THÈSE

en vue de l'obtention du grade de

Docteur de l'Université de Lyon,
délivré par l'École Normale Supérieure de Lyon

discipline : Informatique

Laboratoire de l'Informatique du Parallélisme (LIP)

École Doctorale Infomaths

présentée et soutenue publiquement le 2 juillet 2014 par

Athanasios Tsouanas

On the Semantics of Disjunctive Logic Programs

Directeur de thèse :
Olivier Laurent

Rapporteurs :
Guy McCusker
Dale Miller

Devant la commission d'examen formée de :
Christophe Fouqueré (président)
Ekaterina Komendantskaya
Olivier Laurent
Guy McCusker

On the Semantics of Disjunctive Logic Programs

by Thanos Tsouanas

a PhD dissertation supervised by Olivier Laurent

To miláκι,
for tudo.

Acknowledgements

This is the acknowledgements part, where I thank and give credit to those who I feel played a crucial and helpful rôle in all the work and research that lead up to this thesis.[1] Thus it is very easy to choose where to begin: my supervisor, Olivier Laurent, for all his support (scientific and non-scientific alike), his constructive guidance, and his patience, and for putting up with various peculiarities of mine, including my night-owl sleeping patterns. It has been a really enlightening experience working under his supervision and attending his seminars. My work would have been tremendously harder without his input. I had the pleasure to be part of the wonderful Plume team of the LIP lab here in ENS de Lyon, and I would especially like to thank Alexandre Miquel for many annoyingly intriguing discussions, and Colin Riba and Patrick Baillot for all their help. Filippo Bonchi, Damien Pous and Fabio Zanasi have all assisted me in times that I needed tools from their fields of expertise. Thanks also goes to Daniel Hirschkoff for being ἀστεῖος, and for providing a lot of entertainment with his band, among other things. Credit is definitely due to our secretariat staff members for all their hard work and kindness; I would especially like to thank Catherine Desplanches, Damien Séon & Sèverine Morin, for all they have done. Outstanding technical support has been provided by Serge Torres; thanks!

The MALOA project has funded the vast majority of my PhD studies.[2] I'm grateful to them, and glad that I met many interesting and friendly people in their events, especially Dugald Macpherson and Zoé Chatzidakis. As dictated by my scholarship, I had to spend some time working in different universities: I am thankful to Luke Ong in the University of Oxford, and Dale Miller in École Polytechnique, for giving me the chance to work in their teams, which assisted me a lot with my research. One more reason to thank Dale, is for many enlightening conversations, constructive feedback, and for taking the time to thoroughly read this manuscript. The same goes for Guy McCusker whom I had the pleasure to meet in plenty occasions. Christophe Fouqueré and Katya Komendantskaya were also very kind to accept to be examiners in the jury of my defense, besides my very short notice. John Power's continuous

[1] Some similarities with the acknowledgements of my MSc thesis are merely a good indication of the fact that some values withstand time. Of that, and also of my laziness to come up with new text.

[2] The European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement #238381 (the Marie Curie Initial Training Network in Mathematical Logic).

encouragement and expertise aided me tremendously multiple times that I felt desperately stuck in my work. I owe a lot to him. Also, Paulo and Isabel Oliva offered me priceless support; thank you both!

I'm indebted to the most influential professors I've had in my undergraduate and post-graduate studies, especially to Yiannis Moschovakis whose set theory and recursion theory courses made me realize my passion for mathematical logic. I would also like to thank Aristides Katavolos for his brilliant mathematical analysis courses, as well as Nikolaos Papaspyrou, Costas Koutras, and Dimitrios Thilikos, who played a crucial part in my progress. Martín Escardó's fascinating course introduced me to denotational semantics, for which I am grateful. Panos Rondogiannis, my MSc thesis supervisor, is in fact the one to blame for my interest in the semantics of logic programming and game semantics in particular. He has always been available for discussing any problems or doubts, each of the many times that I needed him during my PhD studies.

Assuming you don't speak Ελληνικά, you should also thank my English teacher, Mike Pozantzis, for being excellent at what he does; I owe him a lot. I feel lucky to have attended the French courses of Yasmine Daâs, even though I did a terrible job as a student—désolé!

Of course I owe a great lot to my family: Maroui, my mother, who is one of the smartest and kindest persons I know, for all her endless love and priceless lessons; my beloved sister, Eva, for being supportive and next to me since a few months before I was even born; and my father, Nikos, an old school hacker to whom I definitely owe—assuming this is a good thing—my geekiness with computers, since he introduced me to them when I could barely speak. I would also like to mention my brother-in-law, Spyridon Samothrakis, for many interesting conversations, scientific or not. My girlfriend, Ramile Carvalho da Silva, has gone through everything that I have throughout all this time that it took to complete my thesis, and even more. It has been a blessing to know that her love and comfort awaits for me after many stressful and disappointing moments, and that I have her support, taking care of me and of all those things that I seem to be incapable of dealing with on my own. Tons of hard parts were only made manageable thanks to her smile or effort. Eu te amo! ♡

It is great to have friends one can rely upon, and with whom one can have a great time as well. I want to thank my brother Yiannis Tzi Tselekounis, Harris Chanialidis, Matoula Petrolia, Georgios Zoulou Kapetanakis, Marilou Karanika, Irini Petroula, Christos Tarantilis, Iraklis Choraitis, and Stathis Darkorang Lytras, for their love and support. Michail Pshlos Dimakos, Panos Tsaknias, and Alexandros Enecon Batsis have in addition provided mathematical insight and help more times than I could possibly remember; and if it hadn't been for Pshlos, I wouldn't even have a BSc degree. Mary Chlouveraki deserves to be singled out, as I feel I owe it to her, my sudden realization of interest in pure mathematics. A brilliant person and dearest friend, probably the only one whose advice I do not automatically ignore—and for good reason. I also thank Barbara Petit and João Aidar for standing by my side in numerous crazy occasions, and together with Rafael Padilha, Fernando Thiers, Nikos Konstantinides, Yannis Arapis, Emília Santos, Rodrigo Pereira, and Joel Creppy, for many relaxing basketball games and fun times in general, especially in periods of great pressure; and café 203 for providing sufficient amounts of much-needed, Bourganel beer. I am grateful to Tainá Lima, for keeping me company while working on the last details of my thesis and defense, and for not letting me fall too hard when I know I would have. Myla Medina, you talked some sense into me when I was in need of; obrigado! Settling in a bureaucratic place without speaking the language is a challenging and tedious task. Christina Apostolopoulou *&* Florent Murat, Lionel Rieg, Jean-Marie Madiot, and Barbara gave me invaluable help, time and time again. I'm obliged to Irina Kalliamvakou for being a solid rock by my side for more than half of my life so far, and for so many things that still cannot be expressed in any language I know. Giselle Reis has been extremely kind and always offering help, even at times that I wouldn't even think of asking. I owe you ω!

I thank all the friendly and welcoming people in UFMG, especially Loïc Cerf, Fernando Pereira, and Mário Alvim, and while I'm at it, the ICEx cafeteria for all the pães de queijo and açaí. I am also grateful to Ronaldo Silva *&* Luzia Carvalho for their generous hospitality, and for offering me a welcoming place where I could concentrate and work on my thesis while I was in Espírito Santo, Brazil. The same goes for Leo Valadão and the schools Uptime Linhares and Yázigi São Mateus. The children's computers school on Filellinon Street, Aghios Nikolaos, Crete, let me use their scanners and printers on a very crucial minutes-before-the-deadline moment, getting stressed on my behalf. Thanks!

I am also indebted to Spyros Lyberis who introduced me to the world of Unix back in the summer of '99, instantly changing my relationship with computers. I have become dependant on all those computer tools that make my life so much easier, so I feel I should definitely give credit to the Vim text editor, Don Knuth for TEX, Theo de Raadt and all the OpenBSD developers, Simon Peyton Jones and the entire Haskell community, xmonad and darcs, and of course, freenode and Wikipedia.

One cannot be productive in research, if one is dead. I'm therefore also thankful to the splendid person who shot me in the back while I was in Brazil, for failing to kill me. I hope they have given up this hobby by now, although I doubt it. Ramile managed to convince three police officers to get me to a hospital—thank you all!—when they were thinking of leaving me behind, with a bullet in my back. I'm grateful to the doctors and nurses of Roberto Silvares hospital in São Mateus, and to the ones in Hôpital Pierre Wertheimer in Bron, especially Dr. Eurico Freitas Olim and Dr. Cédric Barrey. I owe all these people my (bullet-free) life.

Finally, I hail Crete—I was on decks of Cretan ferries when I began developing my own ideas on semantics for disjunctive logic programs. And if I have forgotten to include your name while I should have, feel free to write it in the margin below. There's plenty of space.[3]

Thank you!

[3] Similarly, if you did not like being mentioned, take a pen and cross your name out. Sorry.

Contents

List of Tables

List of Figures

Part I

Basic material

Chapter 1

Introduction

1.1 What is a logic program?

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

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

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:

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

where the commas on the right stand for conjunctions. The standard denotational or declarative semantics for this kind of programs is provided by a specific two-valued model, the so-called *least Herbrand model.* We will briefly review this in Section **4.2**, in an attempt to be self-contained; consult [vEK76] or [Llo87] for further information. One extension is to allow *negations* to appear in bodies of rules:

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

By negation, we mean *negation-as-failure* ([Cla78]); the semantics we have in mind here is supplied by the many-valued *well-founded model*, defined in [VGRS91]. But the extension in which we are mostly interested in this text is the appearance of *disjunctions* in heads:

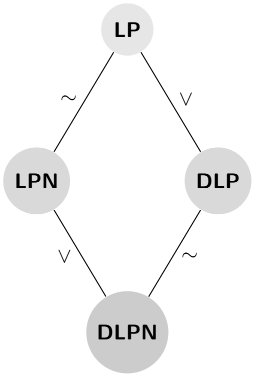
$$\mathtt{a}_1 \vee \cdots \vee \mathtt{a}_n \leftarrow \mathtt{b}_1\,, \cdots, \mathtt{b}_m. \tag{DLP}$$

This enables us to express uncertainty and to derive ambiguous information. Instead of a single least model, in this case, we use a *set of minimal models* for the semantics, as defined in [Min82]; we introduce this in Section **4.3**. Disjunctive logic programs are extensively studied in [LMR92]. Finally, one can consider both extensions simultaneously, by allowing *both* negations in bodies *and* disjunctions in heads:[1]

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

A satisfactory, infinite-valued, model-theoretic semantics for this extension was recently defined in [CPRW07].

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



LP: the so-called *definite* logic programs (with neither negation nor disjunctions);

DLP: disjunctive logic programs;

LPN: logic programs with negation;

DLPN: disjunctive logic programs with negation.

Figure 1.1: The four logic programming languages studied in this thesis.

1.2 Approaches to semantics

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

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

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

Before turning to the next approach, game semantics, one should have a clear understanding of the nature of the aforementioned methods. On one side,

[1] What about disjunctions in bodies, or conjunctions in heads? It is an easy exercise to show that this does not affect the programming language in any meaningful way; it only makes the programmer happier. See also Section **3.4**.

we have the denotational, model-theoretic semantics and their fixpoint characterizations. These provide us with a notion of *correctness* for every possible answer to a goal that we might give to our program. On the operational side, the procedural semantics provide a construction of an answer to our question (the so-called *computed answer*), and this answer has to be correct. Conversely, such a procedure is expected to be able to derive all of the answers that the denotational semantics considers correct. We then say that the procedural semantics is sound and complete with respect to the denotational one. In Chapter **5** we formally define an abstract framework for semantics of logic programming languages, thus eliminating any kind of ambiguity from the term "semantics".

Game semantics. Here we adopt a more anthropomorphic point of view, and treat each program as a set of rules for a game, in which two players compete against each other with respect to the truth of a given goal. One player, the so-called "Doubter" doubts the goal's truthness, while the other player, the "Believer", believing that the goal is true, tries to defend his stance. To get a meaningful semantics out of such games, we look at the *winning strategies* of the players, and depending on their existence, we assign an actual truth value to the goal. A game semantics may have a denotational or an operational flavor, or lie somewhere in-between the two. In [DCLN98], for example, they stay close to the procedural side of semantics, while the game semantics that we investigate here are more of a denotational nature. Part **III** is entirely devoted to this kind of semantics. This game-theoretic approach to semantics is influenced by Lorenzen's dialogue games for logic (see [Lor61]). For an encyclopædic treatment of the use of games in logic, consult [Hod09].

Coalgebraic semantics. This is a very recent approach to semantics for logic programming, which involves using *coalgebraic* methods. Logic programs and their derivation strategies can be modelled as coalgebras on a certain category of presheaves. We will not use coalgebraic semantics in this treatment; for the interested reader, some works on this subject are [BM09], [KMP10], [KP11], [KPS13], and [BZ13]; other examples of approaches that also use categorical methods, include [CM92], [KP96], and [ALM09].

1.3 Outline

Part **I** introduces all the basic material that we will need for our development. In Chapter **2** we fix a propositional language $\mathcal{L}_0$ on which we will build our logic programs, we define the notion of a truth value space, and introduce the family $\mathbb{V}_\kappa$ of such spaces. In the following two chapters we study the syntax and the model-theoretic semantics of the four logic programming languages.

In Part **II** we build our toolkit to deal with disjunctive programs. First, we introduce an abstract framework for semantics in Chapter **5** and show how the semantics that we have seen so far can be placed under this framework. In the next chapter we present restrictions, splittings, combinations, definite instantiations, and D-sections, which are the main tools we will use when dealing with disjunctive programs. In Chapter **7** we define an operator that acts on any semantics of a non-disjunctive logic programming language, and transforms it into an “analogous” semantics for the corresponding disjunctive language. We also demonstrate its use by showing some of its applications.

Part **III** is devoted to game semantics. With the exception of DLPN, each language is directly given a game semantics (LP in Chapter **8**, DLP in **10**, and LPN in **9**). Finally, in Chapter **11**, we show some more applications of the semantic transformation defined in Chapter **7**, this time on game semantics.

1.4 Contributions

The contributions of this dissertation can be summarized as follows:

- An *abstract framework for logic programming semantics* is defined and all semantic approaches that we study are placed within this framework. We introduce the general notion of a *truth value space*, on which we evaluate formulæ. As expected, the booleans form the canonical example of a truth value space, but we need to consider much more general ones when dealing with negation-as-failure.
- The first game semantics for LP was given in [vE86] and [DCLN98]; this was later extended in [GRW08] for the case of LPN programs. Here a *game semantics for DLP programs* is developed in full detail; we prove that it is sound and complete with respect to the standard, minimal model semantics of [Min82]. Even though the game itself can be seen as an extension of the LP game, the formalization and notation we have used is influenced by the games used in functional programming instead.[2]
- We define a *semantic operator* which transforms any given abstract semantics of a non-disjunctive language to a semantics of the “corresponding” disjunctive one. We exhibit the correctness of this transformation by proving that it preserves equivalences of semantics, and we present some applications of it, obtaining some new semantics for DLP and DLPN. In particular, two novel semantics for infinite, propositional DLPN programs are constructed: one model-theoretic and one game-theoretic.

1.5 Related works

Some recent treatments benefit by incorporating tools from areas such as proof theory and linear logic into logic programming:[3] the proof search is often presented in terms of *sequent calculi*, formulæ are not necessarily restricted to Horn clauses of first-order logic, linear connectives are taken into account, etc. [MN12], [MNPS91], [AP91], and [And92], are some works along this line of research, just to mention a few of them. What is more, games have been used successfully in such settings as well: in [PR05] and [MS06], for example, two different approaches for game semantics are presented in the context of *computation-as-proof-search*.

A different school of negation in logic programming, namely *stable model semantics* (see [GL88]) gave rise to a relatively new kind of declarative programming: *answer set programming*, or ASP (see [Gel08]). It allows for disjunctions

[2] This work was published in [Tso13].

[3] This one, does not.

(besides negations), among other things. Unsurprisingly, a game semantics approach is being investigated as well: in [NV07], a game is briefly outlined for programs with single-headed clauses, i.e., for the non-disjunctive fragment of ASP.

1.6 Notation

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

Given a set X, its powerset is $\mathcal{P}(-)$, and we subscript it with n to refer to the set of all subsets of X of cardinality n: e.g., $\mathcal{P}_1(X)$ is the set of all singleton-subsets of X. For the set of all finite subsets of X we use $\mathcal{P}_\mathsf{f}(X)$, so that $\mathcal{P}_\mathsf{f}(X) = \bigcup_{i \in \omega} \mathcal{P}_i(X)$.

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

$$x :: s \triangleq \langle x \rangle +\!\!+ s.$$

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

$$(s_1, s_2) \sqsubseteq (s'_1, s'_2) \overset{\triangle}{\iff} s_1 \sqsubseteq s'_1 \text{ and } s_2 \sqsubseteq s'_2.$$

As has been just demonstrated, $\triangleq$ and $\overset{\triangle}{\iff}$ are used to introduce the definition of a function or symbol, while := is used to "let-bind" the variables appearing on its left to the corresponding expressions that appear on its right (or the other way around). Two abbreviations extensively found in texts of mathematics and logic that we will use are "iff" for "if and only if", and "wff" for "well-formed formula".

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

Chapter 2

Logic (and) languages

2.1 The language $\mathcal{L}_0$ of propositional logic

In this section we describe the propositional logic language which we denote by $\mathcal{L}_0$. Foremost we assume a countably infinite set $\mathcal{At}_0$ whose elements we denote by $a, b, c, \ldots$ and we call *atoms*. We will use the binary connectives $\vee$, $\wedge$, and $\rightarrow$, and the unary connective $\sim$, which is meant to stand for *negation-as-failure*.

A *(well-formed) formula* (wff) is defined inductively: every atom is a formula (called *atomic formula*); and if α and β are formulæ, then so are $\alpha \vee \beta$, $\alpha \wedge \beta$, $\alpha \rightarrow \beta$, and $\sim\alpha$. We might use $\rightarrow$ facing the other way, so that $\alpha \leftarrow \beta$ denotes exactly the same formula as $\beta \rightarrow \alpha$. Furthermore, we use $\alpha \leftrightarrow \beta$ as an abbreviation for the wff $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$, and consider it to have the lowest precedence among these connectives. The *language* $\mathcal{L}_0$ is the set of all wffs.

The *literals* of $\mathcal{L}_0$ are its atomic formulæ (*positive literals*) and their $\sim$-negations (*negative literals*), and we use $\mathcal{Lit}_0$ to denote their set.

2.2 The language $\mathcal{L}_1$ of first-order, predicate logic

Even though we will use almost exclusively the propositional language $\mathcal{L}_0$, we mention here the first-order language $\mathcal{L}_1$, since we will need to refer to it a couple of times. We assume given the countable sets of: variables $\mathcal{Var}$, constant symbols $\mathcal{Con}$, and function symbols $\mathcal{Fun}_n$ for any arity $n > 0$, which we use to inductively define the set of terms $\mathcal{Term}$: a variable is a term, a constant symbol is a term, and if $t_1, \ldots, t_n$ are terms, and $f \in \mathcal{Fun}_n$, then $f(t_1, \ldots, t_n)$ is also a term. We also assume for every $n \in \mathbb{N}$, a countable set of n-ary predicate symbols $\mathcal{Pred}_n$, which we use, in turn, to define the set of atomic formulæ $\mathcal{At}_1$ of $\mathcal{L}_1$: for every $n \in \mathbb{N}$, if $t_1, \ldots, t_n \in \mathcal{Term}$ and $R \in \mathcal{Pred}_n$, then $R(t_1, \ldots, t_n) \in \mathcal{At}_1$, and $\mathcal{At}_1$ contains no other elements besides these. By induction we finally define the set $\mathcal{L}_1$ of wffs: an atomic formula is a wff; if α and β are wffs, and $x \in \mathcal{Var}$, then $\alpha \vee \beta$, $\alpha \wedge \beta$, $\alpha \rightarrow \beta$, $\sim\alpha$, $(\exists x\ \alpha)$, and $(\forall x\ \alpha)$ are also wffs. Again, the literals are the atomic formulæ and their negations, and we denote their set by $\mathcal{Lit}_1$. We will omit the subscripts and simply use $\mathcal{At}$ and $\mathcal{Lit}$ if whether we mean the propositional case or the first-order one is either immaterial or well-understood from the context.

2.3 Truth value spaces

The standard semantics of $\mathcal{L}_0$ is provided by Boolean logic, by mapping each binary connective above to the corresponding boolean operation on $\mathbb{B} = \{\mathsf{F}, \mathsf{T}\}$.

As it turns out, we will need more truth values to handle negation, and therefore we will not tie ourselves to the booleans. Abstracting away the useful properties of $\mathbb{B}$ that we need, we reach the following general definition:

Definition 2.1. A *truth value space* is a completely distributive Heyting algebra with an additional unary operator $\sim$.[1]

The canonical example of a truth value space is $\mathbb{B}$, in which $\sim$ is defined as the classical negation that flips the two values. This space turns out to be too poor for languages that actually use negation as failure, and so we investigate spaces with more values in the next section.

Definition 2.2 (valuation function)**.** Let $\mathcal{V}$ be a truth value space. A function $val : \mathcal{At} \to \mathcal{V}$ is called a (propositional) *valuation function*. We denote the set of valuations ($\mathcal{At} \to \mathcal{V}$) by $\mathcal{Val}_{\mathcal{V}}$, omitting the subscript if it is obvious from the context. Any valuation function val on $\mathcal{At}$ is extended to a valuation over all wffs of $\mathcal{L}_0$ by consulting the operations of $\mathcal{V}$:

$$\begin{aligned} val(\alpha \wedge \beta) &= val(\alpha) \wedge val(\beta); & val({\sim}\alpha) &= {\sim}val(\alpha); \\ val(\alpha \vee \beta) &= val(\alpha) \vee val(\beta); & val(\alpha \to \beta) &= val(\alpha) \Rightarrow val(\beta). \end{aligned}$$

Definition 2.3 ($\mathcal{V}$-tautology)**.** Let $\mathcal{V}$ be a truth value space. The wff α of $\mathcal{L}_0$ is a $\mathcal{V}$*-tautology*, iff $v(\alpha) = \max \mathcal{V}$ for any valuation function v.

Definition 2.4 ($\Gamma \models_{\mathcal{V}} \alpha$, $\alpha \equiv_{\mathcal{V}} \beta$)**.** Let Γ be a set of wffs of $\mathcal{L}_0$ and let $\mathcal{V}$ be a truth value space. We write $\Gamma \models_{\mathcal{V}} \alpha$ and say that the wff α is a *logical consequence of* Γ *in* $\mathcal{V}$ iff $(\bigwedge \Gamma) \to \alpha$ is a $\mathcal{V}$-tautology. Two wffs α and β of $\mathcal{L}_0$ are called *logically equivalent in* $\mathcal{V}$ iff $\alpha \leftrightarrow \beta$ is a $\mathcal{V}$-tautology. In symbols, $\alpha \equiv_{\mathcal{V}} \beta$. Whenever $\mathcal{V}$ is clear from the context we will omit the subscripts in these notations.

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

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

Definition 2.5 (The truth value spaces $\mathbb{V}_\kappa$)**.** Let $\kappa \geq \omega$ be an ordinal number. The structured set[2]

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

[1] See Appendix **A** for more information about lattices and Heyting algebras.

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

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

$$\mathfrak{F}_\kappa \triangleq \{\mathsf{F}_\alpha \mid \alpha < \kappa\}\,; \qquad \mathfrak{U} \triangleq \{\mathsf{U}\}\,; \qquad \mathfrak{T}_\kappa \triangleq \{\mathsf{T}_\alpha \mid \alpha < \kappa\}\,.$$

We denote their union by $\mathbb{V}_\kappa \triangleq \mathfrak{F} \cup \mathfrak{U} \cup \mathfrak{T}$, and equip it with the total order

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

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

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

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

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

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

Definition 2.6. Let κ be an ordinal. The mappings *order* : $\mathbb{V}_\kappa \to \mathbf{ON}$ and *collapse* : $\mathbb{V}_\kappa \to \mathbb{V}_1$ are then defined by

$$\mathit{order}(v) \triangleq \begin{cases} \alpha & \text{if } v \in \{\mathsf{F}_\alpha, \mathsf{T}_\alpha\} \text{ for some ordinal } \alpha < \kappa \\ \kappa & \text{otherwise, i.e., if } v = \mathsf{U}; \end{cases}$$

$$\mathit{collapse}(v) \triangleq \begin{cases} \mathsf{T}, & \text{if } v \in \mathfrak{T}_\kappa, \\ \mathsf{F}, & \text{if } v \in \mathfrak{F}_\kappa, \\ \mathsf{U}, & \text{otherwise.} \end{cases}$$

The intuition behind these truth values is easy to explain: we identify F_0 and T_0 with the usual boolean values F and T, i.e., absolute truth and absolute falsity. The ordinal in the subscript corresponds to a level of doubt that we have, so that F_1 represents a "false" value but with a little doubt, F_2 one with a little more, etc., and similarly for the "true" values. In the middle lies U, which we use in the case that we only have doubts without any bias towards truth or falsity: it is entirely uncertain.

Property 2.1. *For any $\kappa \geq \omega$, $\mathbb{V}_\kappa$ enjoys the following properties:*

(i) *it is bounded;*

(ii) *it is a chain;*

(iii) *it is a distributive lattice;*

(iv) *it is a complete lattice;*

(v) *it is a Heyting algebra;*

(vi) $\mathbb{V}_\kappa \cong \mathbb{V}_\kappa^\partial$.

Lemma 2.2. *For any $\kappa \geq \omega$, $\mathbb{V}_\kappa$ is algebraic.*

Proof. We have just observed that $\mathbb{V}_\kappa$ is a complete lattice (Property 2.1(iv)). We proceed to compute the set of its compact elements $K(\mathbb{V}_\kappa)$. Let $k \in \mathbb{V}_\kappa$

and $S \subseteq \mathbb{V}_\kappa$ such that $k \leq \bigvee S$. For k to be compact, we need a finite $T \subseteq S$ such that $k \leq \bigvee T$. We have four cases to consider:
Case 1: $k = \mathbf{F}_\alpha$ for some non-limit ordinal α. Then S must contain at least some element $t \geq k$, so that $\{t\}$ can be the T that we need, and k is compact.
Case 2: $k = \mathbf{F}_\lambda$ for some limit ordinal λ. Now, taking $S = \{x \mid x < k\}$ we see that $k = \bigvee S$ and therefore k cannot be a compact element.
Case 3: $k = \mathbf{U}$. Similarly to Case 2, we set $S = \mathfrak{F}$ and verify that k is not compact.
Case 4: $k = \mathbf{T}_\alpha$ for some ordinal α. Whether α is limit or not, in this case S will always contain at least some element $t \in \mathbb{V}_\kappa \setminus \{x \in \mathbb{V}_\kappa \mid x \leq \mathbf{T}_{\alpha+1}\}$, so that $t \geq k$, and setting $T = \{t\}$ we see that k is compact.

We have reached the conclusion:

$$K(\mathbb{V}_\kappa) = (\mathfrak{F} \setminus \{\mathbf{F}_\lambda \mid \mathrm{Limit}(\lambda)\}) \cup \mathfrak{T}.$$

We are now in position to prove that $\mathbb{V}_\kappa$ is algebraic, i.e., for every $a \in \mathbb{V}_\kappa$,

$$a = \bigvee \{k \in K(\mathbb{V}_\kappa) \mid k \leq a\}.$$

Indeed, in the nontrivial case that a is not itself compact, it is easy to see that it is the supremum of all the compact elements $k \leq a$. ∎

Theorem 2A. *For any $\kappa \geq \omega$, $\mathbb{V}_\kappa$ is a truth value space.*

Proof. As $\mathbb{V}_\kappa$ has a unary operation $\sim$, we only need to verify that it is a completely distributive Heyting algebra. This follows immediately by Theorem A.7, since we already have Lemma 2.2 and Properties 2.1(iii) and (vi). ∎

We have defined the general structure of a truth value space, and have seen that $\mathbb{B}$ and $\mathbb{V}_\kappa$ are examples of such spaces. In the next two chapters we define the syntax and model-theoretic semantics of the four logic programming languages that we will investigate.

Chapter 3

The syntax of logic programs

In this chapter we describe four logic programming languages, starting with some informal examples meant to indicate how they are used. Then we give formal mathematical definitions that faciliate their study from our perspective, and introduce the reader to some related terminology.

3.1 An informal look

The simplest language that will concern us here is LP. Heads of rules constitute of a single atom, while bodies are conjunctions of atoms. A simple program, for example, is the following:

$$\mathcal{P}_1 = \left\{ \begin{aligned} \texttt{sleeps} &\leftarrow \texttt{tired} \\ \texttt{works} &\leftarrow \texttt{rested} \\ \texttt{eats} &\leftarrow \texttt{rested}\,\texttt{,}\,\texttt{hungry} \\ \texttt{rested} &\leftarrow \end{aligned} \right\}.$$

We might use this simple program to describe the daily life of a rather dull person who is currently rested (this is what the last rule states). We load the program to our implementation and interact with it by asking it questions, or queries, i.e., we ask if a proposition is true or not. The answers have to come from a truth value space $\mathcal{V}$, and in this case we use $\mathbb{B}$.[1] The implementation, if correct, will answer positively ($\mathbf{T}$) if and only if the query is a logical consequence of the program $\mathcal{P}_1$. In this case we say that the query has succeeded; otherwise, it has failed. For example, `works` succeeds here thanks to the second rule, which reduces it to `rested`, which in turn succeeds since it is a fact (the fourth rule). On the other hand, `tired` fails, because there is no evidence to the contrary. This illustrates that by default we consider everything to be false, unless provable otherwise. This assumption, called the *closed world assumption* (CWA), is a very important aspect of logic programming, and it is one of

[1] As long as there are no negations present in the programs, using any other truth value space $\mathcal{V}$ in place of $\mathbb{B}$ is pointless: only its $\mathbb{B}$-isomorphic subset $\{\min \mathcal{V}, \max \mathcal{V}\} \subseteq \mathcal{V}$ will end up being used. This would not have been the case if we had allowed for "facts" of the form $\texttt{a} \leftarrow v$ for every $v \in \mathcal{V}$, but we are not interested in this direction here. As an example of such an approach, the reader is referred to [vE86].

the things that make it so practical as a tool for computing and dealing with database-like information.

In many applications, it is impractical or even literally impossible to explicitly list negative information. Imagine, for example, that we are implementing a program for an airline. We want to ask whether there is a flight between two airports at a given time. Had we not followed this convention, we would have to list, for every pair of airports (x, y), and for every possible time t, the specific information of whether there is a flight from x to y at time t. Following the CWA we only need to list the flights that actually take place, and assume that if no flight is listed on a particular time, then there must indeed be no flight at this time from x to y.

Extending LP. Even on unrealistically simple LP programs like $\mathcal{P}_1$, we can easily spot some limitations of the language: (1) even though `rested` and `tired` are meant to be related in the obvious manner, they are not; and (2) we have no way to express ambiguous information, such as $\texttt{hungry} \vee \texttt{thirsty}$.

The first shortcoming leads to LPN, which introduces the operation $\sim$ in bodies of rules for negation-as-failure. We can now write the program

$$\mathcal{P}_2 = \left\{ \begin{array}{r} \texttt{sleeps} \leftarrow \texttt{tired} \qquad\qquad \\ \texttt{works} \leftarrow {\sim}\texttt{tired} \qquad\qquad \\ \texttt{eats} \leftarrow {\sim}\texttt{tired}\,\texttt{,}\,\texttt{hungry} \end{array} \right\},$$

which describes the typical day of the same dull person, but in a more concise way. To see how we can still conclude that `works` holds, we observe that according to the second rule, it reduces to the (finite) failure of `tired`. And as we have no information to support that `tired` holds, we indeed conclude that $\sim$`tired` succeeds and therefore so does `works`.

The second shortcoming of LP leads to DLP, which relaxes the restriction that the heads can only contain a single atom, by allowing a disjunction of atoms to appear instead. This gives us the flexibility to reason with indefinite information. Consider, the following example:

$$\mathcal{P}_3 = \left\{ \begin{array}{rl} \texttt{mathematician} \leftarrow & \texttt{topologist} \\ \texttt{mathematician} \leftarrow & \texttt{algebraist} \\ \texttt{algebraist} \vee \texttt{topologist} \leftarrow & \end{array} \right\}.$$

In this program, we do not have concrete evidence of whether we are dealing with an algebraist or a topologist, but the third rule states as a matter of fact that (at least) one of the two must hold. Therefore, even though neither the first nor the second rule by themselves are enough to deduce that we are in fact dealing with a mathematician, in this disjunctive setting of DLP programs, we are able to deduce that this is indeed the case.

Naturally, we may allow both extensions simultaneously, and doing so we obtain DLPN which allows both $\sim$ in bodies and $\vee$ in heads, and it is the most general language that will concern us here.

3.2 Terminology

To make it explicit that we refer to the set-theoretic interpretation when we speak of a disjunction or a conjunction, we will use the prefix "L.P." as shown in the following definition, which formally introduces this and related terminology:

Definition 3.1. An *L.P. disjunction* is a finite subset $D \subseteq \mathcal{L}it$. An *L.P. conjunction* is a finite sequence $\mathscr{D}$ of L.P. disjunctions. For obvious reasons we omit the "L.P." prefix whenever no confusion arises. A *clause* is a pair $(H, \mathscr{D})$, in which the *head* $H = \{a_1, \dots, a_n\}$ is an L.P. disjunction, and the *body* $\mathscr{D} = \langle D_1, \dots, D_m \rangle$ is an L.P. conjunction. If the head of a clause is non-empty we call it a *rule*, while if it is empty and $m = 1$, a *goal*.[2] A *fact* is a bodiless clause. In logic programs, rules will be written as

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

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

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

A *clean program* is a countable set of clean rules; it is *disjunctive* (also *proper*), if at least one of its rules is. Note that we have not specified what the atoms in $\mathcal{A}t$ really are. One may consider them to simply be propositional variables without any further structure, just like in propositional calculus. In this case, we have a *propositional program*. Another possibility is to let them be the atomic formulæ of a first-order language, built by its predicates, function symbols, variables, and constants. We then call it a *first-order program*.

▶ *Example 3.1.* Consider the following sets of rules:

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

The program $\mathcal{P}$ is disjunctive and clean, $\mathcal{Q}$ is also disjunctive but not clean, and $\mathcal{R}$ is neither of the two. ◀

▶ *Example 3.2.* Here is a first-order logic program:

$$\left\{ \begin{array}{rcl} \mathtt{daughter}(X,Y) & \leftarrow & \mathtt{child}(X,Y) \,,\, \mathtt{female}(X) \\ \mathtt{spouse}(X,Y) & \leftarrow & \mathtt{married}(X,Y) \,,\, \mathtt{female}(X) \\ \mathtt{married}(X,Y) & \leftarrow & \mathtt{married}(Y,X) \\ \mathtt{child}(\mathit{eva}, \mathit{maroui}) \vee \mathtt{married}(\mathit{sam}, \mathit{eva}) & \leftarrow & \\ \mathtt{female}(\mathit{eva}) & \leftarrow & \\ \mathtt{male}(\mathit{sam}) & \leftarrow & \end{array} \right\}.$$

It is disjunctive and clean. ◀

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

Definition 3.2 (Ground)**.** If a term, an atom, or a formula, ρ or a set of formulæ $\mathcal{P}$ of a first-order language $\mathcal{L}_1$ contains no variables, it is called *ground*.[3] If it does, then by replacing all of its variables with ground terms we obtain a *ground instance* of it. We also define:

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

► *Example 3.3.* Consider the first-order program $\mathcal{E}$ in the language that contains a constant symbol 0, a unary function symbol S, and the unary predicate symbol `even`:

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

We compute:

$$\mathsf{ground}(\mathcal{E}) := \left\{ \begin{array}{rcl} \mathtt{even}(0) & \leftarrow & \\ \mathtt{even}(\mathsf{S}^2(0)) & \leftarrow & \mathtt{even}(0) \\ \mathtt{even}(\mathsf{S}^3(0)) & \leftarrow & \mathtt{even}(\mathsf{S}(0)) \\ \mathtt{even}(\mathsf{S}^4(0)) & \leftarrow & \mathtt{even}(\mathsf{S}^2(0)) \\ & \vdots & \end{array} \right\}.$$

◄

☡ REMARK 3.1 (logic programs and logic formulæ). We have seen that there is an obvious correspondence between logic programs and rules on the one hand and logic formulæ on the other. This allows us to directly use some well-known jargon of mathematical logic: we speak of models, theories, consistency, logical consequences, etc. We should not be too eager to jump to conclusions, however. One of the common pitfalls is to think that the $\sim$ of logic programming corresponds to the $\neg$ of mathematical logic. In fact, the very reason we have chosen to use a different symbol for negation-as-failure, is to keep in mind that this is not the case.

Set-theoretic translations. Given any language of logic $\mathcal{L}$, we will identify a set of $\mathcal{L}$-literals with the $\mathcal{L}$-wff of their disjunction. Note that under this convention, we will identify some actually distinct formulæ, e.g., $a \vee b$ and $b \vee a$; this poses no threat as such formulæ are already equivalent up to commutativity and/or associativity of $\vee$. Similarly, sequences of sets of literals, when regarded as logic formulæ, are identified with conjunctions of disjunctions of literals. Notice that under this convention they are actually formulæ in *conjunctive normal form* (CNF). It is also convenient to consider rules of the form $\alpha \leftarrow \beta$ as pairs (α, β). Thus, we have a mapping from the world of logic to the language of set theory, which is a very convenient tool for our development.

► *Example 3.4.* The set $\{a, b, c\}$ is understood to stand for the disjunction $a \vee b \vee c$, the sequence $\langle \{a\}, \{b, c\} \rangle$ for the conjunction $a \wedge (b \vee c)$, and the pair $(\{p, q\}, \langle \{a, b\}, \{b, c\} \rangle)$ for the implication $((a \vee b) \wedge (b \vee c)) \rightarrow p \vee q$. ◄

[3] See [LMR92, Ch. 2] for more details on first-order languages.

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

▶ *Example 3.5.* Consider the program

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

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

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

Definition 3.3. A logic programming language L is determined by:

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

We define the set of *rules* of L as $\mathbf{R}_L \triangleq \mathbf{H}_L \times \mathbf{B}_L$. A *program* of L is a set of rules of L. We write $\mathbf{P}_L$ for the set of programs of L. In most logic programming languages, the bodies of rules are required to be conjunctions, in which case we denote by $\mathbf{C}_L$ the set of all possible conjuncts out of which bodies are formed; in symbols,

$$\phi \in \mathbf{B}_L \overset{\triangle}{\iff} \phi = \bigwedge C, \text{ for some finite sequence } C \in \mathbf{C}_L^{\star}.$$

Definition 3.4. On the set of rules $\mathbf{R}_L$ of a language L we define two operators head and body as the projections

$$\begin{aligned} \mathsf{head}((H,\mathscr{B})) &\triangleq H, \\ \mathsf{body}((H,\mathscr{B})) &\triangleq \mathscr{B}. \end{aligned}$$

3.3 Four logic programming languages

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

$$\begin{aligned} \mathbf{H}_{\mathrm{LP}} &\triangleq \mathcal{P}_1(\mathcal{At}) & \mathbf{H}_{\mathrm{DLP}} &\triangleq \mathcal{P}_{\mathsf{f}}(\mathcal{At}) \\ \mathbf{C}_{\mathrm{LP}} &\triangleq \mathcal{P}_1(\mathcal{At}) & \mathbf{C}_{\mathrm{DLP}} &\triangleq \mathcal{P}_{\mathsf{f}}(\mathcal{At}) \\ \mathbf{Q}_{\mathrm{LP}} &\triangleq \mathcal{P}_1(\mathcal{At}) & \mathbf{Q}_{\mathrm{DLP}} &\triangleq \mathcal{P}_{\mathsf{f}}(\mathcal{At}) \end{aligned}$$

$$\begin{aligned} \mathbf{H}_{\mathrm{LPN}} &\triangleq \mathcal{P}_1(\mathcal{At}) & \mathbf{H}_{\mathrm{DLPN}} &\triangleq \mathcal{P}_{\mathsf{f}}(\mathcal{At}) \\ \mathbf{C}_{\mathrm{LPN}} &\triangleq \mathcal{P}_1(\mathcal{Lit}) & \mathbf{C}_{\mathrm{DLPN}} &\triangleq \mathcal{P}_{\mathsf{f}}(\mathcal{Lit}) \\ \mathbf{Q}_{\mathrm{LPN}} &\triangleq \mathcal{P}_1(\mathcal{Lit}) & \mathbf{Q}_{\mathrm{DLPN}} &\triangleq \mathcal{P}_{\mathsf{f}}(\mathcal{Lit}). \end{aligned}$$

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

Table 3.1: Summary of representative rules for each language

$\mathtt{p} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b}$	LP
$\mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b}$	cDLP
$\mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b} \vee \mathtt{c}$	DLP
$\mathtt{p} \leftarrow {\sim}\mathtt{a}\,\mathtt{,}\,\mathtt{b}$	LPN
$\mathtt{p} \vee \mathtt{q} \leftarrow {\sim}\mathtt{a}\,\mathtt{,}\,\mathtt{b}$	cDLPN
$\mathtt{p} \vee \mathtt{q} \leftarrow {\sim}\mathtt{a}\,\mathtt{,}\,\mathtt{b} \vee \mathtt{c}$	DLPN

▶ *Example 3.6.* Here are some sample programs written in these languages:

$$\mathcal{P}_1 = \left\{ \begin{array}{l} \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \\ \mathtt{b} \leftarrow \end{array} \right\} \in \mathbf{P}_{\mathrm{LP}} \qquad \mathcal{P}_3 = \left\{ \begin{array}{r} \mathtt{a} \vee \mathtt{b} \leftarrow \quad \\ \mathtt{p} \leftarrow \mathtt{a} \\ \mathtt{p} \leftarrow \mathtt{b} \end{array} \right\} \in \mathbf{P}_{\mathrm{DLP}}$$

$$\mathcal{P}_2 = \left\{ \begin{array}{l} \mathtt{p} \leftarrow \\ \mathtt{r} \leftarrow {\sim}\mathtt{p} \\ \mathtt{s} \leftarrow {\sim}\mathtt{q} \end{array} \right\} \in \mathbf{P}_{\mathrm{LPN}} \qquad \mathcal{P}_4 = \left\{ \begin{array}{r} \mathtt{p} \vee \mathtt{q} \vee \mathtt{r} \leftarrow \qquad \\ \mathtt{p} \leftarrow {\sim}\mathtt{q} \\ \mathtt{q} \leftarrow {\sim}\mathtt{r} \\ \mathtt{r} \leftarrow {\sim}\mathtt{p} \end{array} \right\} \in \mathbf{P}_{\mathrm{DLPN}}$$ ◀

3.4 Clean programs

We use cDLP to denote the language of clean DLP programs, which is determined by:

$$\begin{aligned} \mathbf{H}_{\mathrm{cDLP}} &\triangleq \mathcal{P}_{\mathrm{f}}(\mathcal{A}t) & \mathbf{H}_{\mathrm{cDLPN}} &\triangleq \mathcal{P}_{\mathrm{f}}(\mathcal{A}t) \\ \mathbf{C}_{\mathrm{cDLP}} &\triangleq \mathcal{P}_1(\mathcal{A}t) & \mathbf{C}_{\mathrm{cDLPN}} &\triangleq \mathcal{P}_1(\mathcal{L}it) \\ \mathbf{Q}_{\mathrm{cDLP}} &\triangleq \mathcal{P}_1(\mathcal{A}t) & \mathbf{Q}_{\mathrm{cDLPN}} &\triangleq \mathcal{P}_1(\mathcal{L}it). \end{aligned}$$

Notice that $\mathbf{P}_{\mathrm{cDLP}} \subseteq \mathbf{P}_{\mathrm{DLP}}$.

Following [LMR92, §2.3], a *disjunctive clause* is the universal closure of a logic formula like

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

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

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

or, equivalently (by De Morgan) to

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

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

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

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

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

Definition 3.5 ($\widehat{\mathcal{P}}$ and $\widehat{\phi}$)**.** Let $\mathcal{P}$ be a logic program. Then $\widehat{\mathcal{P}}$ is the program that results if we replace every unclean rule $\phi = (H, \langle D_1, \dots, D_n \rangle)$ of $\mathcal{P}$ by all clean rules in

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

We call $\widehat{\mathcal{P}}$ the *clean version* of $\mathcal{P}$. It follows that if $\mathcal{P} \in \mathbf{P}_{\text{DLP}}$ and $\mathcal{Q} \in \mathbf{P}_{\text{DLPN}}$, then $\widehat{\mathcal{P}} \in \mathbf{P}_{\text{cDLP}}$ and $\widehat{\mathcal{Q}} \in \mathbf{P}_{\text{cDLPN}}$.

This is a simple case of the most general *Lloyd–Topor transformation*, which transforms logic programs containing arbitrary formulæ in their bodies into normal ones. See [LT84], [Llo87, Ch. 4] or [LMR92, pp. 188–189] for more details.

▶ *Example 3.7.* Here is the desugaring of a program $\mathcal{Q} \in \mathbf{P}_{\text{DLP}}$:

$$\mathcal{Q} := \left\{ \begin{matrix} \texttt{e} \lor \texttt{p} \leftarrow \texttt{f} \lor \texttt{g}\,,\texttt{h} \\ \texttt{p} \lor \texttt{q} \leftarrow \texttt{g}\,,\texttt{e} \lor \texttt{r} \end{matrix} \right\} \in \mathbf{P}_{\text{DLP}} \quad \overset{\widehat{\ }}{\longmapsto} \quad \widehat{\mathcal{Q}} := \left\{ \begin{matrix} \texttt{e} \lor \texttt{p} \leftarrow \texttt{f}\,,\texttt{h} \\ \texttt{e} \lor \texttt{p} \leftarrow \texttt{g}\,,\texttt{h} \\ \texttt{p} \lor \texttt{q} \leftarrow \texttt{g}\,,\texttt{e} \\ \texttt{p} \lor \texttt{q} \leftarrow \texttt{g}\,,\texttt{r} \end{matrix} \right\} \in \mathbf{P}_{\text{cDLP}}.$$

◀

Once we have examined the semantics of logic programs in the next chapter, the following property will become apparent:

Property 3.1. *Let $\mathcal{P}$ be a logic program. $\widehat{\mathcal{P}}$ is clean and equivalent to $\mathcal{P}$.*

Almost all of the fundamental program constructions that we investigate preserve "cleanliness"; the one time that it really makes a difference is in Section **10.6**: there, we prove soundness and completeness for cDLP programs first, and then proceed to consider the general case of DLP.

We have explained what we mean by "logic program", and we have defined the syntax of four main logic programming languages. A study of their semantics comes next.

Chapter 4

Model-theoretic declarative semantics

In this chapter, we define the declarative (or denotational) semantics for the logic programming languages we are interested in:[1] We start with the least Herbrand model semantics for LP programs; then we proceed to present the minimal model semantics of Minker (see [Min82] or [LMR92]) which is the *de facto* denotational semantics for DLP programs. We are by no means thorough in this chapter but the interested reader will find pointers for further studying of the semantics presented in each section.

4.1 Definitions and notation

Definition 4.1. Given a *propositional* logic program $\mathcal{P}$, we define its *Herbrand base* $\mathrm{HB}(\mathcal{P})$ to be the set of all atoms that appear in $\mathcal{P}$.

Definition 4.2. Let $\mathcal{V}$ be a truth value space. By a *Herbrand $\mathcal{V}$-interpretation* I of $\mathcal{P}$, we mean any assignment of truth values to the elements of the Herbrand base, i.e., any function $I : \mathrm{HB}(\mathcal{P}) \to \mathcal{V}$. Thanks to the structure of $\mathcal{V}$, such a function I can be extended to all wffs generated from $\mathrm{HB}(\mathcal{P})$ with the logical connectives in $\{\vee, \wedge, \to, \sim\}$ in the straightforward, respectful manner, just like we did in Definition 2.2. We will use the same symbol I for the extended function. When the truth value space $\mathcal{V}$ is $\mathbb{B}$ or when it is obvious from the context, we may omit the prefix "$\mathcal{V}$-". We say that I satisfies a rule $H \leftarrow \mathcal{B}$, if $I(\mathcal{B} \to H) = \max \mathcal{V}$. A Herbrand interpretation that satisfies every rule of $\mathcal{P}$ is called a *Herbrand model* of $\mathcal{P}$.

Definition 4.3. It is customary in the logic programming literature to identify interpretations with sets of atoms when the truth value space is $\mathbb{B}$: for any interpretation $I : \mathrm{HB}(\mathcal{P}) \to \mathbb{B}$, we write

$$a \in I \overset{\triangle}{\iff} I(a) = \mathsf{T}.$$

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

Influenced by this, in any truth value space $\mathcal{V}$, we may use $\emptyset$ to denote the interpretation $I : \mathrm{HB}(\mathcal{P}) \to \mathcal{V}$ defined by:

$$I(x) = \min \mathcal{V}, \qquad \text{for all } x \in \mathrm{HB}(\mathcal{P}),$$

i.e., the bottom element $\perp$ of the poset $(\mathrm{HB}(\mathcal{P}) \to \mathcal{V})$, equipped with the pointwise ordering.

4.2 LP—the least Herbrand model

For LP programs, $\mathbb{B}$ will be the truth value space. LP programs enjoy the following very useful property:

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

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

$$\mathrm{LHM}(\mathcal{P}) \triangleq \bigcap \mathrm{HM}(\mathcal{P}).$$

Definition 4.4 (Least Herbrand model semantics)**.** Let $\mathcal{P}$ be an LP program. The goal $\leftarrow \mathtt{p}$ *succeeds* if $p \in \mathrm{LHM}(\mathcal{P})$.

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

Theorem 4A. *Let $\mathcal{P}$ be an LP program. Then*

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

Proof. See [vEK76, §5] or [Llo87, Theorem 6.2]. ▮

This concludes our brief summary of LP semantics; consult [Llo87] for more information.

4.3 DLP—the minimal models

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

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

First, we compute its Herbrand models:

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

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

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

And $\{p\}$ is not a model, since according to the third rule of our program, at least one of the atoms a or b must be true. However, the first two models are $\subseteq$-*minimal.* In fact, we can rely on the set $\{\{a,p\},\{b,p\}\}$ of minimal models to obtain a meaningful semantics for $\mathcal{P}$.

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

Definition 4.5 (Minimal model semantics)**.** Let $\mathcal{P}$ be a DLP program. The goal $\leftarrow$ `G` *succeeds* if `G` is true in every minimal Herbrand model of $\mathcal{P}$.

The equivalent of Theorem 4A for the disjunctive case is due to Minker:

Theorem 4B. *Let $\mathcal{P}$ be a DLP program. A clause C is a logical consequence of $\mathcal{P}$ iff C is true in every minimal Herbrand model of $\mathcal{P}$.*

Proof. See [Min82] or [LMR92, Theorem 3.3]. ∎

▶ *Example 4.1.* Consider the DLP program

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

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

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

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

$$\begin{array}{rcr} \{a,p\} \models p & \multirow{2}{*}{and} & \{a,p\} \models a \lor c \\ \{c,b,p\} \models p & & \{c,b,p\} \models a \lor c, \end{array}$$

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

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

Remark 4.1. In [LMR92], 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.

We have thus given meaning to disjunctive programs. It is time to do so for ones with negation.

4.4 LPN—the well-founded model

As we have already mentioned, interpreting negation in logic programs is a rather controversial subject. Certainly, *negation-as-failure* (see [Cla78]) is the computational interpretation with the most fruitful applications, and no other interpretation will concern us here. But even if we focus only on negation-as-failure, formally giving semantics to it is neither trivial, nor a problem with a unique satisfying solution. It is fair, however, to state that there have been two dominant and widely accepted, model-theoretic solutions: the *stable model semantics*, and the *well-founded model semantics.*

Negation-as-failure. Given a logic programming system, the main idea behind this approach to negation is that a goal $\leftarrow {\sim}\mathtt{p}$ should succeed, in case the goal $\leftarrow \mathtt{p}$ terminates with failure. For example, given the program

$$\mathcal{P} = \left\{ \begin{array}{l} \mathtt{p} \leftarrow \\ \mathtt{q} \leftarrow {\sim}\mathtt{p} \\ \mathtt{r} \leftarrow {\sim}\mathtt{q} \end{array} \right\},$$

the query $\leftarrow \mathtt{q}$ terminates with failure (because the goal $\leftarrow \mathtt{p}$ terminates with success), which in turn means that the query $\leftarrow \mathtt{r}$ succeeds. This is an operationally simple to explain idea, but defining declarative semantics for it proved to be a difficult problem; see [AB94] for a survey of the various attempts.

The stable model semantics. This solution was proposed in [GL88], and assigns to each program $\mathcal{P}$ a certain set of well-behaved models, called *stable models.* A program may have zero, one, or more such models, which is in this approach the price to pay in order to support negation. This semantics is compatible with the least Herbrand model semantics of LP, in the sense that if negations do not appear in $\mathcal{P}$, then there is only one stable model: the least Herbrand model of $\mathcal{P}$. This school of programming eventually lead to what is now known as *answer set programming* (ASP), which deviated from traditional logic programming practices and its theory and applications lie beyond this thesis; consult [Gel08] for further information.

The well-founded semantics. This semantics was introduced in [VGRS91], and actually assigns to each program $\mathcal{P}$ a *unique* model, called *well-founded.* But there is a different price to pay here: the underlying logic is shifted from two-valued, to many-valued. The original formulation was made using a three-valued logic, which contained an additional truth value, U, representing the unknown truth value. This approach is also compatible with the least Herbrand model semantics, since lack of negations in $\mathcal{P}$ implies that no element of its well-founded model will have the truth value U, and this two-valued model is indeed the least Herbrand model of $\mathcal{P}$. Many years after its appearance, an infinite-valued refinement of the well-founded semantics was presented in [RW05]. There is still one truth value representing the unknown, but the truth values T and F representing “true” and “false” respectively are each replaced by infinitely many truth values: we will use the $\mathbb{V}_\kappa$ spaces for this. From this new semantics, we can obtain the original well-founded model by collapsing all “false” values to F and all “true” values to T. It is indeed a refinement of the well-founded model, and in the sequel, we will refer to it simply as "the well-founded semantics".

The well-founded semantics of LPN

The truth value space we will use for LPN is $\mathbb{V}_\kappa$. For finite programs it will suffice to consider $\kappa := \omega$, while for the general, infinite case, we will set $\kappa := \omega_1$. We introduce the following concise notation:

Definition 4.6. let $f : X \to Y$ be a function and let $y \in Y$. Then

$$f \mid\mid y \triangleq f^{-1}(\{y\}).$$

Definition 4.7. Let I and J be two $\mathbb{V}_\kappa$-interpretations of a program $\mathcal{P}$, and let $\alpha \in \kappa$. We write

$$\begin{aligned} I =_\alpha J &\overset{\triangle}{\iff} \text{for all } \lambda \leq \alpha,\ I \mid\mid \mathsf{T}_\lambda = J \mid\mid \mathsf{T}_\lambda \text{ and } I \mid\mid \mathsf{F}_\lambda = J \mid\mid \mathsf{F}_\lambda \\ I \sqsubseteq_\alpha J &\overset{\triangle}{\iff} I \mid\mid \mathsf{T}_\alpha \subseteq J \mid\mid \mathsf{T}_\alpha,\ I \mid\mid \mathsf{F}_\alpha \supseteq J \mid\mid \mathsf{F}_\alpha,\ \text{and for all } \lambda < \alpha,\ I =_\lambda J \\ I \sqsubset_\alpha J &\overset{\triangle}{\iff} I \sqsubseteq_\alpha J \text{ and } I \neq_\alpha J. \end{aligned}$$

We also define:

$$\begin{aligned} I \sqsubset_\kappa J &\overset{\triangle}{\iff} \text{for some } \alpha \in \kappa,\ I \sqsubset_\alpha J \\ I \sqsubseteq_\kappa J &\overset{\triangle}{\iff} I = J \text{ or } I \sqsubset_\kappa J. \end{aligned}$$

The following proposition justifies the choice of notation:

Proposition 4.1. *On the set of $\mathbb{V}_\kappa$-interpretations*

(i) *for any $\alpha \in \kappa$, $=_\alpha$ is an equivalence relation.*

(ii) *for any $\alpha \in \kappa$, $\sqsubseteq_\alpha$ is a preorder (i.e., it is reflexive and transitive);*

(iii) *$\sqsubseteq_\kappa$ is a partial order (i.e., reflexive, transitive, and antisymmetric).*

Proof. (i) is trivial. (ii) Reflexivity is immediate from the definition, while transitivity follows directly from the transitivities of $\subseteq$ and $=_\alpha$. (iii) Reflexivity and antisymmetry are also immediate. For transitivity, assume that $I \sqsubset_\kappa J$ and $J \sqsubset_\kappa K$, so that there are ordinals α and β such that $I \sqsubset_\alpha J$ and $J \sqsubset_\beta K$. If $\alpha = \beta$, the result follows from the transitivity of $\sqsubset_\alpha$. Otherwise assume without loss of generality that $\alpha < \beta$, so that $I \sqsubset_\alpha J =_\alpha K$, which implies that $I \sqsubset_\alpha K$. By the definition, this means that $I \sqsubset_\kappa K$, as we wanted. ▮

Next, we define a very useful operator acting on interpretations of programs:

Definition 4.8. Let $\mathcal{P}$ be an LPN program and let I be an interpretation. The operator $T_\mathcal{P}$ is defined by:

$$T_\mathcal{P}(I)(p) \triangleq \max \{I(\mathcal{B}) \mid (\{p\}, \mathcal{B}) \in \mathcal{P}\}.$$

We call $T_\mathcal{P}$ the *immediate consequence operator* of $\mathcal{P}$.

Given an LPN program $\mathcal{P}$, its well-founded model $\mathrm{WFM}_\kappa(\mathcal{P})$ is constructed in stages, and it is best described by using $T_\mathcal{P}$: [2] we will approximate $\mathrm{WFM}_\kappa(\mathcal{P})$ by interpretations, starting with the least element $\perp$ of them all, i.e., the interpretation $\emptyset$ which is the constant function with value $\mathbf{F}_0$. We repeatedly iterate $T_\mathcal{P}$ until the set of atoms with value $\mathbf{F}_0$ and the set of atoms with value $\mathbf{T}_0$ have both stabilized, i.e., after, say, m iterations, we have

$$T_\mathcal{P}^m(I) \,||\, \mathbf{F}_0 = T_\mathcal{P}^{m+1}(I) \,||\, \mathbf{F}_0 \qquad \text{and} \qquad T_\mathcal{P}^m(I) \,||\, \mathbf{T}_0 = T_\mathcal{P}^{m+1}(I) \,||\, \mathbf{T}_0.$$

Once this happens, we get our next interpretation by keeping the values of these atoms the same, while resetting all other atoms to the “next” false value, in this case $\mathbf{F}_1$. We proceed in a similar way until the sets of atoms with values $\mathbf{F}_1$ and $\mathbf{T}_1$ have both stabilized as well, and reset the remaining atoms to the value $\mathbf{F}_2$. Atoms that do not get a value by following this procedure are set to be $\mathbf{U}$. Let us see how this procedure works for a concrete example, taken from [RW05]:

▶ *Example 4.2.* Consider the LPN program

$$\mathcal{P} = \left\{ \begin{array}{l} \mathtt{p} \leftarrow \mathtt{\sim q} \\ \mathtt{q} \leftarrow \mathtt{\sim r} \\ \mathtt{s} \leftarrow \mathtt{p} \\ \mathtt{s} \leftarrow \mathtt{\sim s} \end{array} \right\}$$

We start the process:

$$\begin{aligned} \emptyset &= \{(p,\mathbf{F}_0),(q,\mathbf{F}_0),(r,\mathbf{F}_0),(s,\mathbf{F}_0)\} \\ T_\mathcal{P}(\emptyset) &= \{(p,\mathbf{T}_1),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{T}_1)\} \\ T^2_\mathcal{P}(\emptyset) &= \{(p,\mathbf{F}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{T}_1)\}\,. \end{aligned}$$

At this point, the values of order 0 have been stabilized. We reset the remaining values to $\mathbf{F}_1$ and continue in the same manner:

$$\begin{aligned} I_0 &= \{(p,\mathbf{F}_1),(q,\mathbf{F}_1),(r,\mathbf{F}_0),(s,\mathbf{F}_1)\} \\ T_\mathcal{P}(I_0) &= \{(p,\mathbf{T}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{T}_2)\} \\ T^2_\mathcal{P}(I_0) &= \{(p,\mathbf{F}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{T}_2)\}\,. \end{aligned}$$

Now the values of order 1 have been stabilized. Like before, we reset the remaining values, now to $\mathbf{F}_2$:

$$\begin{aligned} I_1 &= \{(p,\mathbf{F}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{F}_2)\} \\ T_\mathcal{P}(I_1) &= \{(p,\mathbf{F}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{T}_3)\} \\ T^2_\mathcal{P}(I_1) &= \{(p,\mathbf{F}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{F}_4)\}\,, \end{aligned}$$

and values of order 2 have been stabilized. Resetting the remaining vales to $\mathbf{F}_3$, we compute:

$$\begin{aligned} I_2 &= \{(p,\mathbf{F}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{F}_3)\} \\ T_\mathcal{P}(I_2) &= \{(p,\mathbf{F}_2),(q,\mathbf{T}_1),(r,\mathbf{F}_0),(s,\mathbf{T}_4)\}\,. \end{aligned}$$

[2] Readers familar with *stratification* will probably recognize the similarities.

Observe now that values of order 3 have disappeared completely, which means that s cannot obtain any value of order less than ω, so we set its value to U, finally reaching the well-founded model:

$$\mathrm{WFM}_\kappa(\mathcal{P}) = \{(p, \mathsf{F}_2), (q, \mathsf{T}_1), (r, \mathsf{F}_0), (s, \mathsf{U})\}\,. \qquad \blacktriangleleft$$

The $\mathbb{V}_\kappa$-valued well-founded model has the following nice property, proved in [RW05]:

Theorem 4C. *Let $\mathcal{P}$ be an LPN program. Then $\mathrm{WFM}_\kappa(\mathcal{P})$ is the $\sqsubseteq_\kappa$-least model of $\mathcal{P}$.*

Given $\mathrm{WFM}_\kappa(\mathcal{P})$, it is immediate to obtain the original, three-valued well-founded model $\mathrm{WFM}(\mathcal{P})$ as defined in [VGRS91]. The following theorem, proven in [RW05], tells us how:

Theorem 4D. *Let $\mathcal{P}$ be an LPN program, and let $p \in \mathrm{HB}(\mathcal{P})$. Then*

$$\mathrm{WFM}(\mathcal{P}) = \mathit{collapse} \circ \mathrm{WFM}_\kappa(\mathcal{P}).$$

4.5 DLPN—the infinite-valued minimal models

In [CPRW07], the approach studied above was extended to be applicable to *finite* DLPN programs as well. We impose the same restriction as in [CPRW07] and consider only finite programs throughout this section. This also implies that $\mathbb{V}_\omega$ is a sufficiently large truth value space.

The shift from definite to disjunctive in the case of negation-free programs resulted in the "compromise" of using a *set* of minimal models, instead of a single least one. The price to pay is the same as we shift from LPN to DLPN programs, although in this case, naturally, these minimal models are many-valued. The ideas involved are rather intuitive, and so we directly jump to an example before stating the results that we are interested in:

▶ *Example 4.3.* Consider the program

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

The only fact in this program will force any model M to include (b, T_0) or (p, T_0). In the first case, we immediately obtain the minimal model

$$M_1 := \{(a, \mathsf{F}_0), (b, \mathsf{T}_0), (p, \mathsf{F}_0)\}\,.$$

In the second case, the fact that p is T_0 implies that the value of ${\sim}p$ must be F_1, which in turn forces either either a or b to have a value at least as big as F_1, so we reach two more minimal models:

$$\begin{aligned} M_2 &:= \{(a, \mathsf{F}_1), (b, \mathsf{F}_0), (p, \mathsf{T}_0)\} \\ \text{and} \quad M_3 &:= \{(a, \mathsf{F}_0), (b, \mathsf{F}_1), (p, \mathsf{T}_0)\}\,. \end{aligned}$$

Therefore, the set of $\mathbb{V}_\omega$-valued minimal models of $\mathcal{P}$ is

$$\mathrm{MM}_\omega(\mathcal{P}) = \{M_1, M_2, M_3\}\,. \qquad \blacktriangleleft$$

The following are the main results regarding this semantics:

Theorem 4E. *Every DLPN program $\mathcal{P}$ has a non-empty,* finite *set of $\sqsubseteq_\omega$-minimal, $\mathbb{V}_\omega$-valued models* $\mathrm{MM}_\omega(\mathcal{P})$.

Proof. See [CPRW07]. ∎

It is argued in [CPRW07] that this set of minimal models captures the intended meaning of a DLPN program, and it is implied that it indeed provides a semantics in an analogous way to the DLP case: the value of a query G can be obtained as the minimum of the values that G has in each of those minimal models.

Theorem 4F. *Let $\mathcal{P}$ be a DLPN program that contains n propositional variables in its rules, and let M be a minimal model of $\mathcal{P}$. Then* $\mathrm{Range}(M) \subseteq \mathbb{V}_n$.

Proof. See [CPRW07]. ∎

The main reason why we focus on this semantics for DLPN programs is the fact that it is purely model-theoretic and it naturally extends both the well-founded semantics of LPN and the minimal model semantics of DLP, as the following theorem states:

Theorem 4G. *Let $\mathcal{P}$ be a DLPN program. Then*

- *if $\mathcal{P}$ is an LPN program,* $\mathrm{MM}_\omega(\mathcal{P}) = \mathrm{WFM}_\omega(\mathcal{P})$;
- *if $\mathcal{P}$ is a DLP program,* $\mathrm{MM}_\omega(\mathcal{P}) = \mathrm{MM}(\mathcal{P})$.

Proof. See [CPRW07]. ∎

4.6 First-order programs vs. infinite propositional ones

Even though we focus completely on propositional logic programs, we summarize below the corresponding notions of the ones we described above, for the first-order case. They are presented in detail in [LMR92, §2.2, §2.4, §3.2]. The reader can safely skip this section on a first reading, coming back to it only before the very last result of Section **10.6**, viz. Corollary 10.33.

Definition 4.9. The *Herbrand universe* of $\mathcal{L}_1$, HU, is the set of all ground terms which can be formed out of the constant and function symbols of $\mathcal{L}_1$. The *Herbrand base* of a logic program $\mathcal{P}$, $\mathrm{HB}(\mathcal{P})$, is the set of all ground atoms which can be formed using the predicate symbols that appear in $\mathcal{P}$ on the ground terms of HU.

Definition 4.10. A *Herbrand interpretation* of $\mathcal{L}_1$ assigns: to each n-ary predicate symbol R of Pred_n, an n-ary function from HU^n to $\mathbb{B}$; to each n-ary function symbol $f \in \mathit{Fun}_n$ the function that maps the element $(t_1, \dots, t_n) \in \mathrm{HU}^n$ to the term $f(t_1, \dots, t_n) \in \mathit{Term}$; and to each constant symbol, itself. A Herbrand interpretation is naturally extended to evaluate all formulæ of $\mathcal{L}_1$ according to the usual truth tables. If a Herbrand interpretation I is a model of $\mathcal{P}$ we call it a *Herbrand model* of $\mathcal{P}$.

It is not difficult to see that infinite, propositional programs are as powerful as finite, first-order ones. Hence, for simplicity, we will be dealing with propositional logic programs unless mentioned otherwise. To see how we end up with infinite programs, start from a non-propositional, finite program $\mathcal{P}$, containing at least one function symbol, and replace each of its rules by all of its ground instances. What you get is a countably infinite program with equivalent denotational semantics. This is exactly what happens in Example 3.3 (p. 16).

▶ *Example 4.4.* Let us confine ourselves to propositional (albeit infinite) programs. Then, instead of the finite first-order program $\mathcal{E}$ of Example 3.3 (repeated here for convenience together with $\mathsf{ground}(\mathcal{E})$), we could write, for instance, the infinite propositional program $\mathcal{E}_0$:

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

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

in which we have chosen a propositional variable $\mathtt{e}_i$ for each first-order atomic formula $\mathtt{even}(\mathsf{S}^i(0))$ of $\mathsf{ground}(\mathcal{E})$. ◀

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

Property 4.2. *Let $\mathcal{P}$ be a first-order DLP. Then $\mathcal{P}$ and $\mathsf{ground}(\mathcal{P})$ have the same minimal models:*

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

☡ Remark 4.2. As long as we are studying denotational semantics, the above property allows us to focus on infinite, propositional programs. This is a very common practice in the field of logic programming semantics. In fact, we assume given an implementation of our programming language (based on some operational semantics—which, is immaterial). We then load our finite, first-order program $\mathcal{P}$, and the implementation computes answers to our queries. Then, a denotational semantics of the equivalent, propositional, and infinite DLP program $\mathsf{ground}(\mathcal{P})$, provides a correctness criterion for those answers. Thus, we avoid variables and function symbols at the cost of finiteness, but this is a fair bargain, as no difficulties arise on the declarative side of semantics (see also [Fit99] for a relevant discussion). In exactly the same sense, when we are giving a fixpoint semantics, we only use the set of ground instances of clauses in $\mathcal{P}$ to define $T_{\mathcal{P}}$ (see [Llo87] and [LMR92]).

We have defined the syntax and the model-theoretic declarative semantics of the logic programming languages that interest us. In the next part, we define an abstract framework for semantics and develop a toolkit for dealing with disjunctive programs. The goal is to describe an operator that acts on semantics of non-disjunctive languages, and yields a new semantics for a corresponding disjunctive language.

Part II

From non-disjunctive to disjunctive semantics

Chapter 5

An abstract framework for semantics

In order to study logic programming languages and their semantics, we introduce in this chapter a formal framework of semantics and examine the four languages we have met with respect to this framework.

5.1 The framework

Definition 5.1. Let L be a logic programming language, let $\mathcal{M}$ be a set whose elements we will call *meanings*, and let $\mathcal{V}$ be a truth value space. Then, an *$\mathcal{M}$-semantics for L* is a function

$$\mathbf{m} : \mathbf{P}_L \to \mathcal{M};$$

a *$\mathcal{V}$-answer function for $\mathcal{M}$* is a function

$$\mathbf{a} : \mathcal{M} \to \mathbf{Q}_L \to \mathcal{V};$$

and a *$\mathcal{V}$-system for L* is a function

$$\mathbf{s} : \mathbf{P}_L \to \mathbf{Q}_L \to \mathcal{V}.$$

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

𝄾 Remark 5.1. Composing a $\mathcal{V}$-answer function for $\mathcal{M}$ with an $\mathcal{M}$-semantics for L we obtain a $\mathcal{V}$-system for L. Therefore, a semantics $(\mathbf{m}, \mathbf{a})$ naturally gives rise to the $\mathcal{V}$-system $\mathbf{a} \circ \mathbf{m}$. In this way, we will be able to use $(\mathbf{m}, \mathbf{a})$ in a context where a $\mathcal{V}$-system is expected.

Definition 5.2 ($\approx$)**.** Let L be a logic programming language. We call two semantics of L $(\mathbf{m}_1, \mathbf{a}_1)$ and $(\mathbf{m}_2, \mathbf{a}_2)$ *equivalent* iff the corresponding $\mathcal{V}$-systems are equal. In symbols,

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

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

Definition 5.3. Let L be a logic programming language. We say that $(\mathbf{m}_1, \mathbf{a}_1)$ *refines* $(\mathbf{m}_2, \mathbf{a}_2)$ with respect to the operator $\mathcal{C}$ iff:

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

If only $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{a}_2$, and $\mathcal{C}$ are given, we write $\mathbf{m}_1 \lhd_{\mathcal{C}}^{\mathbf{a}_2} \mathbf{m}_2$ for $(\mathbf{m}_1, \mathbf{a}_2 \circ \mathcal{C}) \lhd_{\mathcal{C}} (\mathbf{m}_2, \mathbf{a}_2)$, and might even omit the superscript $\mathbf{a}_2$ when it is obvious or implicit by the context.

Lemma 5.1. *The following implication holds:*

$$(\mathbf{m}_1, \mathbf{a}_1) \lhd_{\mathcal{C}} (\mathbf{m}_2, \mathbf{a}_2) \implies (\mathbf{m}_1, \mathbf{a}_1) \approx (\mathbf{m}_2, \mathbf{a}_2). \tag{5.1.1}$$

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

5.2 Semantics of LP

LHM: the least Herbrand model semantics

This is the *least Herbrand model semantics* we met in Section **4.2**.

- $\mathcal{V}_{\mathsf{LHM}} \triangleq \mathbb{B}$.
- $\mathcal{M}_{\mathsf{LHM}}$ is the set of all possible Herbrand interpretations.
- $\mathbf{m}_{\mathsf{LHM}}$ maps an LP program to its least Herbrand model.
- $\mathbf{a}_{\mathsf{LHM}}(M)(p) \triangleq \begin{cases} \mathsf{T}, & \text{if } p \in M \\ \mathsf{F}, & \text{otherwise.} \end{cases}$

5.3 Semantics of DLP

MM: the minimal model semantics

This is the *minimal models semantics* of Section **4.3**.

- $\mathcal{V}_{\mathsf{MM}} \triangleq \mathbb{B}$.
- $\mathcal{M}_{\mathsf{MM}}$ consists of all sets of Herbrand interpretations.
- $\mathbf{m}_{\mathsf{MM}}$ maps a DLP program to the set of its minimal models.
- $\mathbf{a}_{\mathsf{MM}}(\mathscr{M})(Q) \triangleq \begin{cases} \mathsf{T}, & \text{if } Q \text{ is true in every model } M \in \mathscr{M} \\ \mathsf{F}, & \text{otherwise.} \end{cases}$

5.4 Semantics of LPN

WF: the well-founded semantics

This is the *well-founded model semantics* for LPN, as described in Section **4.4**.

- $\mathcal{V}_{\mathsf{WF}} \triangleq \mathbb{V}_1 = \{\mathbf{F}, \mathbf{U}, \mathbf{T}\}$.
- $\mathcal{M}_{\mathsf{WF}}$ consists of all possible Herbrand $\mathbb{V}_1$-interpretations.
- $\mathbf{m}_{\mathsf{WF}}$ maps every LPN program to its three-valued, well-founded model.
- $\mathbf{a}_{\mathsf{WF}}(M)(p) \triangleq M(p)$.

WF$^\kappa$: the infinite-valued well-founded semantics

This is the *infinite-valued well-founded semantics* for LPN, also described in Section **4.4**.

- $\mathcal{V}_{\mathsf{WF}^\kappa} \triangleq \mathbb{V}_\kappa$.
- $\mathcal{M}_{\mathsf{WF}^\kappa}$ consists of all possible Herbrand $\mathbb{V}_\kappa$-interpretations of LPN programs.
- $\mathbf{m}_{\mathsf{WF}^\kappa}$ maps every LPN program to its $\mathbb{V}_\kappa$-valued, well-founded model.
- $\mathbf{a}_{\mathsf{WF}^\kappa}(M)(p) \triangleq M(p)$.

𝄿 Remark 5.2 (The ordinal κ). The ordinal κ that we use in the truth value spaces $\mathbb{V}_\kappa$ may vary, depending on our needs. The reader should note at this point that if the programs are finite, an ordinal as small as ω suffices to give us satisfying semantics, in the sense that collapsing the obtained $\mathbb{V}_\omega$-valued model to a three-valued one will always yield the desired well-founded model. See [RW05] and [Lüd11] for more information.

5.5 Semantics of DLPN

MM$^\omega$: the infinite-valued minimal model semantics

The infinite-valued minimal model semantics, which we introduced in Section **4.5**, and apply for finite, propositional DLPN programs.

- $\mathcal{V}_{\mathsf{MM}^\omega} \triangleq \mathbb{V}_\omega$.
- $\mathcal{M}_{\mathsf{MM}^\omega}$ consists of all possible $\mathbb{V}_\omega$-valued Herbrand interpretations of DLPN programs.
- $\mathbf{m}_{\mathsf{MM}^\omega}$ maps every DLPN program to the set of its minimal, infinite-valued models.
- $\mathbf{a}_{\mathsf{MM}^\omega}(\mathscr{M})(q) \triangleq \bigwedge \{M(q) \mid M \in \mathscr{M}\}$.

𝄾

Chapter 6

Disjunctive operations

In this chapter, we define some operations that can be applied to various disjunctive components of logic programs, for example rules, or even whole programs. We will extensively rely on these operations in the sequel.

6.1 Restricting

Definition 6.1 (Rule restriction)**.** For any given rule ϕ and any set of atoms A, we define the *restriction* of ϕ to A by

$$\phi|_A \triangleq (\mathsf{head}(\phi) \cap A, \mathsf{body}(\phi))\,.$$

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

Definition 6.2 (Program restriction)**.** Let $\mathcal{P}$ be a DLP or DLPN program and let $\phi \in \mathcal{P}$. Then for any set of atoms A, we can define the *restricted program* $\mathcal{P}|_A^\phi$ by restricting the rule ϕ of $\mathcal{P}$ to A:

$$\mathcal{P}|_A^\phi \triangleq (\mathcal{P} \setminus \{\phi\}) \cup \{\phi|_A\}\,.$$

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

► *Example 6.1.* Let $\mathcal{P}$ be the proper DLP program

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

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

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

We observe the following useful property:

Property 6.1. *The restriction of a rule ϕ is stronger than the original rule, in the sense that any interpretation which satisfies $\phi|_A$ must also satisfy ϕ. In the same sense, restricting a disjunctive program makes it stronger.*

6.2 Splitting

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

Definition 6.3 (Proper partition)**.** Suppose that H is a set of atoms. Then a tuple $\mathcal{H} := (H_1, \dots, H_n)$ is a proper partition of H iff

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

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

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

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

Definition 6.4 (Splitting a program)**.** Let ϕ be a proper disjunctive rule, and let $\mathcal{H} := (H_1, \dots, H_n)$ be a proper partition of its head. Then the *splitting of* $\mathcal{P}$ *with respect to* ϕ *over* $\mathcal{H}$ is the tuple

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

► *Example 6.2.* The pair $\left(\mathcal{P}|^{\phi}_{\{p,q\}}, \mathcal{P}|^{\phi}_{\{r\}}\right)$ of Example 6.1 is the splitting of $\mathcal{P}$ with respect to ϕ over $\mathcal{H} := (\{p, q\}, \{r\})$. ◄

To prove the main result of Chapter **10** 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 a DLP program* $\mathcal{P}$ *with respect to* ϕ *over* (H_1, H_2)*. Then*

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

Proof. Let $\phi_1 := \phi|_{H_1}$ and $\phi_2 := \phi|_{H_2}$. For the first inclusion, let $S \in \mathrm{MM}(\mathcal{P})$, and suppose $S \notin \mathrm{MM}(\mathcal{P}_1)$. We need $S \in \mathrm{MM}(\mathcal{P}_2)$. There are two ways in which S can fail to be in $\mathrm{MM}(\mathcal{P}_1)$: either it is a model but not a minimal one, or it is not even a model to begin with.
CASE 1: *S is a non-minimal model of $\mathcal{P}_1$.* There exists then, a proper submodel $S_0 \subsetneq S$ of $\mathcal{P}_1$, with $S_0 \models \mathcal{P}_1$. By definition and Property 6.1, this would also be a model of $\mathcal{P}$, and therefore S would not be minimal in $\mathcal{P}$, which is a contradiction, and so this case may never be.
CASE 2: *S is not a model of $\mathcal{P}_1$.*

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

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

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

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

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

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

As $\mathcal{P}$ is a weaker program than its restrictions (by Property 6.1), we also have the inclusions $\mathrm{HM}(\mathcal{P}_1) \subseteq \mathrm{HM}(\mathcal{P})$ and $\mathrm{HM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P})$. Hence, by taking unions for one last time,

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

Putting (1)–(3) together:

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

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

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

Then the corresponding sets of Herbrand models and minimal models are

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

so that in this example all inclusions are proper:

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

6.3 Combinations

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

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

Definition 6.5 (Combination of disjunctions)**.** The (disjunctive) *combination* of two disjunctions is simply their union:

$$D \curlyvee E \triangleq D \cup E.$$

Definition 6.6 (Combination of conjunctions)**.** Given two conjunctions of disjunctions $\mathscr{D} \coloneqq \langle D_1, \ldots, D_n \rangle$ and $\mathscr{E} \coloneqq \langle E_1, \ldots, E_m \rangle$, their *combination* is another conjunction of disjunctions, logically equivalent to $\mathscr{D} \vee \mathscr{E}$, and defined by the following equation:

$$\mathscr{D} \curlyvee \mathscr{E} \triangleq \langle D_1 \curlyvee E_1, \ldots, D_1 \curlyvee E_m, \ldots, D_n \curlyvee E_1, \ldots, D_n \curlyvee E_m \rangle .$$

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

▶ *Example 6.3.* Combining the following two conjunctions of disjunctions

$$\begin{array}{rl} \texttt{p}\,\texttt{,}\,\texttt{q} \vee \texttt{r}\,\texttt{,}\,\texttt{q} \vee \texttt{b} & \text{i.e.,} \quad \langle \{p\}, \{q, r\}, \{b, q\} \rangle \\ \texttt{a} \vee \texttt{b}\,\texttt{,}\,\texttt{a} \vee \texttt{r} & \text{i.e.,} \quad \langle \{a, b\}, \{a, r\} \rangle \end{array}$$

in the given order, we obtain the conjunction

$$\texttt{p} \vee \texttt{a} \vee \texttt{b}\,\texttt{,}\,\texttt{p} \vee \texttt{a} \vee \texttt{r}\,\texttt{,}\,\texttt{q} \vee \texttt{r} \vee \texttt{a} \vee \texttt{b}\,\texttt{,}\,\texttt{q} \vee \texttt{r} \vee \texttt{a}\,\texttt{,}\,\texttt{q} \vee \texttt{b} \vee \texttt{a}\,\texttt{,}\,\texttt{q} \vee \texttt{b} \vee \texttt{a} \vee \texttt{r},$$

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

Definition 6.7 (Combination of rules)**.** The *combination* of two disjunctive rules $\phi_1 \coloneqq (H_1, \mathscr{D}_1)$ and $\phi_2 \coloneqq (H_2, \mathscr{D}_2)$ is the rule defined by

$$\phi_1 \curlyvee \phi_2 \triangleq (H_1 \curlyvee H_2, \mathscr{D}_1 \curlyvee \mathscr{D}_2).$$

☡ Remark 6.2. It follows that $\phi_1 \curlyvee \phi_2$ will not be a clean rule in general, unless one of the ϕ_i's is a fact. This is the only construction that does not preserve cleanliness. However, we will only use it to extract the head of the combined rule (which causes no trouble),[1] and not to create new, potentially unclean rules.

▶ *Example 6.4.* Combining the following two rules

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

we obtain

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

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

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

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

Definition 6.8 (Combining sequences)**.** Given a sequence $\langle T_1, \dots, T_n \rangle$ of combinable disjunctions, conjunctions, or rules, we set

$$[\,\langle T_1, \dots, T_n \rangle\,] \triangleq T_1 \curlyvee \cdots \curlyvee T_n,$$

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

An alternative presentation of the same definition uses recursion:

$$[\,\langle\,\rangle\,] \triangleq \begin{cases} \emptyset & \text{(disjunctions)} \\ \langle \emptyset \rangle & \text{(conjunctions of disjunctions)} \\ (\emptyset, \langle \emptyset \rangle) & \text{(disjunctive rules)} \end{cases}$$

$$[\,\langle T_1, \dots, T_{n+1} \rangle\,] \triangleq [\,\langle T_1, \dots, T_n \rangle\,] \curlyvee T_{n+1}.$$

6.4 Definite instantiations and D-sections

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

Definition 6.9. Let $\phi = (H, \mathcal{B})$ be a disjunctive rule. If $h \in H$, then the definite rule $(h, \mathcal{B})$ is a *definite instantiation of* ϕ. $\mathrm{D}(\phi)$ is the set of all definite instantiations of ϕ.

▶ *Example 6.5.* Here are some disjunctive rules and their respective sets of definite instantiations:

$$\phi_1 \coloneqq \mathtt{a} \vee \mathtt{b} \leftarrow \mathtt{p}\,\mathtt{,}\,{\sim}\mathtt{q} \qquad \mathrm{D}(\phi_1) = \left\{ \begin{array}{l} \mathtt{a} \leftarrow \mathtt{p}\,\mathtt{,}\,{\sim}\mathtt{q} \\ \mathtt{b} \leftarrow \mathtt{p}\,\mathtt{,}\,{\sim}\mathtt{q} \end{array} \right\},$$

$$\phi_2 \coloneqq \mathtt{e} \vee \mathtt{f} \vee \mathtt{g} \leftarrow \qquad \mathrm{D}(\phi_2) = \left\{ \begin{array}{l} \mathtt{e} \leftarrow \\ \mathtt{f} \leftarrow \\ \mathtt{g} \leftarrow \end{array} \right\},$$

$$\phi_3 \coloneqq \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b} \vee \mathtt{c} \qquad \mathrm{D}(\phi_3) = \left\{ \begin{array}{l} \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b} \vee \mathtt{c} \\ \mathtt{p} \vee \mathtt{q} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b} \vee \mathtt{c} \end{array} \right\}. \qquad ◀$$

Definition 6.10. Let $\mathcal{D} = \{(H_i, \mathcal{B}_i)\}_{i \in I}$ be a disjunctive program, indexed by some set of indices I. A $\mathcal{D}$*-section* is any choice function $f \in \prod_{i \in I} H_i$. We write $\mathrm{S}(\mathcal{D})$ for the set of all $\mathcal{D}$-sections. If f is a $\mathcal{D}$-section, we define the *definite instantiation of* $\mathcal{D}$ *under* f to be the definite program

$$\mathcal{D}_f \triangleq \{(\{f(i)\}\,, B_i)\}_{i \in I}\,.$$

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

▶ *Example 6.6.* Consider the disjunctive program

$$\mathcal{D} \coloneqq \left\{ \begin{array}{lr} \phi_1 \coloneqq & \mathtt{s} \vee \textcircled{\texttt{t}} \leftarrow \mathtt{p}\,\mathtt{,}\,\mathtt{b} \vee \mathtt{c} \\ \phi_2 \coloneqq & \textcircled{\texttt{a}} \vee \mathtt{b} \leftarrow \qquad\quad \\ \phi_3 \coloneqq & \textcircled{\texttt{p}} \leftarrow \mathtt{a} \qquad\quad \\ \phi_4 \coloneqq & \textcircled{\texttt{p}} \leftarrow \mathtt{b} \qquad\quad \\ \phi_5 \coloneqq & \textcircled{\texttt{b}} \vee \mathtt{c} \leftarrow \qquad\quad \end{array} \right\}.$$

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

$$\begin{aligned} f &\coloneqq \{(1,t),(2,a),(3,p),(4,p),(5,b)\} \\ g &\coloneqq \{(1,s),(2,b),(3,p),(4,p),(5,b)\}\,. \end{aligned}$$

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

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

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

This completes our toolkit that will aid us in manipulating disjunctive programs. It is now time to put these tools into use: in the next chapter, we show how we can extend any semantics of a definite language into a semantics of a corresponding disjunctive language, and in Part **III** we will also use them extensively to define a novel game semantics for DLP and prove its soundness and correctness.

Chapter 7

The abstract transformation $(-)^\vee$

In this chapter we define the $(-)^\vee$ operator, which transforms any given semantics of a non-disjunctive logic programming language into a semantics for the "corresponding" disjunctive one. For reasons of simplicity, *we will assume that all programs in this chapter are clean.* This does not really impose any substantial restriction: for an unclean program $\mathcal{D}$, we simply use the semantics of its equivalent, clean version $\widehat{\mathcal{D}}$ (see Definition 3.5).

7.1 Definitions and theory

Definition 7.1 ($(-)^\vee$)**.** The operator $(-)^\vee$ is an overloaded operator that can be applied to:

(1) $\mathcal{M}$-meanings of LP[N] programs:

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

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

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

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

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

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

Theorem 7A. *Let $\mathcal{V}$ be a truth value space, $\mathcal{D}$ a DLP program, G a DLP goal, and I a $\mathcal{V}$-interpratation for $\mathcal{D}$. Then*

$$I\big(\bigwedge \mathcal{D} \to \bigvee G\big) = \bigwedge\nolimits_{\mathcal{P}\in \mathrm{D}(\mathcal{D})} \bigvee\nolimits_{g\in G} I\big(\bigwedge \mathcal{P} \to g\big).$$

Proof. Pick a set of indices J to index $\mathcal{D}$, and denote its rules by R_j, each having a head H_j and a body $\mathcal{B}_j$, so that $\mathcal{D} := \{R_j \mid j \in J\} = \{H_j \leftarrow \mathcal{B}_j \mid j \in J\}$. Now compute:

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

where each step of the computation is justified as follows: (1) by assumption for $\mathcal{D}$ and G; (2) by assumption for R_j; (3) by assumption for H_j; (4) by Property A.9(III); (5) by the fact that $\mathcal{V}$, as a truth value space, is completely distributive, and by the definition of $\mathrm{S}(\mathcal{D})$; (6) by Property A.9(IV); (7) by Property A.9(III) again; (8) by the definiton of $\mathcal{D}_f$; (9) by the definitions of $\mathrm{S}(\mathcal{D})$, $\mathrm{D}(\mathcal{D})$, and $\mathcal{D}_f$; and all steps marked by (*) follow from the fact that I is a $\mathcal{V}$-interpretation. ▌

☡ REMARK 7.1. Note that given a set of wffs A, thanks to the idempotence, commutativity and associativity of $\vee$ and $\wedge$, the various ways of forming a wff from the sets $\bigvee A$ and $\bigwedge A$ will lead to equal truth values when an interpretation acts on them. We have just made use of this on the proof above.

Lemma 7.1. *Let* L *be LP or LPN. Suppose that* $\mathcal{M}$ *is some set of meanings for* L *and* $\mathcal{V}$ *a truth value space. Let* $\mathbf{m}$ *and* $\mathbf{a}$ *be an* $\mathcal{M}$*-semantics and a* $\mathcal{V}$*-answer function for* L *respectively. Then*

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

or, following Remark 5.1, $(\mathbf{m}, \mathbf{a})^{\vee} = ((\mathbf{m})^{\vee}, (\mathbf{a})^{\vee})$*. It follows that if* $(\mathbf{m}_1, \mathbf{a}_1)$ *and* $(\mathbf{m}_2, \mathbf{a}_2)$ *are two semantics for* L*, then*

$$(\mathbf{m}_1, \mathbf{a}_1) \approx (\mathbf{m}_2, \mathbf{a}_2) \implies (\mathbf{m}_1, \mathbf{a}_1)^{\vee} \approx (\mathbf{m}_2, \mathbf{a}_2)^{\vee}. \tag{7.1.1}$$

Proof. We compute:

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

▮

Lemma 7.2. *Let* $\mathcal{V}$ *be a totally ordered, truth value space, and let* $\mathcal{D}$ *be a clean DLP (or DLPN) program. If* M *is a model of* $\mathcal{D}$*, then there is an LP (or LPN) program* $\mathcal{P} \in \mathrm{D}(\mathcal{D})$ *such that* M *is a model of* $\mathcal{P}$*. In symbols,*

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

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

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

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

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

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

As the following counterexample confirms, the above lemma fails to hold in general if we drop the condition that the truth value space has to be totally ordered:

► *Counterexample 7.1.* Consider the truth value space $\mathcal{V} \coloneqq \mathcal{P}(\{0,1\})$ ordered under $\subseteq$, whose elements we will denote by writing $\top$, L, R, and $\bot$, for $\{0,1\}$, $\{0\}$, $\{1\}$, and $\emptyset$ respectively. For the DLP program $\mathcal{D} \coloneqq \{\mathtt{a} \vee \mathtt{b} \leftarrow\ \}$, we compute

$$\mathrm{D}(\mathcal{D}) = \{\mathcal{P}_a, \mathcal{P}_b\}, \qquad \text{where} \qquad \begin{aligned} \mathcal{P}_a &\coloneqq \{\mathtt{a} \leftarrow\ \} \\ \mathcal{P}_b &\coloneqq \{\mathtt{b} \leftarrow\ \}. \end{aligned}$$

Here is a model M of $\mathcal{D}$ that is neither a model of $\mathcal{P}_a$, nor of $\mathcal{P}_b$:

$$M \coloneqq \{(a, \mathsf{L}), (b, \mathsf{R})\}.$$

In fact, $M(a \vee b) = M(a) \vee M(b) = \mathsf{L} \vee \mathsf{R} = \top$, but $M(a) = \mathsf{L} < \top$ and similarly $M(b) = \mathsf{R} < \top$. ◄

7.2 Applications and examples

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

From LP to DLP

Let us start with the simplest case of LP programs and their least Herbrand model semantics, LHM. As a reminder, we are dealing with $\mathcal{V}_{\mathsf{LHM}} = \mathbb{B}$, $\mathcal{M}_{\mathsf{LHM}}$ is the set of all possible Herbrand interpretations, and $\mathbf{m}_{\mathsf{LHM}}$ maps an LP program to its least Herbrand model. Finally, $\mathbf{a}_{\mathsf{LHM}}(M)(p)$ is $\mathbf{T}$ exactly when $p \in M$.

We first notice that using $(-)^{\vee}$ on LHM we obtain a semantics for DLP, which we will denote by $\mathsf{LHM}_{\vee}$. We have:

$$\begin{aligned} \mathcal{V}_{\mathsf{LHM}_{\vee}} &= \mathcal{V}_{\mathsf{LHM}} = \mathbb{B} \\ \mathcal{M}_{\mathsf{LHM}_{\vee}} &= \mathcal{P}(\mathcal{M}_{\mathsf{LHM}}). \end{aligned}$$

We proceed following the definitions:

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

Next we show the equivalence of this new, obtained semantics, with the traditional, minimal model semantics MM.

Theorem 7B. *The* $\mathsf{LHM}_{\vee}$ *and the* MM *semantics are equivalent.*

Proof. To exhibit the equivalence between the minimal model semantics MM and the obtained semantics $\mathsf{LHM}_{\vee}$, we appeal to Lemma 5.1: we define a collector operator $\mathcal{C} : \mathcal{M}_{\mathsf{LHM}_{\vee}} \to \mathcal{M}_{\mathsf{MM}}$ by

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

and verify that $(\mathbf{m}_{\mathsf{LHM}_\vee}, \mathbf{a}_{\mathsf{LHM}_\vee}) \lhd_{\mathcal{C}} (\mathbf{m}_{\mathsf{MM}}, \mathbf{a}_{\mathsf{MM}})$. Indeed, according to Definition 5.3, this amounts to two things: (1) $\mathbf{m}_{\mathsf{MM}} = \mathcal{C} \circ \mathbf{m}_{\mathsf{LHM}_\vee}$, and (2) $\mathbf{a}_{\mathsf{LHM}_\vee} = \mathbf{a}_{\mathsf{MM}} \circ \mathcal{C}$. The latter is immediate from the definitions of the three objects involved. For the first one, observe first that $\mathcal{C}$ is monotone. Next, suppose that $\mathcal{D} \in \mathbf{P}_{\mathrm{DLP}}$. Using the monotonicity of $\mathcal{C}$, and Lemma 7.2 (as $\mathbb{B}$ is totally ordered) we compute:

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

For the other direction,

$$\mathbf{m}_{\mathsf{LHM}_\vee}(\mathcal{D}) \subseteq \{M \mid M \text{ is a model of } \mathcal{D}\},$$

on which we apply the monotone $\mathcal{C}$ on both sides to obtain

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

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

From LPN to DLPN

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

Remember, $\mathcal{V}_{\mathsf{WF}^\kappa} = \mathbb{V}_\kappa$, $\mathcal{M}_{\mathsf{WF}^\kappa}$ is the set of all possible $\mathbb{V}_\kappa$-interpretations, and $\mathbf{m}_{\mathsf{WF}^\kappa}$ maps an LPN program to its least, $\mathbb{V}_\kappa$-valued model. Finally, $\mathbf{a}_{\mathsf{WF}^\kappa}(M)(p)$ is the value of p in M, in other words, $M(p)$.

We denote by $\mathsf{WF}^\kappa_\vee$ the semantics we obtain by using $(-)^\vee$ on WF^κ. This semantics has:

$$\begin{aligned}
\mathcal{V}_{\mathsf{WF}^\kappa_\vee} &= \mathcal{V}_{\mathsf{WF}^\kappa} = \mathbb{V}_\kappa \\
\mathcal{M}_{\mathsf{WF}^\kappa_\vee} &= \mathcal{P}(\mathcal{M}_{\mathsf{WF}^\kappa}).
\end{aligned}$$

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

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

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

Theorem 7C. *The* $\mathsf{WF}^\omega_\vee$ *and the* MM^ω *semantics on finite DLPN programs, are equivalent.*

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

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

and verify that $(\mathbf{m}_{\mathsf{WF}^\omega_\vee}, \mathbf{a}_{\mathsf{WF}^\omega_\vee}) \lhd_{\mathcal{C}} (\mathbf{m}_{\mathsf{MM}^\omega}, \mathbf{a}_{\mathsf{MM}^\omega})$, so that the result will again be a direct consequence of Lemma 5.1. The remaining of the proof is similar to the one of Theorem 7B, except that this time we use the fact that $\mathbb{V}_\omega$ is totally ordered. ∎

This concludes the exposition of our "disjunctive" toolkit. In the next part we turn to games, yet in its last chapter we will reuse the $(-)^\vee$ operator, this time on game semantics.

Part III

Game semantics

Chapter 8

The LPG semantics

In this chapter we introduce the simplest of games that we will use, and which will form the basis for all the remaining games. As an introduction to the subject of game semantics for logic programs, some notions are only described informally. On the next chapter, they will be strictly formalized.

8.1 Introduction to game semantics

Games made their debut in the logic programming scene with [vE86], where we find the first informal description of a game in the logic programming literature. Similar games can also be found as early as [Acz77]. But it was not until [DCLN98], that a game semantics was systematically studied for the case of LP. It was there shown, that it is in fact sound and complete with respect to SLD resolution. This is a rather involved game, which stays close to the procedural semantics, and therefore directly handles first-order programs, taking into account variables, function symbols, substitutions, etc. Approximately a decade later, this game—or, to be fair, its propositional version—was extended in [GRW08], to cover negation for finite, propositional LPN programs. The LPN game semantics is proven to be equivalent to an infinite-valued refinement of the well-founded model semantics (as defined in [RW05]); it is a denotational game semantics. A couple of years after that, two games to deal with DLP and DLPN (again from a denotational point of view) were described informally in [Tso10].

The history of denotational and game semantics for these four versions of logic programming languages is summarized in Table 8.1.

Lang.	**Denotational semantics**	**Game semantics**
LP	least Herbrand model, [vEK76]	LPG, [DCLN98]
DLP	minimal models, [Min82]	DLPG, [Tso13]
LPN	well-founded model, [VGRS91]	LPNG, [GRW08]
DLPN	$\mathbb{V}_\kappa$-valued minimal models, [CPRW07]	(LPNG)$^\vee$, § **11.3**

Table 8.1: Development of game semantics for logic programming.

Why games? There are various benefits of defining a correct game semantics for each of these programming languages. On the operational side, the LP game helps us prune down the space of SLD derivations, by grouping them together using the much smaller space of strategies. In fact, the *alpha-beta algorithm* was used in [LC00] to speed-up the resolution strategies even for the case of *constraint logic programming*. On the denotational side, these games impart elegant characterizations of these main versions of logic programming. As it turns out, starting from LP, one only needs to add a couple of simple game-rules to its game to arrive at DLP; the addition of a different one brings you to LPN. These rules are fairly modular, so that it even makes sense to consider adding all of them simultaneously to deal with DLPN—although we will choose a different route to obtain a game semantics for DLPN in this text. Contrast the simplicity of this approach to the difficulty of treating disjunction and negation relying solely on model-theoretic tools. In addition, this kind of games is also applicable to *intensional logic programming* (as explained in [NR12]), and even outside of the logic programming world, e.g., to *boolean grammars* (see [KNR11]).

8.2 The LPG game

The general picture, shared by all of the games that we consider here, is based on Lorenzen dialogue games (see [Lor61]). There are two players (Player I vs. Player II, or Opponent vs. Player) and two player rôles: *doubter* vs. *believer*.[1] At each round of the game, one player will be the doubter and the other one will be the believer. In the beginning of the game, Player I is the doubter and Player II the believer. When there is no negation present, the players will never change rôles, and so we may also refer to them as Doubter and Believer respectively.[2]

Given a program $\mathcal{P}$ and a goal $\leftarrow \texttt{p}$, the game is about the two players arguing over whether the given goal should succeed 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 from $\mathcal{P}$ that has `p` as its head and play it. For instance, selecting $\texttt{p} \leftarrow \texttt{a}, \texttt{b}, \texttt{c}$, he replies:

Believer: “`p` because `a`, `b`, and `c`.”,

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

Doubter: “Why `b`?”.

[1] We agree to refer to Player I as a *she*, and to Player II as a *he*. There is no particular reason for the specific choice, but keeping their genders separate makes talking about them easier. I first encountered this convention in [Vää11], and I adopted it.

[2] This is not the case for the LPN game: to capture the “unknown” truth value of the well-founded model, we need to allow ties in the game, as well as *rôle-switching* moves. This is explained in Chapter **9**.

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. Doubter has the *benefit of the doubt*, which means that if she can keep doubting forever (in case of an infinite play), she wins. See Section **8.4** for a discussion about the reasoning behind this decision.

This conveys the essence of the games we use for logic programming; all games we define in this text are based on the same principles. We will denote believer moves by β and doubter moves by δ, and refer to the game just described as the LPG game, and denote it by $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow \mathtt{p})$, for a given program $\mathcal{P}$ and a goal clause $\leftarrow \mathtt{p}$. When the game is obvious from the context we may omit the superscript.

8.3 Example plays

▶ *Example 8.1.* Consider the program

$$\mathcal{P} := \left\{ \begin{array}{l} \mathtt{p} \leftarrow \mathtt{q}\,,\mathtt{r} \\ \mathtt{q} \leftarrow \mathtt{s} \\ \mathtt{s} \leftarrow \\ \mathtt{r} \leftarrow \mathtt{t} \\ \mathtt{r} \leftarrow \end{array} \right\}.$$

With the goal $\leftarrow \mathtt{p}$, three maximal plays in $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow \mathtt{p})$ might be the following:

$$\pi_1 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{p}} \\ \quad \mathsf{D}_0 : \mathtt{p} \\ \quad \mathsf{B}_0 : \mathtt{p} \leftarrow \underline{\mathtt{q}}\,,\mathtt{r} \\ \quad \mathsf{D}_1 : \mathtt{q} \\ \quad \mathsf{B}_1 : \mathtt{q} \leftarrow \underline{\mathtt{s}} \\ \quad \mathsf{D}_2 : \mathtt{s} \\ \quad \mathsf{B}_2 : \mathtt{s} \leftarrow \end{array} \right| ,\ \pi_2 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{p}} \\ \quad \mathsf{D}_0 : \mathtt{p} \\ \quad \mathsf{B}_0 : \mathtt{p} \leftarrow \mathtt{q}\,,\underline{\mathtt{r}} \\ \quad \mathsf{D}_1 : \mathtt{r} \\ \quad \mathsf{B}_1 : \mathtt{r} \leftarrow \underline{\mathtt{t}} \\ \quad \mathsf{D}_2 : \mathtt{t} \end{array} \right| ,\ \pi_3 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{p}} \\ \quad \mathsf{D}_0 : \mathtt{p} \\ \quad \mathsf{B}_0 : \mathtt{p} \leftarrow \mathtt{q}\,,\underline{\mathtt{r}} \\ \quad \mathsf{D}_1 : \mathtt{r} \\ \quad \mathsf{B}_1 : \mathtt{r} \leftarrow \end{array} \right| .$$

Conveniently enough, if we underline Doubter's selections—as we have done in the example above—we can condense each play by entirely omitting the lines that correspond to her moves. For the sake of laziness, we will follow this practice from now on. The three plays above are therefore written as:

$$\pi_1 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{p}} \\ \quad \mathsf{B}_0 : \mathtt{p} \leftarrow \underline{\mathtt{q}}\,,\mathtt{r} \\ \quad \mathsf{B}_1 : \mathtt{q} \leftarrow \underline{\mathtt{s}} \\ \quad \mathsf{B}_2 : \mathtt{s} \leftarrow \end{array} \right| ,\ \pi_2 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{p}} \\ \quad \mathsf{B}_0 : \mathtt{p} \leftarrow \mathtt{q}\,,\underline{\mathtt{r}} \\ \quad \mathsf{B}_1 : \mathtt{r} \leftarrow \underline{\mathtt{t}} \end{array} \right| ,\ \pi_3 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{p}} \\ \quad \mathsf{B}_0 : \mathtt{p} \leftarrow \mathtt{q}\,,\underline{\mathtt{r}} \\ \quad \mathsf{B}_1 : \mathtt{r} \leftarrow \end{array} \right| .$$

Plays π_1 and π_3 are won by Believer, who has managed to corner the doubter by playing a fact: there are no conjuncts for her to choose from. On the other hand, π_2 is won by Doubter, who managed to doubt an atom which is not the head of any of the rules of $\mathcal{P}$, and thus Believer cannot make any move and loses the game. ◀

8.4 Benefit of the doubt

Why do we give the benefit of the doubt to the doubter? The following example illustrates how things can go wrong, in case we do not.

▶ *Example 8.2.* Consider the following program

$$\mathcal{P} := \left\{ \begin{array}{l} \mathtt{a} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b} \\ \mathtt{b} \leftarrow \end{array} \right\}$$

and the goal $\leftarrow \mathtt{a}$. A play in $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow \mathtt{a})$ has to begin with Doubter doubting a, to which Believer responds with $\mathtt{a} \leftarrow \mathtt{a}\,\mathtt{,}\,\mathtt{b}$. For as long as Doubter doubts 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 b, at which point Believer would win by playing the fact $\mathtt{b} \leftarrow$. This describes a winning strategy for Believer, but—this being a logic program—we most definitely do not want the goal $\leftarrow \mathtt{a}$ to succeed, because of the denotational semantics:

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

Who has the benefit of the doubt is a decision similar to the closed/open world assumptions (CWA/OWA) in knowledge representation languages. Giving it to the doubter resembles CWA, while giving it to no player resembles OWA;[3] 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 [Rei78] and [Min82].

8.5 Semantics from games

Remember—our objective is to use games to provide denotational semantics for logic programs; in other words, to decide which goals should succeed, i.e., which answers are correct.

Notice now, that the three plays π_1, π_2, and π_3 of Example 8.1 above are played on the same game, of the same program $\mathcal{P}$, with the same goal $\leftarrow \mathtt{p}$. Yet Believer wins two of them, and loses another. This shows that *merely winning or losing a particular play in a game, is not enough information to decide if a goal should succeed or not*: maybe we managed to win a play because our opponent played badly enough, or we lost the play because we didn't play smartly enough. This is exactly what happened in π_2, where Believer chose the rule $\mathtt{r} \leftarrow \mathtt{t}$ to justify his belief on r, thus enabling the doubter to doubt t for which there is no rule that supports it. In π_3 he makes the right move and chooses to play the *fact* $\mathtt{r} \leftarrow$, which immediately grants him victory.

It therefore makes no sense to determine the success of a goal by looking at particular plays. Instead, we focus on *strategies*. Informally, a strategy for a game determines what Player will play in each possible position of the game.[4] If following a strategy σ Player is guaranteed to win against any possible move of Opponent, we call σ a *winning strategy* for that goal.

This allows us to the define the LPG semantics:

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

[4] Formal definitions are given on the next chapter.

Definition 8.1 (LPG semantics)**.** Let $\mathcal{P}$ be an LP program. A goal $\leftarrow \mathtt{p}$ *succeeds*, if Believer has a winning strategy in $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow \mathtt{p})$.

The LPG semantics under the abstract semantic framework

- $\mathcal{V}_{\mathsf{LPG}} \triangleq \mathbb{B}$.
- $\mathcal{M}_{\mathsf{LPG}}$ is the set of strategies based on LP programs.
- $\mathbf{m}_{\mathsf{LPG}}$ maps every LP program $\mathcal{P}$ to the set of strategies for the LPG game based on $\mathcal{P}$.
- $\mathbf{a}_{\mathsf{LPG}}(\Sigma)(q) \triangleq \begin{cases} \mathsf{T}, & \text{if there is a winning strategy } \sigma \in \Sigma \text{ for } q \\ \mathsf{F}, & \text{otherwise.} \end{cases}$

8.6 Soundness and completeness

We state in this section some known results about the LP game.

Theorem 8A (Di Cosmo–Loddo–Nicolet)**.** *The LPG semantics is sound and complete with respect to SLD resolution.*

Theorem 8B (Clark)**.** *SLD resolution is sound and complete with respect to the least Herbrand model semantics.*

The first of these is proven in [DCLN98], while the second one is due to [Cla79] (and can also be found in [Llo87, Theorems 7.1 and 8.6]). Putting the above two theorems together, we arrive at the correctness of the LPG semantics:

Corollary 8.1 (Soundness and completeness of the LP game semantics)**.** *Let $\mathcal{P}$ be an LP program, and $\leftarrow \mathtt{p}$ a goal. Then, there is a winning strategy in the associated game $\Gamma^{LP}_{\mathcal{P}}(\leftarrow \mathtt{p})$ iff $\mathtt{p}$ belongs to the least Herbrand model of $\mathcal{P}$.*

Having understood the basic ideas behind the LPG game, we proceed to formally define and study the LPNG game, which will turn out to reduce to the LPG one when no negations are present.

Chapter 9

The LPNG semantics

As we have mentioned previously, to deal with negation-as-failure in the way of the well-founded semantics, it is essential to have some unknown truth value **U** in the truth value space. In the world of games, we introduce *ties* between the two players. The LPNG game is formally defined for *finite* LPN programs, and thus all programs in this chapter are assumed to be finite. This also has the implication that we can limit ourselves to values in the $\mathbb{V}_\omega$ space (see Remark 5.2).

9.1 The LPNG game

The LPNG game is based on the LPG one, and indeed it reduces to it for negation-free programs. Whereas in the LPG game, the rôles of the players stay the same throughout the course of a play, in LPNG, a believer's move can sometimes be *rôle-switching.* For this reason, we cannot refer to the players as "Doubter" and "Believer", so we speak of "Opponent" and "Player" instead.

The LPNG game is played just like the LPG one, until the moment that the doubter doubts a negated conjunct, say $\sim$p. Essentially we read this as:

Opponent: "Why $\sim$p?",

to which Player (who is the believer) is forced to confirm that he indeed doubts p, which turns him into the doubter, and his opponent into the believer:

Player: "Because I doubt p. Why p?".

and the game continues, but with the rôles of the players swapped.

It is important to stress that *the doubter retains the benefit of the doubt*, but there is a game-changing difference: it is no longer true that every infinite play will be won by one of the players. In case a player has managed to "secure" their rôle as a doubter from some certain point on, then indeed they will win any infinite play. Otherwise, there must be an infinite number of rôle-switching moves, and *the play results in a tie.*

9.2 Example plays

▶ *Example 9.1.* Consider the program

$$\mathcal{P} := \left\{ \begin{array}{l} \mathtt{p} \leftarrow \\ \mathtt{q} \leftarrow {\sim}\mathtt{p} \\ \mathtt{r} \leftarrow {\sim}\mathtt{q} \end{array} \right\},$$

We show two plays for this program; the first is played on $\Gamma^{\mathrm{LPN}}_{\mathcal{P}}(\leftarrow \mathtt{q})$, the second on $\Gamma^{\mathrm{LPN}}_{\mathcal{P}}(\leftarrow \mathtt{r})$. We mark every rôle-switch by drawing a straight line and we keep the convention of omitting the lines that correspond to doubter moves (by simply underlying the doubts)

$$\pi_1 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{q}} \\ \quad \mathsf{P}_0 : \mathtt{q} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{p}}}} \\ \hline \quad \mathsf{O}_2 : \mathtt{p} \leftarrow \end{array} \right|, \qquad \pi_2 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{r}} \\ \quad \mathsf{P}_0 : \mathtt{r} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{q}}}} \\ \hline \quad \mathsf{O}_2 : \mathtt{q} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{p}}}} \\ \hline \quad \mathsf{P}_3 : \mathtt{p} \leftarrow \end{array} \right|.$$

Both plays are won by believers, but in π_1 the believer is Opponent, while in π_2, where there are two rôle-switching moves, Player is the believer. ◀

▶ *Example 9.2.* Let us now see an infamous program of LPN:

$$\mathcal{Q} := \left\{ \begin{array}{l} \mathtt{p} \leftarrow {\sim}\mathtt{q} \\ \mathtt{q} \leftarrow {\sim}\mathtt{p} \end{array} \right\}.$$

Consider the infinite play

$$\pi_3 := \left| \begin{array}{l} \mathsf{goal} : \leftarrow \underline{\mathtt{p}} \\ \quad \mathsf{P}_0 : \mathtt{p} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{q}}}} \\ \hline \quad \mathsf{O}_2 : \mathtt{q} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{p}}}} \\ \hline \quad \mathsf{P}_3 : \mathtt{p} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{q}}}} \\ \hline \quad \mathsf{O}_5 : \mathtt{q} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{p}}}} \\ \hline \quad \mathsf{P}_6 : \mathtt{p} \leftarrow \underline{\underline{{\sim}\underline{\mathtt{q}}}} \\ \hline \qquad \vdots \end{array} \right|.$$

Here no player is able to secure the believer rôle for themselves, so this play is won by neither of them: the outcome is a tie. ◀

9.3 Game semantics

To obtain the refined, infinite-valued model $\mathrm{WFM}_\omega(\mathcal{P})$, we need to take into account the rôle-switching moves played. For this, we will use a payoff function:

Definition 9.1 (Payoff)**.** Let π be a play in some game $\Gamma^{\mathrm{LPN}}_{\mathcal{P}}(\leftarrow p)$. Then the *payoff* functions Φ_ω and Φ are defined by:

$$\Phi_\omega(\pi) \triangleq \begin{cases} \mathsf{T}_n, & \text{if Player wins in } \pi, \\ \mathsf{F}_n, & \text{if Player loses in } \pi, \\ \mathsf{U}, & \text{otherwise,} \end{cases}$$

where n is the number of rôle-switching moves played in π; and

$$\Phi \triangleq \mathit{collapse} \circ \Phi_\omega,$$

where *collapse* is the "subscript-removing" function of Definition 2.6.

▶ *Example 9.3.* Consider the plays π_1, π_2, and π_3 from the examples 9.1 and 9.2 above. Their corresponding payoffs are:

$$\Phi_\omega(\pi_1) = \mathsf{F}_1, \qquad \Phi_\omega(\pi_2) = \mathsf{T}_2, \qquad \Phi_\omega(\pi_3) = \mathsf{U}.$$ ◀

Definition 9.2 (Non-losing strategy)**.** A strategy σ is *non-losing* iff

$$\bigwedge \{\Phi_\omega(\pi) \mid \pi \in \sigma\} \geq \mathsf{U}.$$

𝄾 Remark 9.1. At this point, we are really in possession of two different games: one in which there are only three possible outcomes for each play (win, lose, or tie); and one in which by consulting the payoff function Φ_ω, we actually have various degrees of winning and losing, and a single level of tie. Notice, however, that the two games only differ in the possible outcomes; they share the same moves, plays, and strategies.

We now proceed to directly define the LPNG and LPNG$^\omega$ semantics under the framework of Chapter **5**:

The LPNG semantics

- $\mathcal{V}_{\mathsf{LPNG}} \triangleq \mathbb{V}_1$.
- $\mathcal{M}_{\mathsf{LPNG}}$ is the set of strategies based on LPN programs.
- $\mathbf{m}_{\mathsf{LPNG}}$ maps every LPN program $\mathcal{P}$ to the set of strategies for the LPNG game based on $\mathcal{P}$.
- Finally,

$$\begin{aligned} \mathbf{a}_{\mathsf{LPNG}}(\Sigma)(q) &\triangleq \begin{cases} \mathsf{T}, & \text{if there is a winning strategy in } \Sigma \text{ for } q \\ \mathsf{U}, & \text{else, if there is a non-losing strategy in } \Sigma \text{ for } q \\ \mathsf{F}, & \text{otherwise.} \end{cases} \\ &= \bigvee \left\{ \bigwedge \{\Phi(\pi) \mid \pi \in \sigma\} \;\middle|\; \sigma \text{ is a strategy in } \Sigma \text{ for } q \right\} \end{aligned}$$

The LPNG^ω semantics

- $\mathcal{V}_{\mathsf{LPNG}^\omega} \triangleq \mathbb{V}_\omega$.
- $\mathcal{M}_{\mathsf{LPNG}^\omega}$ is the set of strategies based on LPN programs.
- $\mathbf{m}_{\mathsf{LPNG}^\omega}$ maps every LPN program $\mathcal{P}$ to the set of strategies for the LPNG game based on $\mathcal{P}$.
- $\mathbf{a}_{\mathsf{LPNG}^\omega}(\Sigma)(q) \triangleq \bigvee \{\bigwedge \{\Phi_\omega(\pi) \mid \pi \in \sigma\} \mid \sigma \text{ is a strategy in } \Sigma \text{ for } q\}$

9.4 Soundness and completeness

Soundness and completeness results for both games with respect to the corresponding well-founded semantics are proven in [GRW08]. The following theorem summarizes these results:

Theorem 9A (Soundness and completeness of the LPNG semantics)**.** *Let $\mathcal{P}$ be an LPN program, and let Σ be the set of all strategies of* LPNG *games played on $\mathcal{P}$. Then,*

$$\mathrm{WFM}(\mathcal{P}) = \mathbf{a}_{\mathsf{LPNG}}(\Sigma),$$
$$\mathrm{WFM}_\omega(\mathcal{P}) = \mathbf{a}_{\mathsf{LPNG}^\omega}(\Sigma).$$

We have thus presented a game semantics for LPN programs, which is sound and complete with respect to the well-founded semantics. Next we turn our attention to the other extention of LP programs: DLP.

Chapter 10

The DLPG semantics

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

Soundness and completeness of the DLPG semantics (Theorem 10G)**.** *The game semantics* DLPG *of DLP is equivalent to the minimal model semantics, i.e., given any DLP program $\mathcal{P}$, and any disjunction D,*

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

Two key ideas of Chapter **6** are further developed to prove the two directions of the above result: *combination* (for completeness) and *splitting* (for both). In short, we begin with a finite disjunctive logic program $\mathcal{P}$, and split it in two new DLP programs $\mathcal{P}_1$ and $\mathcal{P}_2$, such that they are in a sense, "less disjunctive". Now, strategies for games in $\mathcal{P}$ can themselves be split to strategies for games in $\mathcal{P}_1$ and $\mathcal{P}_2$, and vice versa: strategies for such games can be combined to form new strategies for games in $\mathcal{P}$. By repeated splitting, we eventually arrive at programs that are not disjunctive at all (LP programs), which we know how to deal with since [DCLN98]. Finally, compactness will allow us to extend this result to the general case of infinite DLP programs.

Be aware, that even though the DLPG game developed here can be thought of as another extension of the LPG game, its formalization is drastically different from those of the games of either LP or LPN, and appears to be novel in the field of logic programming. It has been influenced instead by game semantics in the style of Abramsky–Jagadeesan–Malacaria and Hyland–Ong–Nickau, used for PCF and functional programming in general (see [AJM00], [AM99], [HO00] and [Nic94]). Although we assume no such prior knowledge of this field, the initiated reader should hopefully feel at home. Let us play.

10.1 The simplified DLP game

This game extends the LPG game to deal with disjunctions. First of all, the goal here is assumed to consist of a *disjunction* of atoms. The key difference from the LPG (and LPNG) games is that *in the* DLPG *game the believer can play combo moves: he can use more than one rule to support his belief.* What is more, he can use *not only* rules from the given program, but also *implicit rules*, i.e., rules of the form $\mathtt{a} \leftarrow \mathtt{a}$. Thanks to rule combination, he can disjunctively combine his selection of rules into one, say $\mathtt{G} \leftarrow \mathtt{D}_1, \cdots, \mathtt{D}_n$, and play it against his opponent:

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

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

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

And the game goes on.

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

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

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

▶ *Example 10.1.* Consider the DLP program

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

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

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

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

Note that in both plays of Example 10.1, 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 10.2.* The program now is

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

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

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

won by Believer, and

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

by Doubter. ◄

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

10.2 The DLPG 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 β that he combined to create his move. Likewise, Doubter will not select a single disjunction from the combined move of Believer; instead she will select a sequence of occurrences: *one from each body* of the rules in β—which, we stress, is now a *sequence* of rules. Things become clearer and formal after the definitions and the examples that follow.

Given a DLP program $\mathcal{P}$ and a goal clause $\leftarrow \mathtt{G}$, we will define the associated DLPG game, and denote it by $\Gamma^{\mathrm{DLP}}_{\mathcal{P}}(\leftarrow \mathtt{G})$ or simply by $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ when no confusion may arise.

Just like every game that we investigate in this thesis, this is a two-player game, with the two same player rôles: the *doubter* and the *believer*. As was the case for the LPG game, the rôles of the players never change throughout the game and so we will simply call the players Doubter and Believer again.

Doubter starts by doubting $\mathtt{G}$, the body of the goal clause, and Believer tries to defend it. A player who cannot play a valid move loses the game. Now let us be precise.

Definition 10.1 (Extended program)**.** Given any set of DLP rules $\mathcal{P}$, we can extend it to $\mathcal{P}_+$, which includes all meaningful implicit rules. In detail,

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

Definition 10.2 (Moves)**.** A *doubter move* δ is a sequence of occurrences of disjunctions in bodies of DLP clauses; we refer to these occurrences as the *doubts* of δ. A *believer move* β from $\mathcal{P}$ is a finite sequence of DLP rules from the extended set $\mathcal{P}_+$. Given a move m, we call $[\,m\,]$ the *statement* of the move and refer to the sequence m as the *justification* of the statement.[1] We say that the elements of a believer move β that belong to $\mathcal{P}$ constitute the *proper part* of the justification; the rest, the *implicit*. If $|\beta| > 1$ we call β a *combo move*.

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

Table 10.1: How to read believer and doubter statements.

Table 10.1 suggests how we can read moves aloud. Notice that they resemble an actual dialogue. This motivates the following definition:

Definition 10.3 (Dialogue)**.** A *quasidialogue* from $\mathcal{P}$ is a finite sequence

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

such that:

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

It is a *dialogue* if it also satisfies:

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

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

𝄾 Remark 10.1. Regarding the first restriction imposed on believer moves, one could make the seemingly stronger demand that $\mathsf{head}([\,\beta_i\,]) = [\,\delta_i\,]$. It is easy to check, however, that this changes nothing in the game, since the believer can bring up implicit rules to combine them with his move in case the subset was proper. To see that he can still win exactly the same arguments, remember that by definition, if there is at least one fact in a combination, the resulting rule is also a fact.

Definition 10.4 (Play)**.** A *(quasi)play* of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ is a (quasi)dialogue π from $\mathcal{P}$ which, if non-empty, satisfies the additional property that $[\,\delta_0\,] = \mathtt{G}$. We denote the empty play by $\varepsilon \triangleq \langle\,\rangle$.

Note that dialogues (and plays) are sequences and thus inherit the partial orderings from $\sqsubseteq$ and $\sqsubseteq_{\mathrm{e}}$.

► *Example 10.3.* Consider the same program $\mathcal{Q}$ as in Example 10.2, repeated here for convenience:

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

[1] $[\,-\,]$ was introduced in Definition 6.8.

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

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

is (still) won by Believer, and

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

by Doubter. ◀

Property 10.1. *Any dialogue* $\pi \coloneqq \langle \delta_0, \beta_0, \dots \rangle$ *from* $\mathcal{P}$ *can be considered as a play of* $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$*, where* $\mathtt{G} \coloneqq [\,\delta_0\,]$*.*

☡ Remark 10.2 (Disallowing stalling). We have forced the believer to always include at least one rule from the actual program $\mathcal{P}$, thus banning what we are about to call “stalling” from our games. Still, this concept will be crucial for our exposition: the introduction of plays in which stalling is allowed (i.e., quasiplays) will make a lot of the statements that follow easier to prove.

Definition 10.5 (Follow)**.** We say that Doubter *follows*, if she has only one possible valid move that she can play in response to a believer move β, i.e., if for every rule $\psi \in \beta$, $|\mathsf{body}(\psi)| = 1$. We denote such a follow move by $\overline{\beta}$. In symbols,

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

Definition 10.6 (Stalling)**.** We say that a Believer is *stalling*, if he responds to a doubter move δ by playing the sequence of implicit rules

$$\overline{\delta} \triangleq \langle \mathtt{a} \leftarrow \mathtt{a} \mid a \in [\,\delta\,]\rangle\,.$$

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

☡ Remark 10.3 (Notation). Once more we have overloaded a symbol here, but once more the end justifies the means: we increase readability with no possibility of ambiguity, since the follow-bar can only be applied to believer moves, while the stalling-bar only covers doubter moves. This also brings the handy double-bar notation for the only possible reply to a stalling move, in which case the follow is always defined. In addition, it is easy to verify the following cute, bar-cancellation properties:

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

⁊ Remark 10.4. For any doubter move δ, the move $\overline{\overline{\delta}}$ is equal to δ modulo a change in occurrences: it contains exactly the same doubts. Therefore, it also shares the same statement

$$[\,\delta\,] = \left[\,\overline{\overline{\delta}}\,\right].$$

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

Definition 10.7 ($\mathsf{rmstall}$)**.** Let τ be a quasidialogue from $\mathcal{P}$. We define the function $\mathsf{rmstall}$ recursively by cases on $\ell := |\tau|$:
Case 1: $\ell \leq 1$. Then either $\tau = \langle\,\rangle$ or $\tau = \langle\delta\rangle$ for some doubter move δ. Both alternatives contain no believer moves, and hence zero stallings; we set

$$\mathsf{rmstall}(\tau) \triangleq \tau.$$

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

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

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

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

Property 10.2 (Validity of $\mathsf{rmstall}$)**.** *If τ is a quasidialogue from a program, then* $\mathsf{rmstall}(\tau)$ *is a dialogue from the same program; and if π is a quasiplay in some game, then* $\mathsf{rmstall}(\pi)$ *is a play in the same game. In addition,* $\mathsf{rmstall}$ *leaves dialogues and plays intact:*

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

Property 10.3 ($\mathsf{rmstall}$ is $\sqsubseteq$-monotone)**.**

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

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

Definition 10.8 (Strategy)**.** A *strategy* σ in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ is a set of plays in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, such that:

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

(ii) every play in σ has even length;

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

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

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

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

⁊ Remark 10.5 (Strategies as posets). A strategy σ can be seen as a *poset* $(\sigma, \sqsubseteq_{\mathrm{e}})$ of even-length plays with a bottom element $\bot_\sigma = \varepsilon$.

Definition 10.9 (Total strategy)**.** A strategy σ is called *total*, if for every play $\pi \in \sigma$ and every doubter response to it δ (i.e., a δ such that $\pi \mathbin{+\!\!+} \delta$ is a play), there is *at least one* move β such that $\pi \mathbin{+\!\!+} \langle\delta, \beta\rangle$ is also in σ.

Siding with Believer, we give the following definition:

Definition 10.10 (Winning strategy)**.** A strategy σ is called *winning*, if it is total and finite.

⁊ Remark 10.6 (Infinite plays and limits). Let us pretend awhile that dialogues (and plays) might be infinite. We should then change (among other things) the definition of strategy to demand that it is also closed under limits—or, should we? If we do so, nothing essential will change, as a correspondence between such strategies and the ones we are using here is obvious:

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

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

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

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

Definition 10.11 (answer_σ)**.** Given a play π and a strategy σ in some game $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, the play $\mathsf{answer}_\sigma(\pi) \in \sigma$ will be:

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

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

Property 10.4. *The function* answer_σ *behaves like a closure operator; i.e., whenever it is defined throughout the following statements, it satisfies them:*

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

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

Proposition 10.5. *Let* $\mathscr{D}$, $\mathscr{E}$, *and* $\mathscr{F}$ *be L.P. conjunctions such that* $\mathscr{F} \equiv \mathscr{D} \vee \mathscr{E}$. *Then for every disjunction* $F \in \mathscr{F}$ *there is some disjunction* $D_F \in \mathscr{D}$ *and some disjunction* $E_F \in \mathscr{E}$ *such that* $D_F \cup E_F \subseteq F$.[2]

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

We have exposed every little detail that we will need in order to develop the DLPG semantics in a formal mathematical setting. Let us do so: in the next two sections, we define the combination and splitting of both plays and strategies. The main point is that these constructions preserve all the properties that we need.

10.3 Plays: combining, restricting, and splitting

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.

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

[2] L.P. disjunctions and conjunctions were introduced in Definition 3.1.

and $\phi|_{H_2}$ respectively. The variable q ranges over the size of the partition: $q = 1, 2$.[3] We use π and π_q for dialogues or plays of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$ respectively; τ and τ_q for "quasi-". We remind the reader that we solely use β and δ for believer and doubter moves respectively, and that strategies are usually denoted by σ. In this setting, with this notation, we proceed to define combination, restriction, and splitting of both plays and strategies.

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

$$\text{Forbidden}_q(\beta) \overset{\triangle}{\iff} \phi_q \in \beta.$$

Definition 10.13 (Release)**.** Given a believer move β_i^q of $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$, we define β_i^{q*} to be β_i^q after replacing instances of the forbidden rule ϕ_q by ϕ, so that it becomes valid in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. We call β_i^{q*} the *release* of β_i^q from H_q to H.

Property 10.6. *The release of a move satisfies the following properties:*

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

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

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

Proposition 10.7. *Let β_1 and β_2 be two believer moves from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and let $\beta := {\beta_1}^* +\!\!\!+ {\beta_2}^*$. Then*

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

Proof. We compute

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

where the starred equality holds since

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

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

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

Combining synchronous plays

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

Definition 10.14 (Synchronous)**.** Two quasiplays τ_1 and τ_2 in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively are called *synchronous* (or *in sync with each other*) if both Believers use their forbidden rules in the exact same turns. In symbols,

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

Property 10.8. *If τ_1 is in sync with τ_2, then so is any prefix of τ_1 with any prefix of τ_2.*

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

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

be the two given plays. We set

$$\pi_1 \ddot{\curlyvee} \pi_2 \triangleq \langle \delta_0, \beta_0, \delta_1, \beta_1, \dots \rangle,$$

where the symbols are defined as follows.

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

$$\delta_0 \coloneqq \delta_0^1 \mathbin{+\!\!+} \delta_0^2;$$

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

$$\beta_0 \coloneqq \beta_0^1 \mathbin{+\!\!+} \beta_0^2 = {\beta_0^1}^* \mathbin{+\!\!+} {\beta_0^2}^*,$$

a valid move since $\mathsf{head}([\beta_0]) \subseteq [\delta_0]$ (Property 10.6, Proposition 10.7). Now Doubter must select one occurrence from each rule-body in β_0, and the moves δ_1^1 and δ_1^2 provide just that:

$$\delta_1 \coloneqq \delta_1^1 \mathbin{+\!\!+} \delta_1^2.$$

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

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

which is valid in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. Doubter's turn: δ^1_{i+1} and δ^2_{i+1} consist of occurrences from the rule-bodies of β^1_i and β^2_i respectively. Since releasing only affects heads (Property 10.6(ii)), all occurrences in the rule-bodies of β^q_i are occurrences in the rule-bodies of β^{q*}_i as well. Therefore, she can copy the selections of δ^1_{i+1} and δ^2_{i+1} by playing

$$\delta_{i+1} := \delta^1_{i+1} \mathbin{+\!\!+} \delta^2_{i+1}.$$

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

𝄾 Remark 10.7. We have cheated a bit, since we took for granted that the doubter of $\pi_1 \ddot{\curlyvee} \pi_2$ has in her possession *two* doubter moves δ^1_k and δ^2_k from π_1 and π_2 respectively. This will not hold if the plays have unequal lengths; in this case, the combined play ends as soon as the shortest play ends.

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

Definition 10.15 (Synchronous combination)**.** Given two synchronous quasiplays

$$\begin{aligned} \tau_1 &:= \langle \delta^1_0, \beta^1_0, \delta^1_1, \beta^1_1, \dots \rangle \quad \text{of } \Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G}), \\ \tau_2 &:= \langle \delta^2_0, \beta^2_0, \delta^2_1, \beta^2_1, \dots \rangle \quad \text{of } \Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G}), \end{aligned}$$

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

$$\tau_1 \ddot{\curlyvee} \tau_2 \triangleq \langle \delta_0, \beta_0, \delta_1, \beta_1, \dots \rangle$$

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

$$\begin{aligned} \delta_i &:= \delta^1_i \mathbin{+\!\!+} \delta^2_i, \\ \beta_i &:= \beta^{1*}_i \mathbin{+\!\!+} \beta^{2*}_i. \end{aligned}$$

Proposition 10.9 (Validity of $\ddot{\curlyvee}$)**.** *Given two synchronous quasiplays τ_1 and τ_2 of $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively, $\tau := \tau_1 \ddot{\curlyvee} \tau_2$ is a quasiplay in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. It follows that if τ_1 and τ_2 do not stall simultaneously, τ will be a play of the game $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.*

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

$$\begin{aligned} \mathsf{head}\big([\,\beta^1_i\,]\big) &\subseteq [\,\delta^1_i\,], \\ \mathsf{head}\big([\,\beta^2_i\,]\big) &\subseteq [\,\delta^2_i\,], \end{aligned}$$

and so by taking unions on both sides and by using Proposition 10.7 we obtain

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

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. 70). To verify that τ is indeed a quasiplay, observe that both τ_1 and τ_2 share the same goal with τ, and so $\delta_0 = \delta^1_0 \mathbin{+\!\!+} \delta^2_0$ is a correct first move for a quasidialogue from $\mathcal{P}$ to be a quasiplay of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.

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

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

Corollary 10.10 (Preservation of plays)**.** *The synchronous combination of two synchronous plays is a play.*

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

Property 10.11 (Preservation of parity)**.** *Let τ_1 and τ_2 be two synchronous quasiplays, both of even length. Then $\tau_1 \ddot{\curlyvee} \tau_2$ will also have even length.*

Property 10.12 ($\ddot{\curlyvee}$ is monotone)**.** *Let τ_1 and τ_2 be two synchronous quasiplays. Then for any $\tau_1' \sqsubseteq \tau_1$ and any $\tau_2' \sqsubseteq \tau_2$ we have $\tau_1' \ddot{\curlyvee} \tau_2' \sqsubseteq \tau_1 \ddot{\curlyvee} \tau_2$.*

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

Combining arbitrary plays

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

Definition 10.16 (Synchronization)**.** Let τ_1 and τ_2 be two quasidialogues from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and let ℓ_1 and ℓ_2 be their lengths. We recursively define their *synchronization* $\mathsf{sync}(\tau_1, \tau_2)$ by cases depending on $\ell := \min\{\ell_1, \ell_2\}$.

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

$$\mathsf{sync}(\tau_1, \tau_2) \triangleq (\tau_1, \tau_2)\,.$$

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

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

where rec_1 and rec_2 wrap the recursive calls

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

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

𝄾 Remark 10.8 (Dependencies). Even though we have not incorporated ϕ_1 and ϕ_2 into the symbol sync of synchronization, we stress that it does, in fact, depend on both of those rules: it uses them to determine the (a)–(c). Since we have fixed ϕ, ϕ_1, and ϕ_2 into our notation, however, we allow ourselves to simply use sync instead of an excessively precise name like $\mathsf{sync}_{\phi_1,\phi_2}$. The same is true for various symbols that we use, such as $\curlyvee$ and *.

Proposition 10.13. *Synchronization is a well-defined, total operation.*

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

One can easily verify that synchronization behaves as expected:

Property 10.14 (Validity of synchronization)**.** *Let* $(\dot{\tau_1}, \dot{\tau_2}) := \mathsf{sync}(\tau_1, \tau_2)$. *Then*

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

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

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

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

Proposition 10.15 (Preservation of parity)**.** *If τ_1 and τ_2 have even lengths, then so do the elements of* $\mathsf{sync}(\tau_1, \tau_2)$*.*

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

Proposition 10.16 (sync is monotone)**.** *Let τ_1' and τ_2' be two quasidialogues, and let $\tau_1 \sqsubseteq \tau_1'$ and $\tau_2 \sqsubseteq \tau_2'$. Then* $\mathsf{sync}(\tau_1, \tau_2) \sqsubseteq \mathsf{sync}(\tau_1', \tau_2')$*. Moreover, if $\tau_1 \sqsubset \tau_1'$ and $\tau_2 \sqsubset \tau_2'$, then* $\mathsf{sync}(\tau_1, \tau_2) \sqsubset \mathsf{sync}(\tau_1', \tau_2')$*.*

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

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

Definition 10.17 (Combination of quasidialogues)**.** Let τ_1 and τ_2 be two quasidialogues from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. We define their *combination* $\tau_1 \curlyvee \tau_2$ by composition:

$$\curlyvee \triangleq \ddot{\curlyvee} \circ \mathsf{sync}.$$

In other words,

$$\tau_1 \curlyvee \tau_2 \triangleq \dot{\tau}_1 \mathbin{\ddot{\curlyvee}} \dot{\tau}_2,$$

where $(\dot{\tau}_1, \dot{\tau}_2) \coloneqq \mathsf{sync}(\tau_1, \tau_2)$.

𝄾 Remark 10.9. By Property 10.14(ii), $\curlyvee$ is compatible with $\ddot{\curlyvee}$, in the sense that it yields the same output in case its input is synchronous.

𝄾 Remark 10.10 (End of $\tau_1 \curlyvee \tau_2$). According to the definitions of synchronization and syncronous combination, the combined quasidialogue $\tau_1 \curlyvee \tau_2$ ends exactly when we reach the final move of either τ_1 or τ_2 (whichever comes first).

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

Corollary 10.17 (Validity of combination)**.** *Given two (quasi)dialogues τ_1 and τ_2 from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, their disjunctive combination $\tau_1 \curlyvee \tau_2$ is a (quasi)dialogue from $\mathcal{P}$.*

Corollary 10.18 (Preservation of plays)**.** *If τ_1 and τ_2 are two (quasi)plays in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively, then $\tau_1 \curlyvee \tau_2$ is a (quasi)play in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.*

Corollary 10.19 (Preservation of parity)**.** *If π_1 and π_2 have even lengths, then so does $\pi_1 \curlyvee \pi_2$.*

Corollary 10.20 ($\curlyvee$ is monotone)**.** *Let π_1' and π_2' be two dialogues, and let $\pi_1 \sqsubseteq \pi_1'$ and $\pi_2 \sqsubseteq \pi_2'$. Then $\pi_1 \curlyvee \pi_2 \sqsubseteq \pi_1' \curlyvee \pi_2'$. Moreover, if $\pi_1 \sqsubset \pi_1'$ and $\pi_2 \sqsubset \pi_2'$, then $\pi_1 \curlyvee \pi_2 \sqsubset \pi_1' \curlyvee \pi_2'$.*

𝄾 Remark 10.11 (Losing is not preserved). Even though Doubter may be victorious in two plays, in the combined version she might not be so. As a counterexample, consider the following program and splitting, in which Doubter wins both π_1 and π_2 (i.e., Believer cannot play a valid response to either plays) and yet she is going to lose in the combined play:

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

and the plays

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

Notice that Believer cannot move in neither of the starting plays, but he can certainly move in their combination by playing the rule $\mathtt{a} \vee \mathtt{b} \leftarrow$. However, as we will later see, such misbehaviors are evaded if the plays π_q come from a strategy splitting; the impatient can read Corollary 10.28 (p. 81).

𝄾 REMARK 10.12 (Combining goals). When combining plays, their common goal $\leftarrow \mathtt{G}$ does not really have to be all that common. We can combine a play in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G}_1)$ with a play in $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G}_2)$ to get a new play in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, where $\mathtt{G} := \mathtt{G}_1 \vee \mathtt{G}_2$. To do so, we first extract plays in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ by altering only the first move in each play so that it is restricted to $\mathtt{G}_q$; then we combine. But we will not need to do such a thing in this work.

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

Restricting and splitting plays

Definition 10.18 (Play restriction)**.** Suppose that π is a play and consider the sequence obtained by restricting every believer move in π that contains ϕ to an identical move in which ϕ has been restricted to H_q. We denote this sequence by $\pi|^{\phi}_{H_q}$ and call it the *restriction* of the play π to H_q with respect to ϕ.

Theorem 10A. *For any play π of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, its restriction $\pi_q := \pi|^{\phi}_{H_q}$ is a valid play in $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$.*

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

Naturally we define splitting as a pair of restrictions:

Definition 10.19 (Play splitting)**.** Given a play π of $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, the *play splitting of π with respect to ϕ over $\mathcal{H}$* is the pair

$$\pi|^{\phi}_{\mathcal{H}} \triangleq \left(\pi|^{\phi}_{H_1}, \pi|^{\phi}_{H_2}\right).$$

Thanks to Theorem 10A, 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.

10.4 Strategies: combining, restricting, and splitting

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

Combining strategies

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

Definition 10.20 (Combination of strategies)**.** Given two strategies σ_1 and σ_2 in $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively, we define their *combination*

$$\sigma_1 \curlyvee \sigma_2 \triangleq \{\pi_1 \curlyvee \pi_2 \mid \pi_1 \in \sigma_1 \text{ and } \pi_2 \in \sigma_2\}\,.$$

Definition 10.21. Given a play $\pi \in \sigma := \sigma_1 \curlyvee \sigma_2$, we call the elements of the set

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

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

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

Lemma 10.21 (Reversibility of combination)**.** *Given a play $\pi \in \sigma := \sigma_1 \curlyvee \sigma_2$, we can extract in a unique way, two synchronized quasiplays τ_1 and τ_2, each of length $|\pi|$, such that $\tau_1 \ddot{\curlyvee} \tau_2 = \pi$ and both τ_q agree with σ_q, i.e.,* $\mathsf{rmstall}(\tau_q) \in \sigma_q$*.*

Proof. Corollary 10.19 guarantees that $|\pi|$ is even, which allows us to prove the lemma by induction on $\ell := |\pi| \,/\, 2$:

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

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

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

$$\delta_n := \delta_n^1 \mathbin{+\!\!+} \delta_n^2,$$

such that δ_n^q is a valid doubter response to τ_q'. We now use these δ_n^q to obtain the corresponding believer moves. For this we appeal to the fact that σ_q (being

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

Figure 10.1: Commutative diagram for Property 10.22.

deterministic) can have at most one next-move for the play $\mathsf{rmstall}\big(\tau'_q \mathbin{+\!\!+} \langle\delta^q_n\rangle\big)$, and we know that it has at least one since $\pi \in \sigma_1 \curlyvee \sigma_2$. We first define

$$b^q_n \coloneqq \big(\mathsf{answer}_{\sigma_q} \circ \mathsf{rmstall}\big)\,\big(\tau'_q \mathbin{+\!\!+} \langle\delta^q_n\rangle\big).$$

Note that as this is an answer from σ_q, the quasiplay $\tau'_q \mathbin{+\!\!+} \langle\delta^q_n, b^q_n\rangle$ remains in agreement with it. Now we finally obtain the believer moves by setting

$$(\beta^1_n, \beta^2_n) \coloneqq \begin{cases} (\overline{\delta^1_n}, b^2_n) & \text{if } \phi_1 \in b^1_n \text{ and } \phi_2 \notin b^2_n \\ (b^1_n, \overline{\delta^2_n}) & \text{if } \phi_1 \notin b^1_n \text{ and } \phi_2 \in b^2_n \\ (b^1_n, b^2_n) & \text{otherwise.} \end{cases} \qquad\qquad \blacksquare$$

Definition 10.22 (Projections of plays)**.** We call the quasiplay τ_q of the above lemma the *projection of* π *on* σ_q and denote it by $\mathsf{proj}_{\sigma_q}(\pi)$.

By their construction, projections satisfy the following property:

Property 10.22. *Given any* $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$*,*

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

In other words, denoting by T_q *the sets of quasiplays in* $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$*, the diagram in Figure 10.1 commutes.*

A link between projections and shortest creators is revealed: starting with π as above, first use Lemma 10.21 to obtain the quasiplay projections of π. Then, using $\mathsf{rmstall}$, remove all existing stall–follow moves; and finally, in case the quasiplay ended in a stall move, appeal to the strategy's totality to append the appropriate answer from σ_q. Following these steps, one actually obtains the shortest creators of π from σ_1 and σ_2. This will be clarified shortly in a lemma—but first, a definition that we will need:

Definition 10.23. Let $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$. We define the function $\mathsf{cr}_{\sigma_q} : \sigma \rightharpoonup \sigma_q$ as the composition

$$\mathsf{cr}_{\sigma_q} \triangleq \mathsf{answer}_{\sigma_q} \circ \mathsf{rmstall} \circ \mathsf{proj}_{\sigma_q},$$

and simply write

$$\mathsf{cr}(\pi) \triangleq (\mathsf{cr}_{\sigma_1}(\pi), \mathsf{cr}_{\sigma_2}(\pi))$$

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

Lemma 10.23. *Let $\pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$. Then*

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

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

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

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

$$\pi = \Big\langle \underbrace{\delta_0^1 +\!\!+ \delta_0^2}_{\delta_0},\ \underbrace{\beta_0^{1^*} +\!\!+ \beta_0^{2^*}}_{\beta_0}, \underbrace{\delta_1^1 +\!\!+ \delta_1^2}_{\delta_1},\ \underbrace{\beta_1^{1^*} +\!\!+ \beta_1^{2^*}}_{\beta_1}, \dots, \underbrace{\delta_n^1 +\!\!+ \delta_n^2}_{\delta_n},\ \underbrace{\beta_n^{1^*} +\!\!+ \beta_n^{2^*}}_{\beta_n} \Big\rangle,$$

where $n \coloneqq \min\{n_1, n_2\}$. Now, using the monotonicity of $\mathsf{rmstall}$ and $\mathsf{answer}_{\sigma_q}$ (Properties 10.3 and 10.4), we reason:

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

Notice that τ_q cannot end in a stalling move, and so $\mathsf{rmstall}(\tau_q)$ will have even length. According to Definition 10.11, we can then expect that

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

so that by using Property 10.14(iii) we derive

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

which is what we wanted to prove. ∎

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

Proposition 10.24. *Let $\varepsilon \neq \pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$, and let $\pi_q \coloneqq \mathsf{cr}_{\sigma_q}(\pi)$. Then*

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

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

Proof. This is a consequence of Property 10.22. Perhaps the weird-looking conditions need further explanation. The last believer move of π_q is needed to form the last move of π^- iff the penultimate move of $\mathsf{proj}_{\sigma_q}(\pi)$ is a stalling. This holds because once we project a play to σ_q, stalling moves might appear to the quasiplay projections. If the penultimate move of $\mathsf{proj}_{\sigma_q}(\pi)$ is such a stalling,

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

Figure 10.2a: The penultimate believer move of $\mathsf{proj}_{\sigma_2}(\pi)$ is a stalling.

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

Figure 10.2b: The penultimate believer move of $\mathsf{proj}_{\sigma_1}(\pi)$ is a stalling.

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

Figure 10.2c: Otherwise.

once we delete the last move from π_q to obtain π_q^-, and use $\mathsf{rmstall}$ to remove all stall–follow moves, the resulting play will be of odd length. $\mathsf{answer}_{\sigma_q}(\tau)$ will then reproduce the move we deleted, thus bringing us back to π_q.

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

This proposition might become clearer after inspecting the three commutative diagrams of Figure 10.2. There, the stated equality is represented by the commutativity of an appropriate diagram, one for each case (a)–(c).

Corollary 10.25. *Let $\varepsilon \neq \pi \in \sigma \coloneqq \sigma_1 \curlyvee \sigma_2$. Then*

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

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

Theorem 10B. *The set $\sigma \coloneqq \sigma_1 \curlyvee \sigma_2$, is a strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.*

Proof. Foremost it is indeed a set of plays in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ thanks to Corollary 10.17.

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

Even-length. This is a direct application of Corollary 10.19.

Even-prefix-closed. This is immediate by Corollary 10.25, since both σ_q are strategies and therefore closed under even prefixes.

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

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

For each of them, use Lemma 10.21 to extract its two quasiplay projections on σ_1 and σ_2; (τ_1, τ_2) from π and $(\widetilde{\tau_1}, \widetilde{\tau_2})$ from $\widetilde{\pi}$.

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

so that

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

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

Proposition 10.26 (Preservation of totality). *The combined strategy $\sigma_1 \curlyvee \sigma_2$ is total if both σ_1 and σ_2 are total.*

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

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

We seek a believer move β_{n+1} such that $\pi + \!\!\!+ \langle \delta_{n+1}, \beta_{n+1} \rangle \in \sigma$. Use Lemma 10.21 to extract the quasiplay projections (τ_1, τ_2) of π on σ_1 and σ_2, so that

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

Since δ_{n+1} is a valid next-move for π, it consists of doubts from the bodies of β_n^1 and β_n^2 (Property 10.6(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 σ_q, there must be at least one believer move β_{n+1}^q satisfying

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

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

Proposition 10.27 (Preservation of finiteness). *The combined strategy $\sigma_1 \curlyvee \sigma_2$ is finite if both σ_1 and σ_2 are finite.*

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

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

Corollary 10.28 (Preservation of winning)**.** *The combined strategy* $\sigma_1 \curlyvee \sigma_2$ *is winning if both* σ_1 *and* σ_2 *are winning.*

Restricting and splitting strategies

Definition 10.24 (Strategy restriction)**.** Let σ be a strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. Then the *restriction* of σ with respect to ϕ on H_q, is the set of plays defined by:

$$\sigma|^{\phi}_{H_q} \triangleq \left\{ \pi|^{\phi}_{H_q} \;\middle|\; \pi \in \sigma \right\} .$$

Theorem 10C. *The set of plays* $\sigma_q \coloneqq \sigma|^{\phi}_{H_q}$ *is a valid strategy in* $\Gamma_{\mathcal{P}_q}(\leftarrow \mathtt{G})$.

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

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

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

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

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

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

$$\begin{aligned} \pi_q &\coloneqq \langle \delta^q_0, \beta^q_0, \delta^q_1, \beta^q_1, \ldots, \delta^q_k, \beta^q_k \rangle , \\ \widetilde{\pi_q} &\coloneqq \left\langle \delta^q_0, \beta^q_0, \delta^q_1, \beta^q_1, \ldots, \delta^q_k, \widetilde{\beta^q_k} \right\rangle , \end{aligned}$$

so that there are plays

$$\begin{aligned} \pi &\coloneqq \langle \delta_0, \beta_0, \delta_1, \beta_1, \ldots, \delta_k, \beta_k \rangle , \\ \widetilde{\pi} &\coloneqq \left\langle \widetilde{\delta_0}, \widetilde{\beta_0}, \widetilde{\delta_1}, \widetilde{\beta_1}, \ldots, \widetilde{\delta_k}, \widetilde{\beta_k} \right\rangle , \end{aligned}$$

in σ, such that

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

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

Proposition 10.29 (Preservation of totality)**.** *If* σ *is total, then so is* $\sigma|^{\phi}_{H_q}$.

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

Proposition 10.30 (Preservation of finiteness)**.** *If σ is finite, then so is $\sigma|^{\phi}_{H_q}$.*

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

Corollary 10.31 (Preservation of winning)**.** *If σ is winning, then so is $\sigma|^{\phi}_{H_q}$.*

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

Definition 10.25 (Strategy splitting)**.** Let σ be a strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. The *splitting of σ with respect to ϕ over $\mathcal{H}$* is the pair

$$\sigma|^{\phi}_{\mathcal{H}} \triangleq \left(\sigma|^{\phi}_{H_1}, \sigma|^{\phi}_{H_2}\right).$$

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

10.5 Game semantics

This is essentially the same definition as in the LPG semantics:

Definition 10.26 (DLPG semantics)**.** Let $\mathcal{P}$ be a DLP program. A goal $\leftarrow \mathtt{G}$ *succeeds*, if Believer has a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.

The DLPG semantics

- $\mathcal{V}_{\mathsf{DLPG}} \triangleq \mathbb{B}$.
- $\mathcal{M}_{\mathsf{DLPG}}$ is the set of strategies based on DLP programs.
- $\mathbf{m}_{\mathsf{DLPG}}$ maps every DLP program to the set of corresponding winning strategies in the DLPG game.
- $\mathbf{a}_{\mathsf{DLPG}}(\Sigma)(Q) \triangleq \begin{cases} \mathsf{T}, & \text{if there is a winning strategy } \sigma \in \Sigma \text{ for } Q \\ \mathsf{F}, & \text{otherwise.} \end{cases}$

☡ Remark 10.13. 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! This is one more argument in favor of giving the benefit of the doubt to the Doubter.

𝄾 Remark 10.14 (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.[4] Nevertheless, we will prove in Lemma 10.32 that whenever there is a winning strategy in the DLP game of an LP program, then there is a winning strategy in which the believer never plays more than one rule, and is thus compatible with the LP game. In other words, the extra "combo-rule" of the DLP game is unnecessary in the absence of disjunctions.

10.6 Soundness and completeness

Theorem 10D (Soundness of the finite, clean DLP game semantics)**.** *Let $\mathcal{P}$ be a finite, clean DLP program, and $\leftarrow \mathtt{G}$ a goal. If there is a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, then G is true in every minimal model of $\mathcal{P}$.*

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

Base. Let σ be a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. We claim that *there exists a combo-free winning strategy σ' in the same game.* If so, observe that the head of the first believer move of every play of σ' must be one disjunct $\mathtt{g}$ of $\mathtt{G}$. Hence by altering the first (doubter) moves of its plays to correctly doubt $\mathtt{g}$, we end up with a combo-free, winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{g})$. This is now compatible with the LP game and we can use the soundness of the LP game semantics (Corollary 8.1) to obtain $g \in \mathrm{LHM}(\mathcal{P})$. But this means that G is true in $\mathrm{LHM}(\mathcal{P})$, the only minimal model of $\mathcal{P}$.

Proof of the claim. If σ contains no combo moves, we are done. Otherwise, pick a maximal play from σ such that it contains at least one combo move and look at the last one, β. We show that we can safely replace β by a non-combo move that contains exactly one of its rules, and still win the game. Towards a contradiction, suppose that no such rule exists. Then every rule in β contains a "bad" atom such that when doubted by the doubter, we cannot win. But this contradicts the fact that σ is winning, as it contains no answer for a doubter move that doubts exactly one bad atom from each rule of β.

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

Let σ be a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, and split it likewise to derive two strategies σ_1 and σ_2 for $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively (Theorem 10C). Since σ is winning, σ_1 and σ_2 are also winning (Corollary 10.31), and so by the

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

induction hypothesis we know that G must be true in every minimal model of $\mathcal{P}_1$ and in every minimal model of $\mathcal{P}_2$. In other words, G is true in the union $\mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2)$; so by Lemma 6.2, it is true in every element of $\mathrm{MM}(\mathcal{P})$. ▮

Theorem 10E (Completeness of the finite, clean DLPG game semantics)**.** *Let $\mathcal{P}$ be a finite, clean DLP program, and $\leftarrow \mathtt{G}$ a goal. If G is true in every minimal model of $\mathcal{P}$, then there is a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.*

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

Base. Since $\mathrm{LHM}(\mathcal{P}) \models G$, there is at least one $g \in G$ with $g \in \mathrm{LHM}(\mathcal{P})$. Using the completeness of the LP game semantics (Corollary 8.1), we obtain a winning strategy σ in the LP game for $\mathcal{P}$ with the goal $\leftarrow \mathtt{g}$. Without any modification, we can consider this as a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{g})$. It remains to correct all first (doubter) moves by including all extra doubts from $\mathtt{G}$ in them. In this way, we have a winning strategy σ' in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$.

Induction step. Again, pick a proper DLP rule $\phi \in \mathcal{P}$, and split $\mathcal{P}$ to get $(\mathcal{P}_1, \mathcal{P}_2)$. We know that G is true in every minimal model of $\mathcal{P}$. Therefore, it is also true in every model of $\mathcal{P}$ (because a non-minimal model can only make more formulæ true than a minimal one, not less). Using Lemma 6.2 again, $\mathrm{MM}(\mathcal{P}_1) \cup \mathrm{MM}(\mathcal{P}_2) \subseteq \mathrm{HM}(\mathcal{P})$, so that G is true in every minimal model of $\mathcal{P}_1$ and in every minimal model of $\mathcal{P}_2$. By the induction hypothesis, there are two winning strategies σ_1 and σ_2 in the games $\Gamma_{\mathcal{P}_1}(\leftarrow \mathtt{G})$ and $\Gamma_{\mathcal{P}_2}(\leftarrow \mathtt{G})$ respectively. Using Theorem 10B we can combine them to get a new strategy $\sigma_1 \curlyvee \sigma_2$ for $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, which is winning by Corollary 10.28. ▮

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

Lemma 10.32. *Let $\mathcal{P}$ be a general DLP program, and $\leftarrow \mathtt{G}$ a DLP goal. Then, there exists a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ iff there is a winning strategy in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$.*

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

$$\beta \coloneqq \langle \psi_1, \ldots, \psi_n \rangle .$$

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

$$\widehat{\beta} \coloneqq \widehat{\psi_1} \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \widehat{\psi_n}.$$

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

$D^i \subseteq \left[\,\widehat{\delta_i}\,\right]$. Then, our move in $\Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})$ against a doubter move consisting of such doubts

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

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

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

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

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

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

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

which is impossible.

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

$$\begin{aligned} \beta^{\vee} &:= \langle \psi_1^{\vee}, \ldots, \psi_n^{\vee} \rangle\,, \\ \text{where}\quad \psi_i^{\vee} &:= \text{the first } \rho \in \mathcal{P} \text{ such that } \psi_i \in \widehat{\rho}. \end{aligned}$$

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

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

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

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

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

Theorem 10F (Soundness and completeness for finite DLP)**.** *Let $\mathcal{P}$ be a finite, general DLP program, and $\leftarrow \mathtt{G}$ a DLP goal. Then, there exists a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ iff G is true in every minimal model of $\mathcal{P}$.*

Proof. We have proved everything we need:

$$\begin{array}{cccl}
\begin{array}{c}\text{there is a winning}\\ \text{strategy in } \Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})\end{array} & \Longleftrightarrow & \begin{array}{c}\text{there is a winning}\\ \text{strategy in } \Gamma_{\widehat{\mathcal{P}}}(\leftarrow \mathtt{G})\end{array} & \text{(Lemma 10.32)} \\
 & \Longleftrightarrow & \begin{array}{c}G \text{ is true on every}\\ \text{minimal model of } \widehat{\mathcal{P}}\end{array} & \begin{array}{l}\text{(Theorem 10D} \Rightarrow\text{)}\\ \text{(Theorem 10E} \Leftarrow\text{)}\end{array} \\
 & \Longleftrightarrow & \begin{array}{c}G \text{ is true on every}\\ \text{minimal model of } \mathcal{P}.\end{array} & \text{(Property 3.1)}
\end{array}$$

▮

We now drop the heavy restriction of finiteness. This has the remarkable consequence that we can handle $\mathsf{ground}(\mathcal{P})$ for any first-order DLP program $\mathcal{P}$ (see also Remark 4.2).

Theorem 10G (Soundness and completeness for general DLP)**.** *Let $\mathcal{P}$ be a general DLP program, and $\leftarrow \mathtt{G}$ a disjunctive goal. Then, there exists a winning strategy in $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$ iff G is true in every minimal model of $\mathcal{P}$.*

Proof. Completeness. Assume G is true in every minimal model of $\mathcal{P}$, and therefore also in every model of $\mathcal{P}$, in symbols $\mathcal{P} \models G$. By the compactness theorem of propositional calculus, there exists a finite subset $\mathcal{P}_G \subseteq \mathcal{P}$ such that $\mathcal{P}_G \models G$. Specifically, G is true in every minimal model of $\mathcal{P}_G$, which allows us to use Theorem 10F to obtain a winning strategy σ_G in $\Gamma_{\mathcal{P}_G}(\leftarrow \mathtt{G})$. Observe now that this very strategy is still winning in the "bigger" game $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$, since the believer will never pick any of the additional rules, and therefore the course of the game cannot change.

Soundness. Suppose we have a winning strategy σ for $\Gamma_{\mathcal{P}}(\leftarrow \mathtt{G})$. This can only use a finite number of rules from $\mathcal{P}$, which form a finite subset $\mathcal{P}_\sigma \subseteq \mathcal{P}$. Obviously, σ is still winning in $\Gamma_{\mathcal{P}_\sigma}(\leftarrow \mathtt{G})$. This means that G is true in every minimal model of $\mathcal{P}_\sigma$ (by Theorem 10F again), and therefore in every model of $\mathcal{P}_\sigma$. Since there are no negations involved, G remains true in every model of the superset $\mathcal{P}$; specifically in every minimal one. ▮

Corollary 10.33 (Soundness and completeness for first-order DLP)**.** *Let $\mathcal{P}$ be a first-order DLP program, and C a positive ground clause. Then $\mathcal{P} \models C$ iff there exists a winning strategy in $\Gamma_{\mathsf{ground}(\mathcal{P})}(\leftarrow \mathtt{C})$.*

Proof.

$$\begin{array}{rcll} \mathcal{P} \models C & \iff & C \text{ is true in every m.m. of } \mathcal{P} & \text{(Theorem 4B)} \\ & \iff & C \text{ is true in every m.m. of } \mathsf{ground}(\mathcal{P}) & \text{(Property 4.2)} \\ & \iff & \text{there is a winning } \sigma \text{ in } \Gamma_{\mathsf{ground}(\mathcal{P})}(\leftarrow \mathtt{C}) & \text{(Theorem 10G)} \quad ▮ \end{array}$$

So far, we have studied model-theoretic semantics for all four languages, while game semantics for all but DLPN. In the next chapter we will apply the tools we have developed so far to fill this gap, and also obtain a new, different game for DLP, which still provides equivalent semantics.

7

Chapter 11

Applications of $(-)^{\vee}$ on game semantics

In this chapter, we use the $(-)^{\vee}$ operator on the game semantics of LP and LPN of Chapters **8** and **9** to obtain new, game-theoretic semantics for DLP and DLPN. For DLP, the obtained game semantics is drastically different, albeit equivalent to DLPG, and for DLPN, this appears to be its first game-theoretic semantics. For the same reasons of simplicity that we outlined in the beginning of Chapter **7**, *we again assume that all programs are clean in this chapter.* And as there has been no formal definition of a game semantics for *infinite* LPN programs, *programs with negation are also assumed to be finite.*

11.1 A different game semantics for DLP

We pick the simplest of the games we have encountered, LPG, and apply the $(-)^{\vee}$ operator on it. Following the definition, we have:

$$\mathcal{V}_{\mathsf{LPG}_{\vee}} = \mathcal{V}_{\mathsf{LPG}} = \mathbb{B}$$
$$\mathcal{M}_{\mathsf{LPG}_{\vee}} = \mathcal{P}(\mathcal{M}_{\mathsf{LPG}}).$$

We proceed following the definitions:

$$\begin{aligned}
\mathbf{m}_{\mathsf{LPG}_{\vee}}(\mathcal{P}) &= (\mathbf{m}_{\mathsf{LPG}})^{\vee}(\mathcal{P}) = \mathbf{m}_{\mathsf{LPG}}(\mathrm{D}(\mathcal{P})),\\
\mathbf{a}_{\mathsf{LPG}_{\vee}}(\mathcal{S})(Q) &= (\mathbf{a}_{\mathsf{LPG}})^{\vee}(\mathcal{S})(Q) = \bigwedge\nolimits_{S\in\mathcal{S}} \bigvee\nolimits_{q\in Q} \mathbf{a}_{\mathsf{LPG}}(S)(q),\\
\mathbf{s}_{\mathsf{LPG}_{\vee}}(\mathcal{D})(Q) &= (\mathbf{s}_{\mathsf{LPG}})^{\vee}(\mathcal{D})(Q) = \bigwedge\nolimits_{\mathcal{P}\in\mathrm{D}(\mathcal{D})} \bigvee\nolimits_{q\in Q} \mathbf{s}_{\mathsf{LPG}}(\mathcal{P})(q).
\end{aligned}$$

Translating the last equation, we see that

$$\begin{aligned}
\mathbf{s}_{\mathsf{LPG}_{\vee}}(\mathcal{D})(Q) = \mathbf{T} &\iff \textstyle\bigwedge_{\mathcal{P}\in\mathrm{D}(\mathcal{D})} \bigvee_{q\in Q} \mathbf{s}_{\mathsf{LPG}}(\mathcal{P})(q) = \mathbf{T}\\
&\iff \begin{array}{l}\text{for every } \mathcal{P}\in\mathrm{D}(\mathcal{D}), \text{ there exists a } q\in Q,\\ \text{such that } \mathbf{s}_{\mathsf{LPG}}(\mathcal{P})(q) = \mathbf{T}\end{array}\\
&\iff \begin{array}{l}\text{for every } \mathcal{P}\in\mathrm{D}(\mathcal{D}), \text{ there exists a } q\in Q,\\ \text{such that there is a winning strategy for } \Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow q).\end{array}
\end{aligned}$$

Easily, we can interpret these quantifiers as player moves, so that in the $\mathsf{LPG}_{\vee}$ game, Doubter begins by playing a definite instantiation $\mathcal{P} \in \mathrm{D}(\mathcal{D})$ (essentially what she chooses is a $\mathcal{D}$-section). The following move of the Believer is to choose an element of the goal $q \in Q$, and after this point, the players begin playing the game $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow q)$.

Theorem 11A. *The game semantics* $\mathsf{LPG}_{\vee}$ *and the minimal model semantics* MM *are equivalent.*

Proof. We begin with the equivalence

$$\mathbf{s}_{\mathsf{LPG}} = \mathbf{s}_{\mathsf{LHM}}, \qquad \text{(by Corollary 8.1)}$$

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

$$\begin{aligned} (\mathbf{s}_{\mathsf{LPG}})^{\vee} &= (\mathbf{s}_{\mathsf{LHM}})^{\vee} && \text{(by Lemma 7.1)} \\ &= \mathbf{s}_{\mathsf{MM}} && \text{(by Theorem 7B)} \end{aligned}$$

∎

Using Theorem 10G, we obtain the following

Corollary 11.1. *The game semantics* $\mathsf{LPG}_{\vee}$ *and* DLPG *are equivalent.*

11.2 An encoding of DLP to LP

Suppose now that we are given a *finite* DLP program $\mathcal{D}$ and a DLP goal $G \coloneqq \{g_1, \dots, g_m\}$. We outline here how we can encode both objects into ones of LP. We will define an operator

$$\mathit{encode} \ : \ \mathbf{P}_{\mathrm{DLP}} \times \mathbf{Q}_{\mathrm{DLP}} \to \mathbf{P}_{\mathrm{LP}} \times \mathbf{Q}_{\mathrm{LP}},$$

so that if $\mathit{encode}(\mathcal{D}, G) = (\mathcal{P}, g)$, we will then be able to use the game $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow \mathtt{g})$ to obtain an answer for the initial DLP query G, with respect to the initial program $\mathcal{D}$.

Before diving into the definition of the encoding, note that since $\mathcal{D}$ is finite, then so is $\mathrm{D}(\mathcal{D}) \coloneqq \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$. Now define

$$\mathit{encode}(\mathcal{D}, G) \triangleq (\mathcal{P}, g)$$

where

$$\begin{aligned} \mathcal{P} &\coloneqq \mathcal{P}_1 \uplus \cdots \uplus \mathcal{P}_n \cup \mathit{restrictors}(\mathcal{D}, G) \cup \{\mathit{definitizer}(\mathcal{D}, G)\}, \\ \mathit{restrictors}(\mathcal{D}, G) &\triangleq \left\{\mathtt{p}_i \ \leftarrow \ \mathtt{g}^i_j \mid 1 \le i \le n,\ 1 \le j \le m\right\}, \\ \mathit{definitizer}(\mathcal{D}, G) &\triangleq \mathtt{g} \ \leftarrow \ \mathtt{p}_1 \,, \cdots, \mathtt{p}_n, \end{aligned}$$

and where all atoms p_i and g, are distinct and fresh, i.e., not appearing anywhere in $\mathcal{D}$, G, or any of the $\mathcal{P}_i$'s, while every occurrence of the atom g_j in the program $\mathcal{P}_i$, gives rise to an occurrence of the "tagged" atom g^i_j inside the disjoint union $\biguplus \mathrm{D}(\mathcal{D})$.

Some explanations are due. Firstly, we take the disjoint union of the definite instantiations of $\mathcal{D}$, tagging each atom that appears in $\mathcal{P}_i$ with an ^{i} superscript, as we have already seen for the g_j's. Next, the rôle of the definitizer rule, is to select a specific definite instantiation $\mathcal{P}_i \in \mathrm{D}(\mathcal{D})$. On the other hand, for each instantiation, there are corresponding restrictors whose job is to restrict the disjunctive goal into one of its disjuncts, a single atom.

Examining the game $\Gamma^{\mathrm{LP}}_{\mathcal{P}}(\leftarrow \mathtt{g})$ will shed some light to this: The Opponent begins by doubting $\mathtt{g}$. Next, the Player is forced to play the only rule whose head is g:

$$\mathtt{g} \leftarrow \mathtt{p}_1\,, \cdots, \mathtt{p}_n.$$

Opponent know has to select a $\mathtt{p}_i$. This corresponds to her choice of a definite instantiation of $\mathcal{D}$. As we have seen, Player next gets to decide which element $g_j \in G$ he wants to restrict to, and this is exactly what the restrictors are for: Player will choose the rule $\mathtt{p}_i \leftarrow \mathtt{g}^i_j$. Opponent has no choice but to doubt the only conjunct in the body of that rule, $\mathtt{g}^i_j$, and finally we have arrived in this tagged atom, and so the game is successfully restricted within the rules of the correspondingly tagged $\mathcal{P}_i$.

Notice that in essence, the game just described is really the one of Section **11.1**, but without having to add those artificial rules that we did there: it is the encoding that does their job instead.

11.3 A first game semantics for DLPN

As we have already mentioned, there appears to be no game semantics for DLPN in the literature. In this section, we ameliorate this situation: we obtain two game semantics by using the $(-)^{\vee}$ operator on LPNG and LPNG^{ω}.

According to its definition,

$$\begin{aligned} \mathcal{V}_{\mathsf{LPNG}_{\vee}} &= \mathcal{V}_{\mathsf{LPNG}} = \mathbb{V}_1, & \mathcal{V}_{\mathsf{LPNG}^{\omega}_{\vee}} &= \mathcal{V}_{\mathsf{LPNG}^{\omega}} = \mathbb{V}_{\omega}, \\ \mathcal{M}_{\mathsf{LPNG}_{\vee}} &= \mathcal{P}(\mathcal{M}_{\mathsf{LPNG}}); & \mathcal{M}_{\mathsf{LPNG}^{\omega}_{\vee}} &= \mathcal{P}(\mathcal{M}_{\mathsf{LPNG}^{\omega}}). \end{aligned}$$

Focusing on LPNG^{ω}, we have

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

and similarly for $\mathsf{LPNG}_{\vee}$.

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

Theorem 11B. *For finite DLPN programs, the game semantics $\mathsf{LPNG}^{\omega}_{\vee}$ and the model-theoretic semantics MM^{ω} are equivalent.*

Proof. Completely analogously to the proof of Theorem 11A, starting from the equivalence

$$\mathbf{s}_{\mathsf{LPNG}^{\omega}} = \mathbf{s}_{\mathsf{WF}^{\omega}}, \qquad \text{(by Theorem 9A)}$$

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

$$\begin{aligned}
(\mathsf{s}_{\mathsf{LPNG}^\omega})^\vee &= (\mathsf{s}_{\mathsf{WF}^\omega})^\vee && \text{(by Lemma 7.1)} \\
&= \mathsf{s}_{\mathsf{MM}^\omega} && \text{(by Theorem 7C).}
\end{aligned}$$ ▮

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

𝄾 Remark 11.1. The encoding we presented in Section **11.2** can be applied *mutatis mutandis* for the case of DLPN programs, encoding them into LPN ones. Note that since negations only occur inside the bodies of the $\mathcal{P}_i$'s, the rôle-switching moves may only be played once the play has already been restricted to a specific definite instantiation $\mathcal{P}_i$.

𝄽

Part IV

Appendices

Appendix A

Lattices and Heyting algebras

Everything in this appendix can be found in standard texts on such subjects, e.g., [DP02].

Definition A.1. Let L be a non-empty ordered set. L is a *lattice* iff both $x \vee y$ and $x \wedge y$ exist for all $x, y \in L$. L is a *complete lattice* iff $\bigvee S$ and $\bigwedge S$ exist for all $S \subseteq L$. We say that L has a *one* iff there is an element $1 \in L$ such that $x = x \wedge 1$ for all $x \in L$, and a *zero* iff there is an element $0 \in L$ such that $x = x \vee 0$ for all $x \in L$. Other notations for these two elements are $\top$ and $\bot$ respectively. If such two elements exist in a lattice, we call it *bounded*.

Definition A.2. Given a lattice $(L; \leq_L)$, we form its *dual lattice* $(L^\partial; \leq_{L^\partial})$ by defining

$$x \leq_{L^\partial} y \iff y \leq_L x.$$

Definition A.3. A lattice L is *distributive* iff for all $a, b, c \in L$,

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$$

Definition A.4. A complete lattice L is said to be *completely distributive* iff for any doubly indexed family $\{x_{j,k}\}_{j \in J, k \in K_j} \subseteq L$,

$$\bigwedge\nolimits_{j \in J} \bigvee\nolimits_{k \in K_j} x_{j,k} = \bigvee\nolimits_{f \in F} \bigwedge\nolimits_{j \in J} x_{j,f(j)},$$

where F is the set of all choice functions f that choose for each $j \in J$, an element $f(j) \in K_j$. That is,

$$F = \left\{ f \,:\, J \to \bigcup\nolimits_{j \in J} K_j \mid f(j) \in K_j \right\} = \prod\nolimits_{j \in J} K_j.$$

Definition A.5. Let L be a complete lattice. $k \in L$ is *compact*, if for every $S \subseteq L$, there is a finite $T \subseteq S$, such that

$$k \leq \bigvee S \implies k \leq \bigvee T.$$

The set of all compact elements of L is denoted by $K(L)$.

Definition A.6. A complete lattice L is said to be *algebraic* if, for each $a \in L$,

$$a = \bigvee \{k \in K(L) \mid k \leq a\}\,.$$

Theorem A.7. *Let L be a lattice. Then the following are equivalent:*

(i) *L is distributive and both L and L^∂ are algebraic;*

(ii) *L is completely distributive and algebraic.*

Proof is in [DP02, Theorem 10.29]. ▮

Definition A.8. Let H be a (complete) bounded lattice. H is a (complete) *Heyting algebra* iff for every $a, b \in H$ there is an element $a \to b$ satisfying

$$c \leq a \to b \iff c \wedge a \leq b$$

for every $c \in H$.

Property A.9 (Some equalities)**.** *Let H be a Heyting algebra. Then the following equalities hold between elements of H:*

(i) $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

(ii) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

(iii) $a \Rightarrow (b \vee c) = (a \Rightarrow b) \vee (a \Rightarrow c)$

(iv) $(a \vee b) \Rightarrow c = (a \Rightarrow c) \wedge (b \Rightarrow c)$.

If in addition H is complete*, the following equalities also hold:*

(I) $a \wedge \bigvee_{s \in S} s = \bigvee_{s \in S}(a \wedge s)$

(II) $a \vee \bigwedge_{s \in S} s = \bigwedge_{s \in S}(a \vee s)$

(III) $a \Rightarrow \bigvee_{s \in S} s = \bigvee_{s \in S}(a \Rightarrow s)$

(IV) $(\bigvee_{s \in S} s) \Rightarrow c = \bigwedge_{s \in S}(s \Rightarrow c)$.

References

[AB94] Krzysztof R. Apt and Roland Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19:9–71, 1994.

[Acz77] Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, Amsterdam, 1977.

[AJM00] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, December 2000.

[ALM09] Gianluca Amato, James Lipton, and Robert McGrail. On the algebraic structure of declarative programming languages. *Theoretical Computer Science*, 410(46):4626–4671, 2009.

[AM99] S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Computational Logic: Proceedings of the 1997 Marktoberdorf Summer School*, pages 1–56. Springer-Verlag, 1999.

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

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

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

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

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

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

[Cla79] Keith Clark. Predicate logic as a computational formalism. 1979.

[CM92] Andrea Corradini and Ugo Montanari. An algebraic semantics for structured transition systems and its applications to logic programs. *Theoretical Computer Science*, 103(1):51–106, 1992.

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

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

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

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

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

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

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.

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

[HO00] J. M. E. Hyland and C.-H. Ong. On full abstraction for PCF: I. Models, observables and the full abstraction problem; II. Dialogue games and innocent strategies; III. A fully abstract and universal game model. *Information and Computation*, 163:285–408, 2000.

[Hod09] Wilfrid Hodges. Logic and games. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Spring 2009 edition, 2009.

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

[KNR11] Vassilis Kountouriotis, Christos Nomikos, and Panos Rondogiannis. A game-theoretic characterization of boolean grammars. *Theoretical Computer Science*, 412(12-14):1169–1183, 2011.

[KP96] Yoshiki Kinoshita and A. John Power. A fibrational semantics for logic programs. In Roy Dyckhoff, Heinrich Herre, and Peter Schroeder-Heister, editors, *ELP*, volume 1050 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 1996.

[KP11] Ekaterina Komendantskaya and John Power. Coalgebraic semantics for derivations in logic programming. In *Proceedings of the 4th international conference on Algebra and coalgebra in computer science*, CALCO'11, pages 268–282, Berlin, Heidelberg, 2011. Springer-Verlag.

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

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

[LC00] Jean-Vincent Loddo and Roberto Di Cosmo. Playing logic programs with the alpha-beta algorithm. In Michel Parigot and Andrei Voronkov, editors, *LPAR*, volume 1955 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2000.

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

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

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

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

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

[Min82] Jack Minker. *On indefinite databases and the closed world assumption*, volume 138 of *Lecture Notes in Computer Science*. Springer-Verlag, 1982.

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

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

[MS06] Dale Miller and Alexis Saurin. A game semantics for proof search: Preliminary results. *Electronic Notes in Theoretical Computer Science*, 155:543–563, 2006.

[Nic94] Hanno Nickau. Hereditarily sequential functionals. In Anil Nerode and Yuri Matiyasevich, editors, *LFCS*, volume 813 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 1994.

[NR12] Christos Nomikos and Panos Rondogiannis. A game semantics for intensional logic programming. Presented in Games for Logic and Programming Languages (GaLoP) VII, Dubrovnik, Croatia, 2012.

[NV07] Jonty Needham and Marina De Vos. A games semantics of ASP. In Véronica Dahl and Ilkka Niemelä, editors, *Logic Programming*, volume 4670 of *Lecture Notes in Computer Science*, pages 460–461. Springer Berlin Heidelberg, 2007.

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

[Rei78] R Reiter. On closed world data bases. *Logic and Databases*, pages 55–76, 1978.

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

[Tso10] Thanos Tsouanas. Semantic approaches to logic programming. Master's thesis, 2010.

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

[Vää11] Jouko Väänänen. *Models and Games*. Cambridge series in advanced mathematics. Cambridge University Press, 2011.

[vE86] M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3:37–53, 1986.

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

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

Index of symbols

General index

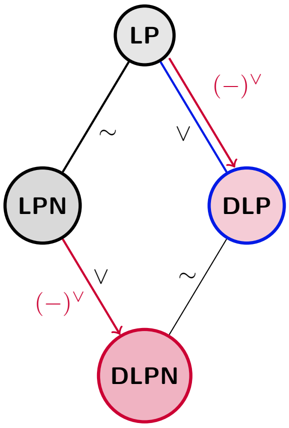
```
~
~
~
~
~
~
:wq
```

On the Semantics of Disjunctive Logic Programs: **Thesis summary**

Programming paradigms. Programming languages come in different forms and flavors. Still, there are some major paradigms of programming, that most languages can be seen to belong to one or another, or to even combine features of more than one of them. Under this perspective, the two main families of languages are *imperative* and *declarative*. Roughly speaking, in imperative languages we program by specifying *how* to reach a solution, in a way that resembles how one would write down a cooking recipe. On the other hand, when programming in declarative languages we define *what* the problem is and *what* is a solution of it, but we leave the step-by-step details (the *how*) to the implementation of the language. Code in declarative languages tends to be more elegant and such languages are also much better suited for reasoning about programs and proving useful properties. The two main paradigms that fall under this family of languages are *functional programming* and *logic programming*. In the first, we program by mathematically defining and evaluating functions; in the latter, a program consists of logic formulæ of the form `this` ← `that`, meant to be read as "*this* holds if *that* holds", or "to solve *this* problem, it suffices to solve *that* one".

Syntax and semantics. A programming language is a two-sided coin, and to understand one and use it well, we need to know both of its sides: its *syntax* and its *semantics*. Syntax dictates what constitutes a well-formed program for a language: how expressions are formed, what symbols or words are allowed and where, etc. But what exactly is *the meaning* of a correctly written program? Answering this question is the job of the language's semantics, which also come in different kinds, the three main ones being *denotational*, *operational*, and *axiomatic*. In denotational semantics one usually defines and uses appropriate *mathematical objects* to denote meanings of expressions and programs; without such a formalism we cannot hope to prove much about a program. For logic programs, the traditional denotational semantics are *model-theoretic*, but recently *game-theoretic* semantics have been studied as well. An operational semantics describes (either in a step-by-step fashion or in a so-called "big-step" style) the execution of programs and is thus closer to the implementation of a language. Finally, axiomatic semantics specify what effect the execution of statements and programs will have on a set of assertions that are assumed to hold before we perform it.

Four logic programming languages. *In this thesis, we study denotational semantics of four logic programming languages*: (1) **LP** which is the simplest case of logic programs; (2) **DLP** which extends LP by adding disjunctions (∨); (3) **LPN** which extends LP by adding negations (∼); and (4) **DLPN** which allows both disjunctions and negations. The following figure represents these extensions schematically:



$$
\begin{aligned}
\mathsf{a} &\leftarrow \mathsf{b}_1\,,\cdots,\mathsf{b}_n && \text{(LP rule)}\\
\mathsf{a}_1 \vee \cdots \vee \mathsf{a}_k &\leftarrow \mathsf{b}_1\,,\cdots,\mathsf{b}_n && \text{(DLP rule)}\\
\mathsf{a} &\leftarrow \mathsf{b}_1\,,\cdots,\mathsf{b}_n\,,{\sim}\mathsf{c}_1\,,\ldots,{\sim}\mathsf{c}_m && \text{(LPN rule)}\\
\underbrace{\mathsf{a}_1 \vee \cdots \vee \mathsf{a}_k}_{\texttt{this}} &\leftarrow \underbrace{\mathsf{b}_1\,,\ldots,\mathsf{b}_n\,,{\sim}\mathsf{c}_1\,,\ldots,{\sim}\mathsf{c}_m}_{\texttt{that}} && \text{(DLPN rule)}
\end{aligned}
$$

The four logic programming languages studied in this thesis, and a representative program rule for each of them. In color, some of the contributions outlined below.

Contributions. The three main contributions of this dissertation can be summarized as follows:

- An *abstract framework for logic programming semantics* is defined, and all semantic approaches that we study are placed within this framework. In this process we define the general notion of a *truth value space* as an appropriate algebraic structure that satisfies a set of axioms. The booleans form the canonical example of such a space, but we need to consider much more general ones when dealing with negation. For this we define and study an infinite family of spaces, parametrized by an ordinal number.

- A game semantics for LP has been known for quite a long time. In 2005, it was extended to a new one for the language of LPN. Here a *novel game semantics for DLP programs* is developed in full detail and we prove that it is sound and complete with respect to the standard, minimal model semantics of Minker. What is more, this is done for infinite programs, which has the remarkable consequence that it provides a denotational semantics even for first-order programs. Even though the game itself can be seen as an extension of the LP game, the formalization and notation we have used is influenced by the games used in functional programming instead.

- We define a *semantic operator* $(-)^\vee$ which transforms any given semantics of a non-disjunctive logic programming language to a semantics of the "corresponding" disjunctive one. We exhibit the correctness of this transformation by proving that it preserves equivalences of semantics, and that the semantics it yields behave as one would expect them to. We present some *applications of this operator* to obtain some new semantics for disjunctive logic programming languages: notably, applying this method on the game semantics for LPN, we obtain a game semantics for DLPN, filling the gap that remained to complete the game semantics part of the picture sketched above.

Keywords: disjunctive logic programming; denotational semantics; game semantics; logic programming; theory of programming languages.