I am grateful to him for spotting this mistake.
that correspond to doubter 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

\[
\mathcal{Q} := \left\{ \begin{array}{c}
p \leftarrow a, b \\
q \leftarrow a, b \\
a \leftarrow d, c \\
b \leftarrow \\
c \lor d \leftarrow b
\end{array} \right\}
\]

and the goal is \( \leftarrow p \lor q \). Here are two valid plays for this game:

\[
\pi_1 := \begin{array}{c}
\text{goal} : \quad \leftarrow p \lor q \\
B_0 : \quad p \lor q \leftarrow a \lor a, a \lor b, a \lor b, b \lor b \\
B_1 : \quad b \leftarrow
\end{array}
\]

won by Believer, and

\[
\pi_2 := \begin{array}{c}
\text{goal} : \quad \leftarrow p \lor q \\
B_0 : \quad p \lor q \leftarrow a \lor a, a \lor b, a \lor b, b \lor b \\
B_1 : \quad a \leftarrow d, c
\end{array}
\]

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 \( \beta \)—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 \( \leftarrow G \), we will define the associated DLP game, and denote it by \( \Gamma_{\mathcal{P}}(\leftarrow G) \). It will always be a two-player game (Doubter vs Believer). There are two *player roles*: the doubter and the believer.
The statement is read as . . .

| D   | “Why D?” or “I doubt D.” |
| E ← | “E because D₁, . . . , and Dₙ.” |
| E ← | “E because it is a fact.” |

Table 2: How to read believer and doubter statements.

The roles of the players never change throughout the game; this is why we simply call the players Doubter and Believer.

Doubter starts by doubting every disjunct in \( G \), 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. 22) 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 \( P \), we can extended it to \( P_+ \), which includes all meaningful implicit rules. In detail, \( P_+ \) is defined as:

\[
P_+ = P \cup \{ a \leftarrow a \mid a \in \text{HB}(P) \} .
\]

**Definition 4.2 (Moves).** A doubter move \( \delta \) is a sequence of occurrences of disjunctions in bodies of DLP rules. We refer to these occurrences as the doubts of \( \delta \). A believer move \( \beta \) from \( P \) is a finite sequence of DLP rules from the extended set \( P_+ \), such that \( \beta \cap P \neq \emptyset \). 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 \( \beta \) that belong to \( P \) constitute the proper part of the justification; the rest, the implicit. If \(|\beta| > 1\) we call the move 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 dialogue from \( P \) is a finite sequence

\[
\pi := (\delta_0, \beta_0, \delta_1, \beta_1, \ldots) ,
\]

such that:

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

---

6For the LPN game, this is not the case: to capture the third, “unknown” truth value of the well-founded model, we need to allow ties in the game, and therefore, rôle-switching moves. See [9] for more information on this.
Remark 4.1. Regarding the restriction imposed on believer moves, one could make the seemingly stronger demand that $\text{head}([\delta_i]) = [\delta_i]$. 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 play of $\mathcal{P} \leftarrow \mathcal{G}$ is a dialogue $\pi$ from $\mathcal{P}$ which, if non-empty, satisfies the additional property that $[\delta_0] = \mathcal{G}$. We denote the empty play by $\varepsilon = \langle \rangle$.

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

Example 4.3. Consider the same program $\Omega$ as in Example 4.2, repeated here for convenience:

\[
\Omega := \{ \begin{array}{l}
  p \leftarrow a, b \\
  q \leftarrow a, b \\
  a \leftarrow d, c \\
  b \leftarrow \\
  c \lor d \leftarrow b
\end{array} \}
\]

(the goal is still $\leftarrow p \lor q$). 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.

$\pi_1 :=$

\[
\begin{array}{l}
  \text{goal : } \leftarrow p \lor q \\
  \beta_0 : \quad p \leftarrow a, b \\
  \quad q \leftarrow a, b \\
  \quad [\beta_0] : \quad p \lor q \leftarrow a \lor a, a \lor b, b \lor a, b \lor b \\
  \beta_1 : \quad b \leftarrow \\
\end{array}
\]

(still) won by Believer, and

$\pi_2 :=$

\[
\begin{array}{l}
  \text{goal : } \leftarrow p \lor q \\
  \beta_0 : \quad p \leftarrow a, b \\
  \quad q \leftarrow a, b \\
  \quad [\beta_0] : \quad p \lor q \leftarrow a \lor a, a \lor b, b \lor a, b \lor b \\
  \beta_1 : \quad a \leftarrow d, c
\end{array}
\]

by Doubter.

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

17
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: introducing plays in which stalling is allowed 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 $\beta$, i.e., if for every rule $\psi \in \beta$, $|\text{body}(\psi)| = 1$. We denote such a follow move by $\overline{\beta}$. In symbols,

$$\overline{\beta} \overset{\text{df}}{=} \{ \mathcal{D} \in \text{body}(\psi) \mid \psi \in \beta \}.$$

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

$$\overline{\delta} \overset{\text{df}}{=} \{ a \leftarrow a \mid a \in [\delta] \}.$$

In this way he is forcing the doubter to follow with $\overline{\delta}$, which has the same doubts as $\delta$ 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 $\delta$ since $\text{head}([\overline{\delta}]) = [\delta]$.

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:

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

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

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

**Definition 4.7 (Quasi-).** A quasidialogue is a dialogue in which stalling is allowed; likewise, a quasiplay is a play in which stalling is allowed.

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 $\text{rmstall}$ (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 4.8 (rmstall).** Let $\tau$ be a quasidialogue from $\mathcal{P}$. We define the function $\text{rmstall}$ recursively by cases on $\ell := |\tau|$:  

18
Program 1: The definition of rmstall, Haskell-style.

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

\[
\text{rmstall}(\tau) = \tau.
\]

Case 2: \( \ell = 2 \). Then \( \tau = \langle \delta, \beta \rangle \), so we set

\[
\text{rmstall}(\tau) = \begin{cases} 
\langle \delta \rangle & \text{if } \beta = \overline{\delta}, \\
\langle \delta, \beta \rangle & \text{otherwise}.
\end{cases}
\]

Case 3: \( \ell \geq 3 \). Then \( \tau = \langle \delta, \beta, \delta' \rangle + \tau' \), and we need to recurse:

\[
\text{rmstall}(\tau) = \begin{cases} 
\text{rmstall}(\delta : \tau') & \text{if } \beta = \overline{\delta}, \\
\langle \delta, \beta \rangle + \text{rmstall}(\delta' : \tau') & \text{otherwise}.
\end{cases}
\]

Readers fluent in functional programming languages might find the Haskell code of Program 1 easier to digest.\(^7\) It is of course, the same definition.

**Property 4.2** (Validity of rmstall). If \( \tau \) is a quasidialogue from a program, then \( \text{rmstall}(\tau) \) is a dialogue from the same program; and if \( \pi \) is a quasiplay in some game, then \( \text{rmstall}(\pi) \) is a play in the same game.

**Property 4.3** (rmstall is \( \sqsubseteq \)-monotone).

\[
\tau \sqsubseteq \tau' \implies \text{rmstall}(\tau) \sqsubseteq \text{rmstall}(\tau').
\]

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 4.9** (Strategy). A strategy \( \sigma \) in \( \Gamma_\mathcal{P}(\leftarrow \mathcal{G}) \) is a set of plays in \( \Gamma_\mathcal{P}(\leftarrow \mathcal{G}) \), such that:

\(^7\)Apologies to the logic programming community for this tiny treachery.
(i) $\sigma \neq \emptyset$;

(ii) every play in $\sigma$ has even length;

(iii) $\sigma$ is closed under even prefixes (and therefore always contains $\varepsilon$):

$$\pi' \sqsubseteq_e \pi \in \sigma \implies \pi' \in \sigma;$$

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

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

**Remark 4.5 (Strategies as posets).** A strategy $\sigma$ can be seen as a poset $(\sigma, \sqsubseteq_e)$ of even-length plays with a bottom element $\bot = \varepsilon$.

**Definition 4.10 (Total strategy).** A strategy $\sigma$ is called total, if for every play $\pi \in \sigma$ and every doubter response to it $\delta$, there is at least one move $\beta$ such that $\pi + \langle \delta, \beta \rangle$ is also in $\sigma$.

Siding with Believer, we give the following definition:

**Definition 4.11 (Winning strategy).** A strategy $\sigma$ 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 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 $\sqsubseteq_e$-chains).

On the other hand, if we do not close under limits, we will end up distinguishing between strategies like $\sigma$ and $\sigma_\infty$, where:

$$\sigma := \{ \langle \delta_0, \beta_0, \ldots, \delta_n, \beta_n \mid n \in \omega \} ,$$

$$\sigma_\infty := \sigma \cup \{ \sqcup \sigma \} .$$

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 game semantics of disjunctive logic programs.
Definition 4.12 (answer). Given a play $\pi$ and a strategy $\sigma$ in some game $\Gamma_P(\leftarrow G)$, the play $\text{answer}_\sigma(\pi) \in \sigma$ will be:

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

This is a well-defined partial function because $\sigma$ is deterministic, and so in the second branch, if such a $\beta$ exists, it is necessarily unique. Furthermore, if $\sigma$ is total, $\text{answer}_\sigma(\pi)$ 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 $\text{answer}_\sigma$ behaves like a closure operator; i.e., whenever it is defined it satisfies:

- $\pi \subseteq \text{answer}_\sigma(\pi)$ (extensive),
- $\pi \subseteq \pi' \implies \text{answer}_\sigma(\pi) \subseteq \text{answer}_\sigma(\pi')$ (monotone),
- $\text{answer}_\sigma(\text{answer}_\sigma(\pi)) = \text{answer}_\sigma(\pi)$ (idempotent).

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 $D \lor E \equiv D \land \neg E$ and of the following proposition:

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

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

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 semantics for disjunctive logic programs; in other words, to decide which goals succeed. However, depending on the cleverness of the players involved, for the same program $P$ and the same goal $\leftarrow G$, Believer might win one time and lose another. Therefore it makes no sense to determine the success of a goal by looking at a single play. Instead we say:

Definition 4.13 (DLP game semantics). Let $P$ be a DLP program. A goal $\leftarrow G$ succeeds, if Believer has a winning strategy in $\Gamma_P(\leftarrow G)$. 

21
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:

**Example 4.4.** Consider the following program

\[
\mathcal{P} := \{ \texttt{a \leftarrow a, b} \}
\]

and the goal \( \texttt{a} \). A play in \( \Gamma_\mathcal{P}(\leftarrow \texttt{a}) \) has to begin with Doubter doubting \( \texttt{a} \), to which Believer responds with \( \texttt{a \leftarrow a, b} \). For as long as Doubter doubts \( \texttt{a} \), 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 \( \texttt{b} \), at which point Believer would win by playing the fact \( \texttt{b \leftarrow} \). This describes a winning strategy for Believer, and—this being a logic program—we most definitely do not want \( \texttt{a} \) to be derivable from \( \mathcal{P} \).

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;

\[8\] 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 [15] and [14].

**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.\[9\] However, 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 when the program is not disjunctive.

---

8This would introduce strategies that are neither winning nor losing: they lead to ties.

9The 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. It cannot happen in an LP program.
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 \( \gamma \) and \(| \cdot |\) 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, \( \leftarrow \mathcal{G} \) a goal clause, and \( \phi \) a proper DLP rule of \( \mathcal{P} \); \( \mathcal{H} := (H_1, H_2) \) will be a proper partition of \( \mathcal{H} := \text{head}(\phi) \), and \( (\mathcal{P}_1, \mathcal{P}_2) \) the corresponding splitting of \( \mathcal{P} \) wrt \( \phi \) over \( \mathcal{H} \). Naturally we will write just \( \phi_1 \) and \( \phi_2 \) for the rules \( \phi|_{H_1} \) and \( \phi|_{H_2} \) respectively. The variable \( q \) ranges over the size of the partition: \( q = 1, 2 \). We use \( \pi \) and \( \tau \) for dialogues or plays of \( \Gamma_{\mathcal{P}}(\leftarrow \mathcal{G}) \) and \( \Gamma_{\mathcal{P}_q}(\leftarrow \mathcal{G}) \) respectively; \( \pi \) and \( \tau \) for “quasi-”.

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 \( \beta \) of \( \Gamma_{\mathcal{P}_q}(\leftarrow \mathcal{G}) \) is called **forbidden** in \( \Gamma_{\mathcal{P}}(\leftarrow \mathcal{G}) \) iff it includes \( \phi_q \). In symbols,

\[
\text{Forbidden}_q(\beta) \iff \phi_q \in \beta.
\]

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

\[\text{There is nothing special about the number 2 here, and everything that we develop can be stated } \text{mutatis mutandis} \text{ for the case in which } |\mathcal{H}| \text{ is any larger number. However, 2 is as large as we need.}\]
Property 5.1. The release of a move satisfies the following properties:

(i) $\beta^q_i = \phi \iff \phi \notin \beta^q_i$;

(ii) $\text{body}(\beta^q_i) = \text{body}(\beta^q_i)$.

Proposition 5.2. Let $\beta_1$ and $\beta_2$ be two believer moves from $P_1$ and $P_2$ respectively, and let $\beta := \beta_1^* + \beta_2^*$. Then

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

Proof. We compute

$$\text{head}([\beta_1]) \cup \text{head}([\beta_2]) = \left( \bigcup_{\psi \in \beta_1} \text{head}(\psi) \right) \cup \left( \bigcup_{\psi \in \beta_2} \text{head}(\psi) \right)$$

$$= \bigcup_{\psi \in \beta_1 + \beta_2} \text{head}(\psi)$$

$$= \text{head}([\beta_1 + \beta_2])$$

$$= \text{head}([\beta_1^* + \beta_2^*])$$

$$= \text{head}([\beta]),$$

where the starred equality holds since

$$\text{head}(\phi_1) \cup \text{head}(\phi_2) = H_1 \cup H_2 = \text{head}(\phi).$$

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

Combining synchronous plays

The three games $\Gamma_P(\leftarrow G)$, $\Gamma_{P_1}(\leftarrow G)$, and $\Gamma_{P_2}(\leftarrow G)$ would be identical except that $\phi$ can only be part of believer moves of $\Gamma_P(\leftarrow G)$, $\phi_1$ of those of $\Gamma_{P_1}(\leftarrow G)$, and $\phi_2$ of those of $\Gamma_{P_2}(\leftarrow G)$. Since moves are sequences, it would be delightful to simply concatenate the moves of $\pi_1$ with those of $\pi_2$ to obtain a play in $\Gamma_P(\leftarrow G)$. Doubter moves are no obstacles to this plan, but believer moves can be troublesome: they may contain the rules $\phi$, which are not allowed in $\Gamma_P(\leftarrow G)$. This problem is easy to solve if both moves include their forbidden rules: we simply replace each of them by $\phi$—a reasonable action, since $\phi \equiv \phi_1 \lor \phi_2$. To deal with this case, we begin with a key definition.

Definition 5.3 (Synchronous). Two quasiplays $\tau_1$ and $\tau_2$ in $\Gamma_{P_1}(\leftarrow G)$ and $\Gamma_{P_2}(\leftarrow G)$ respectively are called synchronous (or in sync with each other) if both Believers use their forbidden rules in the exact same turns. In symbols,

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

Property 5.3. If $\tau_1$ is in sync with $\tau_2$, then so is any prefix of $\tau_1$ with any prefix of $\tau_2$. 

24
Let $\pi_1$ and $\pi_2$ be plays of even length in the games $\Gamma_{\mathcal{P}_1}(\leftarrow G)$ and $\Gamma_{\mathcal{P}_2}(\leftarrow G)$ respectively, and suppose that the two plays are synchronous. We will describe a new play $\pi_1 \tilde{\circ} \pi_2$, the synchronous combination of $\pi_1$ and $\pi_2$ wrt $\phi$ 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

$$\pi_1 =: \langle \delta_{0}^{1}, \beta_{0}^{1}, \delta_{1}^{1}, \beta_{1}^{1}, \ldots \rangle$$

$$\pi_2 =: \langle \delta_{0}^{2}, \beta_{0}^{2}, \delta_{1}^{2}, \beta_{1}^{2}, \ldots \rangle$$

be the two given plays. We set

$$\pi_1 \tilde{\circ} \pi_2 \overset{\text{df}}{=} \langle \delta_{0}, \beta_{0}, \delta_{1}, \beta_{1}, \ldots \rangle,$$

where the symbols are defined as follows.

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

$$\delta_{0} := \delta_{0}^{1} \downarrow \delta_{0}^{2},$$

Believer replies with

$$\beta_{0} := \beta_{0}^{1} \downarrow \beta_{0}^{2},$$

a valid move since $\text{head}([\beta_{0}]) \subseteq [\delta_{0}]$ (Proposition 5.2). Doubter must now select one occurrence from each rule-body in $\beta_{0}$, and the moves $\delta_{1}^{1}$ and $\delta_{1}^{2}$ provide just that:

$$\delta_{1} := \delta_{1}^{1} \downarrow \delta_{1}^{2}.$$ 

The game goes on until the turn $i$ in which the believers of $\pi_1$ and $\pi_2$ both play their forbidden rules $\phi_1$ and $\phi_2$ in their justifications. At this point, it is of course Believer’s turn in the combined play $\pi_1 \tilde{\circ} \pi_2$. We set his move to be

$$\beta_{i} := \beta_{i}^{1*} \downarrow \beta_{i}^{2*},$$

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

$$\delta_{i+1} := \delta_{i+1}^{1} \downarrow \delta_{i+1}^{2}.$$ 

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 $\pi_1 \tilde{\circ} \pi_2$ has in her possession two doubter moves $\delta_{k}^{1}$ and $\delta_{k}^{2}$ from $\pi_1$ and $\pi_2$ 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

\[
\tau_1 := \langle \delta_0^1, \beta_0^1, \delta_1^1, \beta_1^1, \ldots \rangle \text{ of } \Gamma_{\mathcal{P}_1}(\leftarrow \mathcal{G}),
\]

\[
\tau_2 := \langle \delta_0^2, \beta_0^2, \delta_1^2, \beta_1^2, \ldots \rangle \text{ of } \Gamma_{\mathcal{P}_2}(\leftarrow \mathcal{G}),
\]

their synchronous combination \( \tilde{\tau}_1 \tilde{\tau}_2 \) over \( \phi \) wrt \( \mathcal{H} \) is the sequence

\[
\tau_1 \tilde{\tau}_2 = \langle \delta_0, \beta_0, \delta_1, \beta_1, \ldots \rangle
\]

of length \( |\tilde{\tau}_1 \tilde{\tau}_2| = \min \{|\tau_1|, |\tau_2|\} \), where the symbols involved are defined by

\[
\delta_i := \delta_i^1 + \delta_i^2,
\]

\[
\beta_i := \beta_i^1 + \beta_i^2.
\]

**Proposition 5.4 (Validity of \( \tilde{\tau} \)).** Given two synchronous quasiplays \( \tau_1 \) and \( \tau_2 \) of \( \Gamma_{\mathcal{P}_1}(\leftarrow \mathcal{G}) \) and \( \Gamma_{\mathcal{P}_2}(\leftarrow \mathcal{G}) \) respectively, \( \tau := \tau_1 \tilde{\tau}_2 \) is a quasiplay in \( \Gamma_{\mathcal{P}}(\leftarrow \mathcal{G}) \).

It follows that if \( \tau_1 \) and \( \tau_2 \) do not stall simultaneously, \( \tau \) will be a play of the game \( \Gamma_{\mathcal{P}}(\leftarrow \mathcal{G}) \).

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

\[
\text{head}(\beta_i) \subseteq [\delta_i],
\]

\[
\text{head}(\beta_i) \subseteq [\delta_i],
\]

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

\[
\text{head}(\beta_i) = \text{head}(\delta_i) \cup \text{head}(\delta_i) \subseteq [\delta_i] \cup [\delta_i] = [\delta_i + \delta_i] = [\delta_i],
\]

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. 25). To verify that \( \tau \) is indeed a quasiplay, observe that both \( \tau_1 \) and \( \tau_2 \) share the same goal with \( \tau \), and so \( \delta_0 = \delta_0^1 + \delta_0^2 \) is a correct first move for a quasidialogue from \( \mathcal{P} \) to be a quasiplay of \( \Gamma_{\mathcal{P}}(\leftarrow \mathcal{G}) \).

For the second claim, observe that a stalling move in \( \tau_1 \tilde{\tau}_2 \) would imply the existence of two simultaneously played stalling moves, one in \( \tau_1 \) and one in \( \tau_2 \), 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 \( \tilde{\tau} \) immediately yields a couple of more preservation properties:

**Property 5.6 (Preservation of parity).** Let \( \tau_1 \) and \( \tau_2 \) be two synchronous quasiplays, both of even length. Then \( \tau_1 \tilde{\tau}_2 \) will also have even length.

**Property 5.7 (\( \tilde{\tau} \) is monotone).** Let \( \tau_1 \) and \( \tau_2 \) be two synchronous quasiplays.

Then for any \( \tau_1' \subseteq \tau_1 \) and any \( \tau_2' \subseteq \tau_2 \) we have \( \tau_1' \tilde{\tau}_2' \subseteq \tau_1 \tilde{\tau}_2 \).

So far, so good. Not every pair of plays is synchronous though:
Combining arbitrary plays

Starting with two arbitrary quasiplays \( \tau_1 \) and \( \tau_2 \), we build two new, synchronized quasiplays \( \hat{\tau}_1 \) and \( \hat{\tau}_2 \) 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 \( \hat{\tau}_1 \) with \( \hat{\tau}_2 \). It is exactly this idea that we exploit to extend the definition of \( \tau_1 \bowtie \tau_2 \) to cover asynchronous plays: **synchronize first, then combine.**

**Definition 5.5** (Synchronization). Let \( \tau_1 \) and \( \tau_2 \) be two quasidialogues from \( \mathcal{P}_1 \) and \( \mathcal{P}_2 \) respectively, and let \( \ell_1 \) and \( \ell_2 \) be their lengths. We recursively define their **synchronization** \( \text{sync}(\tau_1, \tau_2) \) 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:

\[
\text{sync}(\tau_1, \tau_2) \equiv (\tau_1, \tau_2).
\]

**Case 2:** \( \ell \geq 2 \). Then \( \tau_1 =: (\delta_1, \beta_1) + \tau'_1 \) and \( \tau_2 =: (\delta_2, \beta_2) + \tau'_2 \), and we set:

\[
\text{sync}(\tau_1, \tau_2) = \begin{cases} 
(\langle \delta_1, \beta_1 \rangle + \text{rec}_1, \langle \delta_2, \beta_2 \rangle + \text{rec}_2) & \text{if (a),} \\
(\langle \delta_1, \beta_1 \rangle + \text{rec}_1, \langle \delta_2, \beta_2 \rangle + \text{rec}_2) & \text{if (b),} \\
(\langle \delta_1, \beta_1 \rangle + \text{rec}_1, \langle \delta_2, \beta_2 \rangle + \text{rec}_2) & \text{if (c),}
\end{cases}
\]

where \( \text{rec}_1 \) and \( \text{rec}_2 \) wrap the recursive calls

\[
\text{(rec}_1, \text{rec}_2) := \begin{cases} 
\text{sync}(\delta_1 = \beta_2 := \tau'_1, \tau'_2) & \text{if (a),} \\
\text{sync}(\tau'_1, \delta_1 = \beta_2 := \tau'_2) & \text{if (b),} \\
\text{sync}(\tau'_1, \tau'_2) & \text{if (c),}
\end{cases}
\]

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.

The mathematical definition above is admittedly a bit convoluted to write down formally, albeit easy to describe. Once again we provide the same definition in Haskell code in Program 2.

**Remark 5.2** (Dependencies). Even though we have not incorporated \( \phi_1 \) and \( \phi_2 \) into the symbol \( \text{sync} \) 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 \( \phi, \phi_1, \) and \( \phi_2 \) into our notation, however, we allow ourselves to simply use \( \text{sync} \) instead of an excessively precise name like \( \text{sync}_{\phi_1, \phi_2} \). The same is true for various symbols that we use: e.g., \( \gamma, \gamma^* \), etc.

**Proposition 5.8.** Synchronization is a well-defined, total operation.
| case_a = (d1 : stall d1 : fst rec, d2 : b2 : snd rec)  
| case_b = (d1 : b1 : fst rec, d2 : stall d2 : snd rec)  
| case_c = (d1 : b1 : fst rec, d2 : b2 : snd rec)  

where  

case_a = forbidden1 b1 && not (forbidden2 b2)  
case_b = forbidden2 b2 && not (forbidden1 b1)  
case_c = otherwise  

rec  
| case_a = sync (phi1, phi2) (fixocc d1 : b1 : ms1, ms2)  
| case_b = sync (phi1, phi2) (ms1, fixocc d2 : b2 : ms2)  
| case_c = sync (phi1, phi2) (ms1, ms2)  

forbidden1 = elem phi1  
forbidden2 = elem phi2  

stall ds = [ (d, [d]) | d <- ds ]  
follow = map (head . snd)  
fixocc = follow . stall  

sync (phi1, phi2) (tau1, tau2) = (tau1, tau2)  

Program 2: The definition of sync, Haskell-style.

Proof. Observe that on every recursive call of sync both of its arguments are themselves quasidialogues from \( P_1 \) and \( P_2 \); 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 \((\tau_1, \tau_2) \defeq \text{sync}(\tau_1, \tau_2)\). Then  

(i) the pair \((\tau_1, \tau_2)\) is synchronous;  
(ii) if \((\tau_1, \tau_2)\) happens to be synchronous, then \(\text{sync}(\tau_1, \tau_2) = (\tau_1, \tau_2)\);  
(iii) \(\text{rmstall}(\tau_1) = \text{rmstall}(\tau_2)\);  
(iv) \text{sync} never produces two stalling moves in the same turn.

Proposition 5.10 (Preservation of parity). If \(\tau_1\) and \(\tau_2\) have even lengths, then so does \(\text{sync}(\tau_1, \tau_2)\).
Proof. This is immediate by the very definition of sync, which generates its output \( \text{sync}(\tau_1, \tau_2) \) incrementally by pairs of moves, while consuming pairs of moves from its inputs \( \tau_1 \) and \( \tau_2 \). Thus, if both of its inputs are of even length, the same will be true for its output.

**Proposition 5.11** (sync is monotone). Let \( \tau'_1 \) and \( \tau'_2 \) be two quasidialogues, and let \( \tau_1 \subseteq \tau'_1 \) and \( \tau_2 \subseteq \tau'_2 \). Then \( \text{sync}(\tau_1, \tau_2) \subseteq \text{sync}(\tau'_1, \tau'_2) \). Moreover, if \( \tau_1 \sqcup \tau'_1 \) and \( \tau_2 \sqcup \tau'_2 \), then \( \text{sync}(\tau_1, \tau_2) \sqcup \text{sync}(\tau'_1, \tau'_2) \).

Proof. Just like in the previous proof, we only need to observe how synchronization really works. In fact, the construction of \( \text{sync}(\tau_1, \tau_2) \) must end with a call to case 1 of its definition, with either \( \ell = 0 \) or \( \ell = 1 \). In either case, extending any of \( \tau_1 \) or \( \tau_2 \) 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 \( \tau_1 \) and \( \tau_2 \) be two quasidialogues from \( P_1 \) and \( P_2 \) respectively. We define their combination \( \tau_1 \upharpoonright \tau_2 \) by composition:

\[
\circdf{\gamma} = \gamma \circ \text{sync}.
\]

In other words,

\[
\tau_1 \upharpoonright \tau_2 \circdf{} \tau_1 \tilde{\upharpoonright} \tau_2,
\]

where \( (\tau_1, \tau_2) := \text{sync}(\tau_1, \tau_2) \).

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

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

By defining the general combination as a composition of \( \text{sync} \) and \( \tilde{\gamma} \), we readily get their composable properties as corollaries:

**Corollary 5.12** (Validity of combination). Given two (quasi)dialogues \( \tau_1 \) and \( \tau_2 \) from \( P_1 \) and \( P_2 \) respectively, their disjunctive combination \( \tau_1 \upharpoonright \tau_2 \) is a (quasi)-dialogue from \( P \).

**Corollary 5.13** (Preservation of plays). If \( \tau_1 \) and \( \tau_2 \) are two (quasi)plays in \( \Gamma_{P_1} (\leftarrow G) \) and \( \Gamma_{P_2} (\leftarrow G) \) respectively, then \( \tau_1 \upharpoonright \tau_2 \) is a (quasi)play in \( \Gamma_P (\leftarrow G) \).

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

29
Corollary 5.15 ($g$ is monotone). Let $\pi'_1$ and $\pi'_2$ be two dialogues, and let $\pi_1 \sqsubseteq \pi'_1$ and $\pi_2 \sqsubseteq \pi'_2$. Then $\pi_1 \sqcap \pi_2 \sqsubseteq \pi'_1 \sqcap \pi'_2$. Moreover, if $\pi_1 \sqsubseteq \pi'_1$ and $\pi_2 \sqsubseteq \pi'_2$, then $\pi_1 \sqcap \pi_2 \sqsubseteq \pi'_1 \sqcap \pi'_2$.

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 $\pi_1$ and $\pi_2$ (i.e., Believer cannot play a valid response to either plays) and yet she does not win in the combined play:

$$P := \left\{ \begin{array}{l} p \leftarrow a \\ p \leftarrow b \\ a \lor b \leftarrow \\ c \lor d \leftarrow \end{array} \right\}, \quad P_1 := \left\{ \begin{array}{l} p \leftarrow a \\ p \leftarrow b \\ a \lor b \leftarrow \\ c \leftarrow \end{array} \right\}, \quad P_2 := \left\{ \begin{array}{l} p \leftarrow a \\ p \leftarrow b \\ a \lor b \leftarrow \\ d \leftarrow \end{array} \right\}$$

and the plays

$$\begin{array}{l} \pi_1 \left\{ \begin{array}{l} \text{goal} : \leftarrow p \\ \beta_0^1 : \ p \leftarrow a \end{array} \right\} \quad \pi_2 \left\{ \begin{array}{l} \text{goal} : \leftarrow p \\ \beta_0^2 : \ p \leftarrow b \end{array} \right\} \\ \pi \left\{ \begin{array}{l} \text{goal} : \leftarrow p \\ \beta_0 : \ p \leftarrow a \\ \beta_0 : \ p \leftarrow b \end{array} \right\} \end{array}$$

Notice that Believer cannot move in neither of the starting plays, but he can certainly move in their combination by playing the rule $a \lor b \leftarrow$. However, as we will later see, such misbehaviours are evaded if the plays $\pi_q$ come from a strategy splitting; the impatient can read Corollary 5.23 (p. 36).

Remark 5.6 (Combining goals). When combining plays, their common goal $\leftarrow G$ does not really have to be all that common. We can combine a play in $\Gamma_{\pi_1} (\leftarrow G_1)$ with a play in $\Gamma_{\pi_2} (\leftarrow G_2)$ to get a new play in $\Gamma_{\pi} (\leftarrow G)$, where $G := G_1 \lor G_2$. To do so, we first extract plays in $\Gamma_{\pi_1} (\leftarrow G)$ and $\Gamma_{\pi_2} (\leftarrow G)$ by altering only the first move in each play so that it is restricted to $G_q$; then we combine.

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 $\pi$ is a play and consider the sequence obtained by restricting every believer move in $\pi$ that contains $\phi$ to an identical move in which $\phi$ has been restricted to $H_q$. We denote this sequence by $\pi|^{\phi}_{H_q}$ and call it the restriction of the play $\pi$ to $H_q$ wrt $\phi$.

Theorem 5A. For any play $\pi$ of $\Gamma_{\pi} (\leftarrow G)$, its restriction $\pi_q := \pi|^{\phi}_{H_q}$ is a valid play in $\Gamma_{\pi_q} (\leftarrow G)$. 

30
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 of $\pi$ includes the forbidden rule $\phi$. Denoting by $\beta^q$ the corresponding believer move of $\pi_q$, it is evident that $\text{head}([\beta^q]) \subseteq \text{head}([\beta])$ 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). The play splitting of $\pi$ wrt $\phi$ over a proper partition $(H_1, H_2)$ is the pair

$$(\pi_1, \pi_2) := \left(\pi^\phi|_{H_1}, \pi^\phi|_{H_2}\right).$$

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 in 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 $\sigma_1$ and $\sigma_2$ in $\Gamma_{\pi_1} (\leftarrow G)$ and $\Gamma_{\pi_2} (\leftarrow G)$ respectively, we define their combination

$$\sigma_1 \uplus \sigma_2 := \{ \pi_1 \uplus \pi_2 \mid \pi_1 \in \sigma_1 \text{ and } \pi_2 \in \sigma_2 \}.$$

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

$$C(\pi) \overset{\Delta}{=} \{(\pi_1, \pi_2) \in \sigma_1 \times \sigma_2 \mid \pi_1 \uplus \pi_2 = \pi\}$$

the creators of $\pi$ from $\sigma_1$ and $\sigma_2$. Equipped with the product order induced by either $\subseteq$ or $\subseteq$, the set $C(\pi)$ 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 $\pi$ from $\sigma_1$ and $\sigma_2$. Note that the two orderings $\subseteq$ and $\subseteq$ coincide in this poset since all plays involved come from strategies, and therefore are of even length.

We 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 $\pi \in \sigma := \sigma_1 \uplus \sigma_2$, we can extract in a unique way, two synchronized quasiplays $\tau_1$ and $\tau_2$, each of length $|\pi|$, such that $\tau_1 \uplus \tau_2 = \pi$ and both $\tau_q$ agree with $\sigma_q$, i.e., $\text{rmstall}(\tau_q) \in \sigma_q$.

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

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

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

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

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

$$\tau'_1 := \langle \delta^1_0, \beta^1_0, \ldots, \delta^1_{n-1}, \beta^1_{n-1} \rangle$$
$$\tau'_2 := \langle \delta^2_0, \beta^2_0, \ldots, \delta^2_{n-1}, \beta^2_{n-1} \rangle$$

which combine into $\pi^-$ and agree with the corresponding strategies. Since $\pi$ is a play, the doubter move $\delta_n$ is either the first move ($n = 0$), or it consists of doubts from the bodies of the last believer moves in $\tau'_1$ and $\tau'_2$ ($n > 0$). In the first case, we set $\delta^q_n := \delta_n$, while in the second one, we split it in a unique way in two parts,

$$\delta_n := \delta^1_n \uplus \delta^2_n,$$

such that $\delta^q_n$ is a valid doubter response to $\tau'_q$. We now use these $\delta^q_n$ to obtain the corresponding believer moves $\beta^q_n$. For this we appeal to the fact that $\sigma_q$ (being deterministic) can have at most one next-move for the play $\text{rmstall}(\tau'_q \uplus \langle \delta^q_n \rangle)$, and we know that it has at least one since $\pi \in \sigma_1 \uplus \sigma_2$:

$$\beta^q_n := (\text{answer}_{\sigma_q} \circ \text{rmstall}) (\tau'_q \uplus \langle \delta^q_n \rangle).$$

Note that as this is an answer from $\sigma_q$, the quasiplay $\tau'_q \uplus \langle \delta^q_n, \beta^q_n \rangle$ remains in agreement with it.

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

By their construction, projections satisfy the following property:

Property 5.17. Given any $\pi \in \sigma := \sigma_1 \uplus \sigma_2$,

$$\text{proj}_{\sigma_q}(\pi^-) = \left( \text{proj}_{\sigma_q}(\pi) \right)^-.$$ 

In other words, denoting by $\Gamma_q$ the sets of quasiplays in $\Gamma_{\sigma_q}(\leftarrow G)$, the diagram in Figure 1 commutes.

A link between projections and shortest creators is revealed: starting with $\pi$ as above, first use Lemma 5.16 to obtain the quasiplay projections of $\pi$. Then, using $\text{rmstall}$, 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 \( \sigma_q \). Following these steps, one actually obtains the shortest creators of \( \pi \) from \( \sigma_1 \) and \( \sigma_2 \). This will be clarified shortly in a lemma—but first, a definition that we will need:

**Definition 5.12.** Let \( \pi \in \sigma := \sigma_1 \uplus \sigma_2 \). We define the function \( cr_{\sigma_q} : \sigma \rightarrow \sigma_q \) as the composition

\[
    cr_{\sigma_q} \equiv \text{answer}_{\sigma_q} \circ \text{rmstall} \circ \text{proj}_{\sigma_q},
\]

and simply write

\[
    cr(\pi) \equiv (cr_{\sigma_1}(\pi), cr_{\sigma_2}(\pi))
\]

for the function \( cr : \sigma \rightarrow \sigma_1 \times \sigma_2 \). Both functions will turn out to be total.

**Lemma 5.18.** Let \( \pi \in \sigma := \sigma_1 \uplus \sigma_2 \). Then

\[
    cr(\pi) = \min C(\pi).
\]

**Proof.** Let \( \pi := (\delta_0, \beta_0, \ldots, \delta_n, \beta_n) \in \sigma_1 \uplus \sigma_2 \) and pick any pair of creators \( (\pi_1, \pi_2) \in C(\pi) \), so that \( \pi_1 \uplus \pi_2 = \pi \). We will show that \( cr(\pi) \subseteq (\pi_1, \pi_2) \). Set \( (\tau_1, \tau_2) := \text{sync}(\pi_1, \pi_2) \), and write

\[
    \tau_1 := (\delta_0^1, \beta_0^1, \delta_1^1, \beta_1^1, \ldots, \delta_n^1, \beta_n^1)
\]

\[
    \tau_2 := (\delta_0^2, \beta_0^2, \delta_1^2, \beta_1^2, \ldots, \delta_n^2, \beta_n^2).
\]

By definition, \( \pi = \tau_1 \uplus \tau_2 \), so that

\[
    \pi = \left( \delta_0^1 + \delta_0^2, \beta_0^1 + \beta_0^2, \delta_1^1 + \delta_1^2, \beta_1^1 + \beta_1^2, \ldots, \delta_n^1 + \delta_n^2, \beta_n^1 + \beta_n^2 \right),
\]

where \( n := \min \{n_1, n_2\} \). Now, using the monotonicity of \( \text{rmstall} \) and \( \text{answer}_{\sigma_q} \) (properties 4.3 and 4.4), we reason:

\[
    \text{proj}_{\sigma_q}(\pi) \subseteq \tau_q \Rightarrow \text{rmstall}\left( \text{proj}_{\sigma_q}(\pi) \right) \subseteq \text{rmstall}(\tau_q)
\]

\[
    \Rightarrow \text{answer}_{\sigma_q}\left( \text{rmstall}\left( \text{proj}_{\sigma_q}(\pi) \right) \right) \subseteq \text{answer}_{\sigma_q}\left( \text{rmstall}(\tau_q) \right)
\]

\[
    \Rightarrow cr(\pi) \subseteq \text{answer}_{\sigma_q}\left( \text{rmstall}(\tau_q) \right).
\]
Figure 2a: The penultimate believer move of $\proj_{\sigma_2}(\pi)$ is a stalling.

Notice that $\tau_q$ cannot end in a stalling move, and so $\rmstall(\tau_q)$ will have even length. According to Definition 4.12, we can then expect that

$$\answer_{\sigma_q}(\rmstall(\tau_q)) = \rmstall(\tau_q),$$

so that by using Property 5.9(iii) we derive

$$\text{cr}(\pi) \subseteq \text{e}\ \text{rmstall}(\tau_q) = \pi_q,$$

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 $\varepsilon \neq \pi \in \sigma := \sigma_1 \vee \sigma_2$, and let $\pi_q := \text{cr}_{\sigma_q}(\pi)$. Then

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

depending on whether

(a) the penultimate believer move of $\proj_{\sigma_2}(\pi)$ is a stalling,
(b) the penultimate believer move of $\proj_{\sigma_1}(\pi)$ 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 $\pi_q$ is needed to form the last move of $\pi^-$ iff the penultimate move of $\proj_{\sigma_2}(\pi)$ is a stalling. This holds because once we project a play to $\sigma_q$, stalling moves might appear to the quasiplay projections. If the penultimate move of $\proj_{\sigma_2}(\pi)$ is such a stalling, once we delete the last move from $\pi_q$ to obtain $\pi_q^-$, and use $\rmstall$ to remove all stall–follow moves, the resulting play will be of odd-length. $\answer_{\sigma_q}(\pi)$ will then reproduce the move we deleted, thus bringing us back to $\pi_q$.

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

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

**Corollary 5.20.** Let $\varepsilon \neq \pi \in \sigma := \sigma_1 \vee \sigma_2$. Then
Figure 2b: The penultimate believer move of $\text{proj}_{\sigma_1}(\pi)$ is a stalling.

Figure 2c: Otherwise.

\begin{enumerate}
\item $\min C(\pi^{-}) \subseteq_{e} \min C(\pi)$;
\item $\min C(\pi') \subseteq_{e} \min C(\pi)$, for any $\pi' \subseteq_{e} \pi$.
\end{enumerate}

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

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

**Non-empty.** Since $\sigma_1$ and $\sigma_2$ are strategies, they are both non-empty, and so there is at least one play in $\sigma$: their combination.

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

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

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

\[
\pi = (\delta_0, \beta_0, \ldots, \delta_n, \beta_n)
\]

\[
\tilde{\pi} = (\delta_0, \beta_0, \ldots, \delta_n, \ldots).
\]

For each of them, use Lemma 5.16 to extract its two quasiplay projections on $\sigma_1$ and $\sigma_2$; $(\tau_1, \tau_2)$ from $\pi$ and $(\tilde{\tau}_1, \tilde{\tau}_2)$ from $\tilde{\pi}$.

\[
\tau_1 = \langle \delta_0^1, \beta_0^1, \ldots, \delta_n^1, \beta_n^1 \rangle \\
\tilde{\tau}_1 = \langle \delta_0^1, \beta_0^1, \ldots, \delta_n^1, \beta_n^1 \rangle
\]

\[
\tau_2 = \langle \delta_0^2, \beta_0^2, \ldots, \delta_n^2, \beta_n^2 \rangle \\
\tilde{\tau}_2 = \langle \delta_0^2, \beta_0^2, \ldots, \delta_n^2, \beta_n^2 \rangle,
\]

so that

\[
\pi = \langle \delta_0^1 + \delta_0^2, \beta_0^1 + \beta_0^2, \ldots, \delta_n^1 + \delta_n^2, \beta_n^1 + \beta_n^2 \rangle
\]

\[
\tilde{\pi} = \langle \delta_0^1 + \delta_0^2, \beta_0^1 + \beta_0^2, \ldots, \delta_n^1 + \delta_n^2, \beta_n^1 + \beta_n^2 \rangle.
\]
Now, which of the four statements $\phi \in \beta_n^*$ and $\phi \in \bar{\beta}_n^*$ 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 $\sigma_q$, 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 $\sigma_1 \uplus \sigma_2$ is total if both $\sigma_1$ and $\sigma_2$ 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 $\beta_{n+1}$ such that $\pi \uplus \langle \delta_{n+1}, \beta_{n+1} \rangle \in \sigma$. Use Lemma 5.16 to extract the quasiplay projections $(\tau_1, \tau_2)$ of $\pi$ on $\sigma_1$ and $\sigma_2$, so that $\pi = \tau_1 \uplus \tau_2 =: \langle \delta_0^1 + \delta_0^2, \beta_0^1 + \beta_0^2, \ldots, \delta_n^1 + \delta_n^2, \beta_n^1 + \beta_n^2 \rangle$.

Since $\delta_{n+1}$ is a valid next-move for $\pi$, it consists of doubts from the bodies of $\beta_{n+1}^1$ and $\beta_{n+1}^2$ (Property 5.1(ii)), so that it can be unambiguously split into two sequences of such doubts $\delta_{n+1} =: \delta_{n+1}^1 + \delta_{n+1}^2$. By the totality of $\sigma_q$, there must be at least one believer move $\beta_{n+1}^q$ satisfying $\pi_q^\uplus := \text{rmstall} (\tau_q) \uplus \langle \delta_{n+1}^q, \beta_{n+1}^q \rangle \in \sigma_q$.

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

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

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

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

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

5.2.2. Restricting and splitting strategies

**Definition 5.13** (Strategy restriction). Let $\sigma$ be a strategy in $\Gamma_P (\leftarrow G)$. Then the restriction of $\sigma$ wrt $\phi$ on $H_q$, is the set of plays defined by:

$$\sigma|_{H_q}^\phi \equiv \left\{ \pi|_{H_q}^\phi \mid \pi \in \sigma \right\}.$$  

**Theorem 5C.** The set of plays $\sigma_q := \sigma|_{H_q}^\phi$ is a valid strategy in $\Gamma_P (\leftarrow G)$.
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 \( \sigma_q \) has even length.

Even-prefix-closed. Let \( \pi'_q \subseteq \pi_q \in \sigma_q \), 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 \( \pi_q = \pi|_{H_q} \). We now compute:

\[
\pi'_q = \pi_q|_\ell = \left( \pi|_{H_q} \right)|_\ell = \left( \pi|_{H_q} \right)|^\phi_{H_q} \subseteq \sigma,
\]
which shows that \( \pi'_q \in \sigma_q \) as was desired.

Deterministic. Suppose that we have the following plays in \( \sigma_q \):

\[
\pi = \langle \delta_0, \beta_0, \delta_1, \beta_1, \ldots, \delta_k, \beta_k \rangle,
\]

\[
\bar{\pi} = \langle \bar{\delta}_0, \bar{\beta}_0, \bar{\delta}_1, \bar{\beta}_1, \ldots, \bar{\delta}_k, \bar{\beta}_k \rangle,
\]
so that there are plays

\[
\pi = \langle \delta_0, \beta_0, \delta_1, \beta_1, \ldots, \delta_k, \beta_k \rangle,
\]

\[
\bar{\pi} = \langle \bar{\delta}_0, \bar{\beta}_0, \bar{\delta}_1, \bar{\beta}_1, \ldots, \bar{\delta}_k, \bar{\beta}_k \rangle,
\]
in \( \sigma \), such that

\[
\pi_q = \pi|_{H_q} \quad \text{and} \quad \bar{\pi}_q = \bar{\pi}|_{H_q}.
\]

Observe that since play-splitting leaves all doubter moves unaltered, we have that \( \delta_i = \bar{\delta}_i (= \delta_i^\phi) \) 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 \( \beta_j = \bar{\beta}_j \) holds for \( 0 \leq j < i \leq k \). Now look at the plays \( \pi|_{2i+2} \) and \( \bar{\pi}|_{2i+2} \) which both belong in \( \sigma \) since it is prefix-closed. Then \( \beta_i = \bar{\beta}_i \), since both plays belong in the same (deterministic) strategy \( \sigma \). This essentially yields \( \pi = \bar{\pi} \) and consequently \( \pi_q = \bar{\pi}_q \), which establishes the determinacy of \( \sigma_q \).

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

Proof. We are given a play \( \pi_q \in \sigma_q \) and a doubter move \( \delta \) such that \( \pi_q \vdash \delta \) is valid in \( \Gamma_{P_q}(\leftarrow G) \), and we seek a believer next-move for it. By the hypothesis, there exists some \( \pi \in \sigma \), such that \( \pi_q = \pi|_{H_q} \). Since bodies are left intact by restriction, \( \pi \vdash \delta \) is valid in \( \Gamma_{P}(\leftarrow G) \). Hence, by the totality of \( \sigma \), there is a believer move \( \beta \) such that \( \pi \vdash (\delta, \beta) \in \sigma \). This means that \( (\pi \vdash (\delta, \beta))|_{H_q} \in \sigma_q \), and its last move is the believer move that we sought.

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

Proof. By definition, every play in \( \sigma|_{H_q} \) is created by restricting a play of \( \sigma \). This grants finiteness, as \( \sigma|_{H_q} \) is nothing more than the image of a function \( (\pi \mapsto \pi|_{H_q}) \) whose domain is the finite set \( \sigma \).
Corollary 5.26 (Preservation of winning). If \( \sigma \) is winning, then so is \( \sigma|_{H_{\phi}} \).

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

**Definition 5.14 (Strategy splitting).** Let \( \sigma \) be a strategy in \( \Gamma_{\mathcal{P}(\leftarrow \mathcal{G})} \), and let \((\mathcal{P}_1, \mathcal{P}_2)\) be a splitting of \( \mathcal{P} \) wrt some \( \phi \in \mathcal{P} \) over \( \mathcal{H} := (H_1, H_2) \). The splitting of \( \sigma \) over \((\mathcal{P}_1, \mathcal{P}_2)\) is the pair

\[
(\sigma_1, \sigma_2) := (\sigma|_{H_1}, \sigma|_{H_2}) .
\]

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 are sound and complete wrt 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 wrt SLD resolution.

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

The first of these is proved in [6], while the second one is due to [5] (and can also be found in [11, 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 \( \leftarrow p \) a goal. Then there is a winning strategy in the associated LP game iff \( p \) 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 \( \mathcal{P} \) wrt \( \phi \) over \((H_1, H_2)\). Then

\[
\text{MM}(\mathcal{P}) \subseteq \text{MM}(\mathcal{P}_1) \cup \text{MM}(\mathcal{P}_2) \subseteq \text{HM}(\mathcal{P}).
\]

**Proof.** Let \( \phi_1 := \phi|_{H_1} \) and \( \phi_2 := \phi|_{H_2} \). For the first inclusion, let \( S \in \text{MM}(\mathcal{P}) \), and suppose \( S \notin \text{MM}(\mathcal{P}_1) \). We need \( S \in \text{MM}(\mathcal{P}_2) \). There are two ways in which \( S \) can fail to be in \( \text{MM}(\mathcal{P}_1) \): 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 \( S_0 \subseteq S \) 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 \( \mathcal{P}_1 \).

\[
S \in \text{MM}(\mathcal{P}) \implies S \in \text{HM}(\mathcal{P}) \\
\implies S \models \psi \text{ for all } \psi \in \mathcal{P} \\
\implies S \models \psi \text{ for all } \psi \in \mathcal{P}_1 \setminus \{ \phi_1 \} \quad (\mathcal{P}_1 \setminus \{ \phi_1 \} \subset \mathcal{P}) \\
\implies S \nmid \phi_1 \quad \text{(by case hypothesis)}
\]

Since \((\phi_1, \phi_2)\) is the splitting of \( \phi \) over \((H_1, H_2)\), \( S \) is forced to satisfy \( \phi_2 \), and therefore satisfies every element of \( \mathcal{P}_2 \), so that \( S \in \text{HM}(\mathcal{P}_2) \).

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, \( S \in \text{MM}(\mathcal{P}_2) \). We have proven the inclusion

\[
\text{MM}(\mathcal{P}) \subseteq \text{MM}(\mathcal{P}_1) \cup \text{MM}(\mathcal{P}_2). \tag{1}
\]

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

\[
\text{MM}(\mathcal{P}_1) \cup \text{MM}(\mathcal{P}_2) \subseteq \text{HM}(\mathcal{P}_1) \cup \text{HM}(\mathcal{P}_2). \tag{2}
\]

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

\[
\text{HM}(\mathcal{P}_1) \cup \text{HM}(\mathcal{P}_2) \subseteq \text{HM}(\mathcal{P}). \tag{3}
\]

Putting (1)–(3) together:

\[
\text{MM}(\mathcal{P}) \subseteq \text{MM}(\mathcal{P}_1) \cup \text{MM}(\mathcal{P}_2) \subseteq \text{HM}(\mathcal{P}_1) \cup \text{HM}(\mathcal{P}_2) \subseteq \text{HM}(\mathcal{P}). \tag{4}
\]

**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

\[
\mathcal{P} := \left\{ \begin{array}{c}
\mathbf{b} \leftarrow \\
\mathbf{a} \lor \mathbf{b} \lor \mathbf{c} \leftarrow
\end{array} \right\} \quad \leadsto \quad \mathcal{P}_1 := \left\{ \begin{array}{c}
\mathbf{b} \leftarrow \\
\mathbf{a} \leftarrow
\end{array} \right\}, \quad \mathcal{P}_2 := \left\{ \begin{array}{c}
\mathbf{b} \leftarrow \\
\mathbf{b} \lor \mathbf{c} \leftarrow
\end{array} \right\}.
\]

Then the corresponding sets of Herbrand models and minimal models are

\[
\text{HM}(\mathcal{P}) = \{ \{ \mathbf{b} \} \}, \quad \text{MM}(\mathcal{P}) = \{ \{ \mathbf{b} \} \}, \quad \text{HM}(\mathcal{P}_1) = \{ \{ \mathbf{a}, \mathbf{b} \} \}, \quad \text{MM}(\mathcal{P}_1) = \{ \{ \mathbf{a}, \mathbf{b} \} \}, \quad \text{HM}(\mathcal{P}_2) = \{ \{ \mathbf{b}, \mathbf{c} \} \}, \quad \text{MM}(\mathcal{P}_2) = \{ \{ \mathbf{b} \} \},
\]

so that in this example all inclusions are proper:

\[
\text{MM}(\mathcal{P}) \subsetneq \text{MM}(\mathcal{P}_1) \cup \text{MM}(\mathcal{P}_2) \subsetneq \text{HM}(\mathcal{P}_1) \cup \text{HM}(\mathcal{P}_2) \subsetneq \text{HM}(\mathcal{P}).
\]
Theorem 6C (Soundness of the DLP game semantics). Let $\mathcal{P}$ be a clean DLP program, and $\leftarrow G$ a goal. If there is a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow G)$, then $G$ is true in every minimal model of $\mathcal{P}$.

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

**Base.** Let $\sigma$ be a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow G)$. We claim that there exists a combo-free winning strategy $\sigma'$ in the same game. If so, observe that the head of the first believer move of every play of $\sigma'$ must be one disjunct $g$ of $G$. Hence by altering the first (doubter) moves of its plays to correctly doubt $g$, we end up with a combo-free, winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow g)$. This is now compatible with the LP game and we can use the soundness of the LP game semantics (Corollary 6.1) to obtain $g \in \text{LHM}(\mathcal{P})$. But this means that $G$ is true in $\text{LHM}(\mathcal{P})$, the only minimal model of $\mathcal{P}$.

**Proof of the claim.** If $\sigma$ contains no combo moves we are done. Otherwise, pick a maximal play from $\sigma$ such that it contains at least one combo move and look at the last one, $\beta$. We show that we can safely replace $\beta$ 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 $\beta$ contains a "bad" atom such that when doubted by the doubter, we cannot win. But this contradicts the fact that $\sigma$ is winning, as it contains no answer for a doubter move that doubts exactly one bad atom from each rule of $\beta$.

**Induction step.** Since $\mathcal{P}$ is a proper DLP program, we can pick a proper DLP rule $\phi \in \mathcal{P}$, and split $\mathcal{P}$ over it wrt some proper partition $(H_1, H_2)$ of $\text{head}(\phi)$ to get $(\mathcal{P}_1, \mathcal{P}_2)$. Notice that $N(\mathcal{P}_1) = N(\mathcal{P}_2) = N(\mathcal{P}) - 1$, which allows us to use the induction hypothesis for both programs $\mathcal{P}_1$ and $\mathcal{P}_2$.

Let $\sigma$ be a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow G)$, and split it to derive two strategies $\sigma_1$ and $\sigma_2$ for $\Gamma_{\mathcal{P}_1}(\leftarrow G)$ and $\Gamma_{\mathcal{P}_2}(\leftarrow G)$ respectively (Theorem 5C). Since $\sigma$ is winning, $\sigma_1$ and $\sigma_2$ are also winning (Corollary 5.26), and so by the induction hypothesis we know that $G$ must be true in every minimal model of $\mathcal{P}_1$ in every minimal model of $\mathcal{P}_2$. In other words, $G$ is true in the union $\text{MM}(\mathcal{P}_1) \cup \text{MM}(\mathcal{P}_2)$; so by Lemma 6.2, it is true in every element of $\text{MM}(\mathcal{P})$.

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

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

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

**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, MM(\(P_1\)) \(\cup\) MM(\(P_2\)) \(\subseteq\) HM(\(P\)), so that \(G\) is true in every minimal model of \(P_1\) and in every minimal model of \(P_2\). By the induction hypothesis, there are two winning strategies \(\sigma_1\) and \(\sigma_2\) in the games \(\Gamma_{\forall P_1}(\leftarrow G)\) and \(\Gamma_{\forall P_2}(\leftarrow G)\) respectively. Using Theorem 5B we can combine them to get a new strategy \(\sigma_1 \land \sigma_2\) for \(\Gamma_{\forall P}(\leftarrow G)\), which is winning by Corollary 5.23.

Lemma 6.3. Let \(P\) be a general DLP program, and \(\leftarrow G\) a disjunctive goal. Then there is a winning strategy in \(\Gamma_{\forall P}(\leftarrow G)\) iff there is a winning strategy in \(\Gamma_{\forall P}^b(\leftarrow G)\).

Proof. “\(\Rightarrow\)”: We are given a winning strategy \(\sigma\) for \(\Gamma_{\forall P}(\leftarrow G)\). To win in \(\Gamma_{\forall P}^b(\leftarrow G)\), we move as follows: suppose that the believer following \(\beta\) plays \(\beta = \langle \psi_1, \ldots, \psi_n \rangle\).

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

\[\hat{\beta} := \hat{\psi}_1 + \cdots + \hat{\psi}_n.\]

It remains to verify two things: (i) that it is indeed a valid move, and (ii) that we know how to win from any next doubter move \(\hat{\delta}\). (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 \(\hat{\delta}_1\) on \(\hat{\psi}_i\), there is a disjunction \(D^i \in \text{body}(\psi_i)\) such that \(D^i \subseteq \left[\hat{\delta}_1\right]\). Then, our move in \(\Gamma_{\forall P}^b(\leftarrow G)\) against a doubter move consisting of such doubts

\[\hat{\delta} := \hat{\delta}_1 + \cdots + \hat{\delta}_n\]

is \(\sigma\)'s move for the doubts \(\delta := \langle D^1, \ldots, D^n \rangle\), valid thanks to the claim and the trivial observation that

\[\left[\delta\right] = \left[\hat{\delta}_1 + \cdots + \hat{\delta}_n\right] = \left[\hat{\delta}_1\right] \cup \cdots \cup \left[\hat{\delta}_n\right].\]

Proof of the claim. Assume otherwise. Then there is a rule \(\psi_i \in \beta\) of the form

\[\psi_i := E_i \leftarrow D^i_1, \ldots, D^i_k,\]

and a \(\hat{\delta}_i\) on \(\hat{\psi}_i\) such that for all \(D_j^i \in \text{body}(\psi_i)\), \(D_j^i \not\subseteq \left[\hat{\delta}_i\right]\), i.e., there is a \(d_j^i \in D_j^i\) with \(d_j^i \not\in \left[\hat{\delta}_i\right]\). But this implies that \(\hat{\delta}\) did not doubt anything from the body of the rule

\[E_i \leftarrow d_1^i, \ldots, d_k^i, \in \hat{\psi}_i,\]

41
which is impossible.

"≤": In this direction we are given a strategy \( \sigma \) in \( \Gamma_\mathcal{P}(\leftarrow \mathcal{G}) \). Again, to win in \( \Gamma_\mathcal{P}(\leftarrow \mathcal{G}) \), we follow \( \sigma \) 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 \( \beta := \langle \psi_1, \ldots, \psi_n \rangle \) is the move that \( \sigma \) would play, where some of the \( \psi_i \)'s are transformed. We then play

\[
\beta^\gamma := \langle \psi_1^\gamma, \ldots, \psi_n^\gamma \rangle,
\]

where \( \psi_i^\gamma := \) the first \( \rho \in \mathcal{P} \) such that \( \psi_i \in \beta \).

We must verify the same claims (i)--(ii) as above. (i) is trivial for the same reason. For (ii), let \( \delta^\gamma := \langle D_1, \ldots, D_n \rangle \) be such a doubter response to \( \beta^\gamma \) so that \( D_i \in \text{body}(\psi_i^\gamma) \). This translates easily to a doubt \( \delta \) on \( \beta \), namely

\[
\delta := \langle \{d_1\}, \ldots, \{d_n\} \rangle,
\]

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

\[
[\delta] = \{d_1, \ldots, d_n\} \subseteq D_1 \cup \cdots \cup D_n = [\delta^\gamma].
\]

This allows us to copy what move \( \sigma \) would play in \( \Gamma_\mathcal{P}(\leftarrow \mathcal{G}) \) against \( \delta \) and go on in this manner until the winning strategy \( \sigma \) plays a fact \( \psi \) among its rules; and then \( \psi^\gamma \) will also be a fact.

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

**Proof.** We have proved everything we need:

\[
\begin{align*}
\text{there is a winning} & \quad \iff \quad \text{there is a winning} \\
\text{strategy in } \Gamma_\mathcal{P}(\leftarrow \mathcal{G}) & \quad \iff \quad \text{strategy in } \Gamma_\mathcal{P}(\leftarrow \mathcal{G}) \\
\text{G is true on every} & \quad \iff \quad \text{G is true on every} \\
\text{minimal model of } \mathcal{P} & \quad \iff \quad \text{minimal model of } \mathcal{P}.
\end{align*}
\]

7. Conclusion

In this article we defined a game semantics for disjunctive logic programs and proven 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 uniting 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 [4]. 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 Hyland–Ong games (see [10]). Once all four versions of logic programming have been dealt with in a uniform way via games, a fruitful connection with HO games might be possible. In this way, game semantics could build a long-desired bridge between logic programming and functional programming, and also provide a connection with proof theory through innocence.

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.\footnote{As a sadistic (and embarrassing) exercise, try proving Theorems 6C–6D by induction on the number of minimal models of $\mathcal{P}$ and not on the number of occurrences of $\lor$. Good luck.}

I would also like to thank Panos Rondogiannis who introduced me to logic programming, for his encouragement to further investigate the games I described in my M.Sc. thesis [17], and for his ongoing helpful comments.

The Marie Curie Fellowship MALOA is funding my research, for which I am obviously grateful.

Finally, as this is a continuation of ideas that were born in my M.Sc. thesis, every person (or thing) mentioned in its acknowledgments deserves some credit here too. The list is too long.

References


C. Baral, G. Brewka, J. Schlipf (Eds.), Logic Programming and Non-
monotonic Reasoning, volume 4483 of Lecture Notes in Computer Science,


logic programming, in: C. Palamidessi, H. Glaser, K. Meinke (Eds.),
Principles of Declarative Programming, volume 1490 of Lecture Notes
10.1007/BFb0056626.


[8] M. Fitting, Fixpoint semantics for logic programming—a survey, Theoretical

for well-founded negation in logic programming, Annals of Pure and Ap-
plied Logic 151 (2008) 70–88. First Games for Logic and Programming
Languages Workshop.

ables and the full abstraction problem; II. Dialogue games and innocent
strategies; III. A fully abstract and universal game model, Information and


[12] J.W. Lloyd, R.W. Topor, Making prolog more expressive, J. Logic Pro-


[16] P. Rondogiannis, W.W. Wadge, Minimum model semantics for logic pro-
467.

2010.

Index

\[ A, 6, 21 \]
\[ N(\mathcal{P}), 40 \]
\[ T_\mathcal{P}, 3 \]
\[ \Gamma_\mathcal{P}(\leftarrow \mathcal{G}), 4, 15 \]
\[ \sqsubseteq, 5 \]
\[ \gamma \]
\[ D \gamma E, 8 \]
\[ \mathcal{G} \gamma \mathcal{G}, 8 \]
\[ \phi_1 \gamma \phi_2, 8 \]
\[ \sigma_1 \gamma \sigma_2, 31 \]
\[ \tau_1 \gamma \tau_2, 29 \]
\[ \tau_1 \tilde{\tau}_2, 26 \]
\[ \vdash \mathcal{P} \phi, 9 \]
\[ \phi^\sigma_{\mathcal{A}}, 9 \]
\[ \sigma^\phi_{\mathcal{H}_n}, 36 \]
\[ \vdash, 5 \]
\[ \cong, 5 \]
\[ \sqsubseteq, 5 \]
\[ \subseteq, \sqsubseteq, 5 \]
\[ \subseteq, \sqsubseteq, 5 \]
\[ \triangleright, 5 \]
\[ \triangleright, 5 \]
\[ \cong, 5, 21, 24 \]
\[ \tau, 32 \]
\[ \text{body}(\phi), 6 \]
\[ \text{cr}(\pi), 33, 34 \]
\[ \text{cr}_{\sigma_c}, 33 \]
\[ \text{head}(\phi), 6 \]
\[ \text{proj}_{\sigma_c}(\pi), 32 \]
\[ \text{rmstall}(\tau), 18 \]
\[ \text{sync}(\tau_1, \tau_2), 27 \]
\[ \beta, 18 \]
\[ \alpha, 18 \]
\[ \pi^-, 21, 32, 34 \]
\[ \tau^-, 5 \]
\[ \varepsilon, 17, 20 \]
\[ \mathcal{P}_+, 16 \]

agree

\[ \tau \text{ with } \sigma, 32 \]

AJM games, 5

believer move, 16

benefit of the doubt, 16, 22

body, 2, 6

chain, 20

clause, 6

body, 6

fact, 6

goal, 6

head, 6

clean

program, 6

rule, 6

version, see also \[ \hat{\mathcal{P}}, 7 \]

closed world assumption, 22

closure operator, 21

CNF, 6

code, see Haskell code

combination, 4

of conjunctions, 8

of dialogues, 29

of disjunctions, 8

of rules, 8

of strategies, 31

of synchronous plays, 25, 26

combo move, 16, 22, 40

commutative diagram, 32, 34

completeness, 4, 38

conjunction, 6
combination, 8
creators, 31
shortest, 31
CWA, see closed world assumption
declarative semantics, see denotational semantics
decomposition, 31
denotational semantics, 10
description logic, 22
desugaring, 7
diagram
  commutative, 32, 34
dialogue, 16
  combination, 29
  synchronization, 27
disjunction, 2, 6
  combination, 8
disjunctive clause, 7
disjunctive combination, see combination
DLP, 3, 6
  compatibility, 22
  conjunction, see conjunction
denotational semantics, 12
disjunction, see disjunction
game
  simplified, 13
  game semantics, 21
  completeness, 40
  soundness, 40
  program, see program rule, see rule
DLPN, 3
doubt, 16
doubter move, 16
draw, 22
extensive operator, 21
fact, 6
fixpoint semantics, 3
follow move, 18
forbidden move, 23
functional programming, 3, 5, 43
game
LP, 13
LPN, 13
play, 17
simplified DLP, 13
strategy, 19
game semantics, 3, 21, 21
general
  program, 6
goal, 6
  success, 11, see also semantics
Haskell code
  rmstall, 19
  sync, 28
head, 2, 6
Herbrand
  base, see also HB(\mathcal{P}), 11
  interpretation, 11
  least model, see also LHM(\mathcal{P}), 11
  model, 11
  least, 2, 4
HO games, see Hyland–Ong games
Hyland–Ong games, 43
idempotent
  operator, 21
implicit rule, 14
innocence, 43
justification, 16
knowledge representation, 22
least Herbrand model, see Herbrand model, 11, 38
least upper bound, 20
limit, 20
Lloyd–Topor transformation, 7
logic programming, 2
LP, 3, 38
denotational semantics, see also least Herbrand model, 11
game, 13, 38
game semantics, 22
  completeness, 38
  soundness, 38
LPN, 3
game, 13, 22
lub, see least upper bound

minimal model semantics, 4, 12
Minker semantics, see minimal model semantics
mip, see model intersection property
model
Herbrand, see Herbrand model
minimal, 4
well-founded, 2, 4, 22
model intersection property, 11
model-state, 12
monotone
answer$_r(\pi)$, 21, 33
rmstall$(\tau)$, 19, 33
sync$(\tau_1, \tau_2)$, 29
$\tau_1 \gtrless \tau_2$, 30
$\tau_1 \gtrless \tau_2$, 26
operator, 21
move
believer, 16
combo, see combo move
doubter, 16
doubts of $\delta$, 16
follow, 18
forbidden, 23
justification of $\beta$, 16
release, 23
stalling, 18
statement, 16
negation, 2
negation-as-failure, 2
notation, 5, 18, 23

open world assumption, 22
operational semantics, see procedural semantics
operator
closure, 21
extensive, 21
idempotent, 21
monotone, 21
revision, 3
OWA, see open world assumption

PCF, 5
play, see also preservation, 17
combination, 29
synchronous, 26
creators, 31
in sync with, see synchronous
restriction, 30
splitting, 31
synchronization, 27
synchronous, see synchronous
player rôles, 15
poset, 20
preservation
of finiteness
$\sigma^\phi_{H_q}$, 37
$\sigma_1 \gtrless \sigma_2$, 36
of parity
sync$(\tau_1, \tau_2)$, 28
$\tau_1 \gtrless \tau_2$, 29, 32, 35
$\tau_1 \gtrless \tau_2$, 26
of plays
$\tau_1 \gtrless \tau_2$, 29
$\tau_1 \gtrless \tau_2$, 26
of totality
$\sigma^\phi_{H_q}$, 37
$\sigma_1 \gtrless \sigma_2$, 36
of winning
$\sigma^\phi_{H_q}$, 38, 40
$\sigma_1 \gtrless \sigma_2$, 36, 41
procedural semantics, 3
program
clean, 6
dialogue, 16
genral, 6
proper, 6
propositional, 6
restriction, 9
splitting, 10
projection, 32
proof theory, 43
proper
program, 6
rule, 6
propositional programs, 6
quasidialogue, 18
quasiplay, 18, 32
rôle, 15
refutation, 3
release, see also $\beta^*_i$, 23
resolution
  SLD, 38
restriction
  of a play, 30
  of a program, 9
  of a rule, 9
  of a strategy, 36
revision operator, 3
rule, 6
  body, 2, 6
  clean, 6
  combination, 8
  head, 2, 6
  implicit, 14
  proper, 6
  restriction, 9

semantic web, 22
semantics
  declarative, see denotational
  denotational, 2, 10
  fixpoint, 3
  game, 3, 21
  least Herbrand model, 11
  least model-state, 12
  minimal model, see minimal model semantics, 12
  operational, see procedural
  procedural, 3
  well-founded, see well-founded
SLD resolution, 38, see also procedural semantics
soundness, 4, 38
splitting, 4
  of a play, 31
  of a program, 10
  of a strategy, 38
stalling, 18, 18
statement, 16
strategy, see also preservation, 19
  combination, 31

combo-free, 20, 40
restriction, 36
splitting, 38
total, 20
winning, 20
success of a goal, see goal success
synchronization, see also sync, 27
synchronous
  combination, see combination of synchronous plays
  plays, 24
syntactic sugar, 7
tie, 22
total strategy, see also preservation, 20
truth values, see also $T$, $F$, 5
well-founded model, see model winning
  strategy, 20