The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

Splitting Answer Set Programs with Respect to Intensionality Statements

Jorge Fandinno[*] **and Yuliya Lierler**[*]

University of Nebraska at Omaha
{jfandinno,ylierler}@unomaha.edu

Abstract

Splitting a logic program allows us to reduce the task of computing its stable models to similar tasks for its subprograms. This can be used to increase solving performance and to prove the correctness of programs. We generalize the conditions under which this technique is applicable, by considering not only dependencies between predicates but also their arguments and context. This allows splitting programs commonly used in practice to which previous results were not applicable.

Introduction

Answer set programming (ASP; Lifschitz 2008) is a declarative logic programming paradigm well-suited for solving knowledge-intensive search problems. Its success relies on the combination of a rich knowledge representation language with efficient solvers for finding solutions to problems expressed in this language (Lifschitz 2019). Solutions for logic programs in ASP are called "stable models". Lifschitz and Turner (1994) introduced a fundamental *splitting* method (or, simply, splitting) in the theory of ASP. Often, splitting is used to reduce the task of computing the stable models of a logic program to the task of computing these of its subprograms. Also, the splitting is used to understand the meaning of a program in terms of its smaller components and it has become a key instrument in constructing proofs of correctness for logic programs (Cabalar, Fandinno, and Lierler 2020). The original condition required to split a logic program was generalized by Oikarinen and Janhunen (2008) in the context of propositional programs. Also, when stable models were defined for arbitrary first-order formulas (Pearce and Valverde 2005; Ferraris, Lee, and Lifschitz 2007) and infinitary propositional formulas (Truszczyński 2012), the splitting method was extended to these new settings (Ferraris et al. 2009; Harrison and Lifschitz 2016). However, when dealing with first-order formulas (logic programs with variables are often identified with first-order formulas of a restricted syntactic form), the splitting condition is not sufficiently general for some applications.

For instance, consider the following simplified fragment of the blocks world encoding (Lifschitz 2002):

$$on(B,L,T+1) \leftarrow on(B,L,T),\ not\ non(B,L,T+1) \quad (1)$$
$$on(B,L,T+1) \leftarrow move(B,L,T) \quad (2)$$
$$non(B,L',T) \leftarrow on(B,L,T),\ location(L'),\ L \neq L' \quad (3)$$

We are interested in splitting this fragment so that one subprogram contains instances of these rules for all time points before a (positive) threshold t and another contains rules corresponding to the time points at or after t. This form of splitting will help, for example, understanding iterative solving of planning problems (Gebser et al. 2019) and arguing correctness of action descriptions (that frequently result in rules of the form presented in our blocks world example). However, the proposed splitting is impossible under the currently available Splitting Theorem for first-order formulas (Ferraris et al. 2009). The splitting relies on identifying "non-circularly" dependent rules of a program. Ferraris et al. (2009) define dependencies in terms of predicate symbols. In our example, $on/3$ depends on itself in rule (1).

Here, we formulate a more general condition for the applicability of the splitting method that considers a refined version of the dependency graph that takes into account not only the predicate dependencies but also their arguments and context. Among others, this makes it applicable to the discussed example. The notion of *intensionality statement* introduced in this work is of key importance. This is a refinement of the idea of intensional and extensional predicates (Ferraris, Lee, and Lifschitz 2011) originally stemming from databases. It provides us with means for specifying arguments of the predicates for which these are seen as intensional or extensional. Thus, the same predicate symbol may be both intensional and extensional depending on its context. This refinement is the basis for a new Splitting result.

The rest of the paper is organized as follows. After reviewing some preliminaries, we introduce the notion of intensionality statement. Then, we present the key ideas for the new Splitting Theorem in the context of logic programs. Finally, we generalize this result to first-order theories.

Preliminaries

We start by reviewing a many-sorted first-order language. The use of a many-sorted language is motivated by its ability to formalize commonly used features of logic programs

[*]These authors contributed equally.

such as arithmetic operations (Fandinno et al. 2020; Lifschitz 2021) and aggregate expressions (Fandinno, Hansen, and Lierler 2022). We follow the presentation by Fandinno and Lifschitz (2022, Appendix A). After the presentation of many-sorted language, the review the logic of here-and-there follows. Syntax and semantics of disjunctive logic programs concludes this section.

Many-sorted first-order theories. A (many-sorted) signature consists of symbols of three kinds—*sorts*, *function constants*, and *predicate constants*. A reflexive and transitive *subsort* relation is defined on the set of sorts. A tuple $s_1, \dots, s_n$ ($n \geq 0$) of *argument sorts* is assigned to every function constant and to every predicate constant; in addition, a *value sort* is assigned to every function constant. Function constants with $n = 0$ are called *object constants*. We assume that for every sort, an infinite sequence of *object variables* of that sort is chosen. *Terms* over a (many-sorted) signature σ are defined recursively as usual. *Atomic formulas* over σ are either (i) expressions of the form $p(t_1, \dots, t_n)$, where p is a predicate constant and $t_1, \dots, t_n$ are terms such that their sorts are subsorts of the argument sorts $s_1, \dots, s_n$ of p, or (ii) expressions of the form $t_1 = t_2$, where t_1 and t_2 are terms such that their sorts have a common supersort. *(First-order) formulas* over signature σ are formed from atomic formulas and the 0-place connective $\bot$ (falsity) using the binary connectives $\wedge$, $\vee$, $\rightarrow$ and the quantifiers $\forall$, $\exists$. The other connectives are treated as abbreviations: $\neg F$ stands for $F \rightarrow \bot$ and $F \leftrightarrow G$ stands for $(F \rightarrow G) \wedge (G \rightarrow F)$.

An *interpretation* I of a signature σ assigns

- a non-empty *domain* $|I|^s$ to every sort s of I, so that $|I|^{s_1} \subseteq |I|^{s_2}$ whenever s_1 is a subsort of s_2,
- a function f^I from $|I|^{s_1} \times \cdots \times |I|^{s_n}$ to $|I|^s$ to every function constant f with argument sorts $s_1, \dots, s_n$ and value sort s, and
- a Boolean-valued function p^I on $|I|^{s_1} \times \cdots \times |I|^{s_n}$ to every predicate constant p with argument sorts $s_1, \dots, s_n$.

If I is an interpretation over a signature σ then by σ^I we denote the signature obtained from σ by adding, for every element d of domain $|I|^s$, its *name* d^* as an object constant of sort s. The interpretation I is extended to σ^I by defining $(d^*)^I = d$. The value t^I assigned by an interpretation I of σ to a ground term t over σ^I and the satisfaction relation (denoted, $\models$) between an interpretation of σ and a sentence over σ^I are defined recursively, in the usual way (Lifschitz, Morgenstern, and Plaisted 2008, Section 1.2.2). An interpretation is called a *model* of a *theory* – a (possibly infinite) set of sentences – when it satisfies every sentence in this theory. Some of the examples considered contain integers, function constants such as $+$ and comparison predicate constants such as $\leq$ or $>$ (used utilizing infix notation common in arithmetic). We call these function and predicate constants *arithmetic*. In these examples, we assume that (i) the underlying signature contains a sort integer and (ii) interpretations of special kind are considered so that they interpret arithmetic function and predicate constants as customary in arithmetic (see, for example, treatment of the integer sort by Fandinno et al. (2020) and Lifschitz (2021)). If $\mathbf{d}$ is a tuple $d_1, \dots, d_n$ of domains elements of I then $\mathbf{d}^*$ stands for the tuple $d_1^*, \dots, d_n^*$ of their names. If $\mathbf{t}$ is a tuple $t_1, \dots, t_n$ of ground terms then $\mathbf{t}^I$ is the tuple $t_1^I, \dots, t_n^I$ of values assigned to them by I.

Here-and-there. The first-order logic of here-and-there, introduced by Pearce and Valverde (2004, 2005), forms a monotonic base for stable model semantic (Gelfond and Lifschitz 1988, 1991). The many-sorted case was recently studied by Fandinno and Lifschitz (2022). By $At(I)$ we denote the set of ground atoms of the form $p(\mathbf{d}^*)$ such that $I \models p(\mathbf{d}^*)$, where p is a predicate symbol and $\mathbf{d}$ is a tuple of elements of domains of I. An *HT-interpretation* of σ is a pair $\langle \mathcal{H}, I \rangle$, where I is an interpretation of σ, and $\mathcal{H}$ is a subset of $At(I)$. (In terms of Kripke models with two worlds, $\mathcal{H}$ describes the predicates in the here-world and I captures the there-world). The satisfaction relation $\models_{\text{ht}}$ between HT-interpretation $\langle \mathcal{H}, I \rangle$ of σ and a sentence F over σ^I is defined recursively:

- $\langle \mathcal{H}, I \rangle \models_{\text{ht}} p(\mathbf{t})$, if $p((\mathbf{t}^I)^*) \in \mathcal{H}$
- $\langle \mathcal{H}, I \rangle \models_{\text{ht}} t_1 = t_2$ if $t_1^I = t_2^I$;
- $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F \wedge G$ if $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F$ and $\langle \mathcal{H}, I \rangle \models_{\text{ht}} G$;
- $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F \vee G$ if $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F$ or $\langle \mathcal{H}, I \rangle \models_{\text{ht}} G$;
- $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F \rightarrow G$ if (i) $\langle \mathcal{H}, I \rangle \not\models_{\text{ht}} F$ or $\langle \mathcal{H}, I \rangle \models_{\text{ht}} G$, and (ii) $I \models F \rightarrow G$;
- $\langle \mathcal{H}, I \rangle \models_{\text{ht}} \forall X\, F(X)$ if $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F(d^*)$ for each $d \in |I|^s$, where s is the sort of X;
- $\langle \mathcal{H}, I \rangle \models_{\text{ht}} \exists X\, F(X)$ if $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F(d^*)$ for some $d \in |I|^s$, where s is the sort of X.

If $\langle \mathcal{H}, I \rangle \models_{\text{ht}} F$ holds, we say that $\langle \mathcal{H}, I \rangle$ *satisfies* F and that $\langle \mathcal{H}, I \rangle$ is an *HT-model* of F. If two formulas have the same HT-models then we say that they are *HT-equivalent*.

It is easy to see that $(At(I), I)$ is an HT-model of a sentence F whenever I is a model of F. About a model I of a theory Γ, we say it is *stable* if, for every proper subset $\mathcal{H}$ of $At(I)$, HT-interpretation $\langle \mathcal{H}, I \rangle$ does not satisfy Γ. In application to theories of a single sort, this definition is equivalent to the original definition by Pearce and Valverde (2004, 2005). In addition, if the theory is finite, then this definition of a stable model is also equivalent to the definition of such model in terms of the operator SM (Ferraris, Lee, and Lifschitz 2007, 2011), when all predicate are considered to be intensional. We generalize the distinction between intensional and extensional predicates in the next section.

Disjunctive logic programs. A *disjunctive rule* is a formula of the form $Head \leftarrow Body$ where $Head$ is a list of atomic formulas and $Body$ is a list of *literals*, that is, atomic formulas possibly preceded by one or two occurrences of negation *not*. A *disjunctive program* is a (possibly infinite) set of disjunctive rules. We identify each rule $Head \leftarrow Body$ with the universal closure of the formula $B \rightarrow H$, where B is the conjunction of all literals in $Body$ after replacing *not* by $\neg$; and H is the disjunction of all atomic formulas in $Head$. We often write rules as formulas $B \rightarrow H$ omitting the reference to the universal closure. For instance, rule (1) is understood as the universal closure of formula

$$on(B, L, T) \wedge \neg non(B, L, T) \rightarrow on(B, L, T + 1). \qquad (4)$$

We may also write rule (1) in this form. We assume that variable T in rules (1-3) is of sort integer. In examples, we assume that any variable that is used as argument of an arithmetic function is of this sort. Under these assumptions, the definition of stable models of sets of sentences stated earlier also applies to (possible infinite) disjunctive logic programs with arithmetic. Note that these assumptions are enough to showcase all examples in this paper. For a more general characterization of how sorts are assigned to variables, we refer to Fandinno et al. (2020) and Lifschitz (2021).

Stable Models with Intensionality Statements

The distinction between *extensional* and *intensional* predicates (Ferraris, Lee, and Lifschitz 2011) describes the inherent meaning of a group of rules in a precise way; and it is similar to the distinction between input and non-input atoms by Oikarinen and Janhunen (2008). It is sometimes convenient to treat words *extensional* and *intensional* as synonymous to words *input* and *defined*, respectively. Intuitively, the meaning of input predicate symbols is not fixed within the considered group of rules; these rules may constrain the interpretations of such predicate symbols but they do not "define" them. To the contrary, intensional predicate symbols can be viewed as defined by the group. For instance, consider rules (1-3) that we denote as Π_{block}. Let us elaborate on the meaning of Π_{block}. Intuitively, Π_{block} captures the behavior of the property *on*/3 that changes when relevant movement occurs and otherwise it obeys the commonsense law of inertia (i.e., that things stay as they are unless forced to change). To obtain such a reading of Π_{block} predicates *move*/3 and *location*/3 should be declared as extensional. On the other hand, predicates *on*/3 and *non*/3 should be declared as intensional as their behavior is defined by rules in Π_{block}. It looks as such separation of predicate symbols allows us to identify Π_{block} with its intuitive meaning and yet the devil is in the details. The reading that we have stated fails to mention the special case of the initial situation. At the initial state, i.e., when $T = 0$, we do not assume that rules in Π_{block} define *on*/3, rather that these values are specified elsewhere. Thus, we would like to declare that *on*/3 is extensional in the initial situation (when $T = 0$) and intensional in all other situations (when $T \neq 0$). However, the granularity desired for this example is not currently possible because the distinction between extensional and intensional by Ferraris, Lee, and Lifschitz (2011) is made at the predicate level, disregarding the context provided by their arguments. We address this issue here.

We generalize the idea of distinguish between extensional and intensional predicate symbols by allowing the possibility to declare circumstances under which a predicate symbol is considered to be intensional (if these circumstances are not the case the predicate symbol is considered extensional). This is achieved by associating a first-order formula with each predicate constant. For instance, in Π_{block} associating predicate symbol *on*/3 with the formula $T \neq 0$ will result in proper treatment of its rules respecting not only their core meaning but also the corner case of the initial situation.

Formally, we identify each predicate symbol p/n with an atom of the form $p(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ are pairwise distinct variables of appropriate sorts. An *intensionality statement* λ over a signature σ is a function mapping each predicate symbol p/n in σ to a formula $F(X_1, \ldots, X_n)$ such that

- $X_1,\ldots,X_n$ are the only free variables in $F(X_1,\ldots,X_n)$,
- every predicate symbol q/m in F satisfies $\lambda(q/m) \equiv \bot$.

We abbreviate $\lambda(p/n)(X_1,\ldots,X_n)$ as $\lambda^p(X_1,\ldots,X_n)$ when arity n is clear from the context. We say that a predicate symbol is (purely) *intensional*/*extensional* when it is associated with a valid/unsatisfiable formula, respectively.

Consider, for instance, predicate symbol *on*/3 and let $\lambda^{on}(B, L, T)$ be formula $T \neq 0$. Intuitively, this intensionality statement states that all ground atoms formed by predicate constant p with the third argument different from 0 are intensional; if the third argument is 0, then they are extensional. The complete intensionality statement, which we call β, for our motivating example follows:

$$\begin{aligned} \beta^{on}(B,L,T) &\text{ is } T \neq 0 \\ \beta^{non}(B,L,T) &\text{ is } \top \\ \beta^{move}(B,L,T) &\text{ is } \bot \\ \beta^{location}(L) &\text{ is } \bot \\ \beta^{\prec}(X_1,X_2) &\text{ is } \bot \quad \text{for any comparison } \prec \end{aligned} \tag{5}$$

It states that predicate *move* and all comparison symbols are purely extensional, *non* is purely intensional and *on* is intensional under all circumstances except the initial situation.

Let σ be a signature and λ be an intensionality statement over σ. By $EM(\lambda)$, we denote the set consisting of a sentence of the form

$$\forall \mathbf{X}\,(\neg\lambda^p(\mathbf{X}) \rightarrow p(\mathbf{X}) \vee \neg p(\mathbf{X}))$$

for every predicate symbol p/n in σ, where $\mathbf{X}$ is a tuple of variables of the appropriate length and sort (in the sequel we adopt this convention and use $\mathbf{X}$ to denote tuples of variables). For a theory Γ over σ, we say that an interpretation I is λ-*stable* if it is a stable model of $\Gamma \cup EM(\lambda)$. Note that, if every predicate symbol is intensional in λ, then the λ-stable models coincide with the stable models.

For example, let F be the rule

$$p(X,1) \rightarrow p(X,2) \tag{6}$$

and let $\lambda^p(X_1, X_2)$ be formula $X_2 = 2$ (we assume both arguments of $p/2$ be of sort integer). The intensionality statement formula $EM(\lambda)$ is the universal closure of

$$\neg(X_2 = 2) \rightarrow p(X_1, X_2) \vee \neg p(X_1, X_2).$$

The following sets of ground atoms correspond to the four λ-stable models of F with domain $\{1, 2\}$

$$\emptyset \qquad \{p(1,1),\, p(1,2)\} \qquad \{p(2,1),\, p(2,2)\}$$
$$\{p(1,1),\, p(2,1),\, p(1,2),\, p(2,2)\},$$

where we list the atoms that are true in these models (as customary in logic programming). As a more elaborated example, we can see that an interpretation I that satisfies

$$\begin{aligned} location^I = \{l_1, l_2\} \qquad move^I &= \{(b, l_2, 0)\} \\ on^I &= \{(b, l_1, 0)\} \cup \{(b, l_2, t') \mid t' > 0\} \\ non^I &= \{(b, l_2, 0)\} \cup \{(b, l_1, t') \mid t' > 0\} \end{aligned}$$

is a β-stable model of Π_{block}.

Strong Equivalence with Intensionality Statements

Informally, two programs are strongly equivalent if one can replace the other in any context without changing the stable models of the program (Lifschitz, Pearce, and Valverde 2001, 2007). Let Γ_1 and Γ_2 be theories over the same signature σ. We say that sets Γ_1 and Γ_2 are *strongly equivalent with respect to an intensionality statement* λ (or, simply λ-*strongly equivalent*) if $\Gamma_1 \cup \Delta$ and $\Gamma_2 \cup \Delta$ have the same λ-stable models for any theory Δ over σ.

By definition of λ-stable model, we immediately obtain the following sufficient condition for λ-strong equivalence.

Proposition 1 *If $\Gamma_1 \cup EM(\lambda)$ and $\Gamma_2 \cup EM(\lambda)$ have the same HT-models, then Γ_1 and Γ_2 are λ-strongly equivalent.*

Proposition 1 allows us to conclude that we can replace, without changing the β-stable models, rule (1) in Π_{block} by rules

$$\begin{aligned} on(B, L, T+1) \leftarrow\ & on(B, L, T), && (7)\\ & not\ non(B, L, T+1),\ T < t \\ on(B, L, T+1) \leftarrow\ & on(B, L, T), && (8)\\ & not\ non(B, L, T+1),\ T \geq t \end{aligned}$$

where t is any positive integer. Note that (7) and (8) are obtained from rule (1) by respectively adding $T < t$ and $T \geq t$ to its body. In fact (1) is λ-strong equivalent to to $\{(7), (8)\}$ for every λ where comparison symbols are extensional. Analogous rewritings can be performed on the remaining rules of Π_{block}. By $\Pi^{<}_{block}$ we denote the program obtained from Π_{block} by adding $T < t$ to the body of rules (1) and (2), and adding $T \leq t$ to the body of rule (3). By $\Pi^{>}_{block}$ we denote the program obtained from Π_{block} by adding $T \geq t$ to the body of rules (1) and (2), and adding $T > t$ to the body of rule (3). Programs Π_{block} and $\Pi^{<}_{block} \cup \Pi^{>}_{block}$ are β-strongly equivalent. We use program $\Pi^{<}_{block} \cup \Pi^{>}_{block}$ to showcase our Splitting method in the following section. In particular, the proposed method allows us to split this program into its two subcomponents $\Pi^{<}_{block}$ and $\Pi^{>}_{block}$.

Splitting Theorem with Intensionality Statements

To make the key results of this work comprehensible, we describe our Splitting method using two settings. First, we introduce the core ideas of the method in a restricted context, where considered theories are composed of disjunctive rules. Then, we generalize the method to arbitrary theories.

Splitting Disjunctive Programs

The statement of the proposed Splitting Theorem refers to the concept of the predicate dependency graph defined below. This definition, requires the notion of a *partition* of an intensionality statement. Formally, given two intensionality statement, λ_1 and λ_2, over a signature σ by $\lambda_1 \sqcup \lambda_2$ we denote the intensionality statement defined as

$$(\lambda_1 \sqcup \lambda_2)^p(X_1, \ldots, X_n) = \lambda_1^p(X_1, \ldots, X_n) \vee \lambda_2^p(X_1, \ldots, X_n).$$

for every predicate symbol p in the signature. By $\lambda_1 \sqcap \lambda_2$, we denote the intensionality statements defined as

$$(\lambda_1 \sqcap \lambda_2)^p(X_1, \ldots, X_n) = \lambda_1^p(X_1, \ldots, X_n) \wedge \lambda_2^p(X_1, \ldots, X_n).$$

for every predicate symbol p. Intensionality statements λ_1 and λ_2 are *equivalent* if the universal closure of formula

$$\lambda_1^p(X_1, \ldots, X_n) \leftrightarrow \lambda_2^p(X_1, \ldots, X_n)$$

is logically valid for every predicate symbol p. If λ_1 and λ_2 are equivalent, then we write $\lambda_1 \equiv \lambda_2$. By $\lambda_\top$ and $\lambda_\bot$ we denote the intensionality statements where all predicate symbols are intensional and extensional, respectively. Intensionality statements λ_1 and λ_2 are called *disjoint* if $\lambda_1 \sqcap \lambda_2 \equiv \lambda_\bot$. A set of intensionality statements $\{\lambda_1, \ldots, \lambda_k\}$ is called a *partition* of some intensionality statement λ if $\lambda_1 \sqcup \ldots \sqcup \lambda_k \equiv \lambda$ and $\lambda_i \sqcap \lambda_j \equiv \lambda_\bot$ for all $i \neq j$. For instance, we can see that $\{\beta_1, \beta_2\}$ is a partition of intensionality statement β, where β_1 and β_2 are obtained from β by modifying the formulas associated with predicate constants on and non as follows:

$$\begin{aligned} \beta_1^{on}(B, L, T) &\ \text{is}\ T \neq 0 \wedge T \leq t \\ \beta_1^{non}(B, L, T) &\ \text{is}\ T \leq t \\ \beta_2^{on}(B, L, T) &\ \text{is}\ T > t \\ \beta_2^{non}(B, L, T) &\ \text{is}\ T > t. \end{aligned} \quad (9)$$

An occurrence of a predicate constant, or any other subexpression, in a formula is called *negated* if it occurs in the scope of negation (i.e in the antecedent of any implication of the form $F \rightarrow \bot$); *nonnegated* otherwise. As an example, predicate constant on occurs nonnegated in the antecedent of (4), while non does not. Given a program Π (understood as a set of sentences) and a partition Λ of some intensionality statement λ, the *(directed) graph of dependencies with respect to* Λ, denoted $\mathfrak{G}_\Lambda(\Pi)$, is defined as follows:

- Its vertices are pairs (p, λ_i) such that p is a predicate symbol occurring in Π, $\lambda_i \in \Lambda$ is an intensionality statement and $\exists \mathbf{X}\ \lambda_i^p(\mathbf{X})$ is satisfiable.
- It has an edge from (p, λ_i) to (q, λ_j) when for some rule $B \rightarrow H$ of Π,
  - $p(\mathbf{t})$ occurs in H, and
  - there is a nonnegated occurrence of $q(\mathbf{r})$ in B and
  - the following sentence is satisfiable

$$\exists \mathbf{X} \left(B \wedge p(\mathbf{t}) \wedge \lambda_i^p(\mathbf{t}) \wedge \lambda_j^q(\mathbf{r}) \right), \quad (10)$$

  where $\mathbf{X}$ are the free variables in $B \rightarrow H$.

We say that a partition $\Lambda = \{\lambda_1, \ldots, \lambda_k\}$ is *separable* (on $\mathfrak{G}_\Lambda(\Pi)$) when

every infinite walk $v_1, v_2, \ldots$ of $\mathfrak{G}_\Lambda(\Pi)$ visits at most one λ_i infinitely many times, that is, there is some $i \in \{1, \ldots, k\}$ s.t $\{l \mid v_l = (p, \lambda_j)\}$ is finite for all $j \neq i$.

Note that, if (the signature of) Π is finite, every possible infinite walk must contain a cycle and, thus, this condition is equivalent to saying that every cycle is confined to one component of the partition. In general, checking cycles is not enough for infinite programs, even when these programs contain no variables (Harrison and Lifschitz 2016).

Continuing with our running example, let $\Pi'_{block} = \Pi^{<}_{block} \cup \Pi^{>}_{block}$ and $\Lambda_{block} = \{\beta_1, \beta_2\}$ be partition of β. Then, graph $\mathfrak{G}_{\Lambda_{block}}(\Pi'_{block})$ consists of vertices

$$(on, \beta_1), \quad (on, \beta_2), \quad (non, \beta_1), \quad (non, \beta_2)$$

and contains five edges: one leading from (on, β_1) to itself, another one leading from (on, β_2) to itself, one leading from (on, β_2) to (on, β_1) — induced by rule (8) — and two leading from (non, β_i) to (on, β_i) with $1 \leq i \leq 2$. Note that there is no edge from (on, β_1) to (on, β_2). For instance, we can see that rules (7) and (8) do not induce such edge because the sentence of the form (10) corresponding to these rules contain conjuncts $T + 1 \leq t$ and $T > t$, which make them unsatisfiable. Conjunct $T + 1 \leq t$ is obtained from $\beta_1^{on}(B, L, T{+}1)$ corresponding to the head occurrence of $on/3$. The other conjunct is obtained from $\beta_2^{on}(B, L, T)$ corresponding to its body occurrence. Therefore, partition Λ_{block} is separable.

A program Π is *negative* on some intensionality statement λ if, for every rule $B \rightarrow H$ of Π and every atom $p(\mathbf{t})$ occurring in H, the following sentence is unsatisfiable

$$\exists \mathbf{X}\, (B \wedge p(\mathbf{t}) \wedge \lambda^p(\mathbf{t})),$$

where $\mathbf{X}$ are the free variables in $B \rightarrow H$. For instance, programs $\Pi^{<}_{block}$ and $\Pi^{>}_{block}$ are negative on β_2 and β_1, respectively. We provide a part of an argument for the claim that $\Pi^{<}_{block}$ is negative on β_2. In particular, we argue that rule (7) is negative on β_2. This rule has $on(B, L, T + 1)$ in its head and $T < t$ in its body. It is sufficient to show that a formula of the form

$$\exists BLT\, (F \wedge T < t \wedge \beta_2^{on}(B, L, T + 1)),$$

is unsatisfiable. Recall that $\beta_2^{on}(B, L, T + 1)$ is $T + 1 > t$.

Theorem 1 (Splitting disjunctive programs) *Let* $\Pi = \Pi_1 \cup \ldots \cup \Pi_n$ *be a disjunctive program and* $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ *be a partition of* λ *such that*

- Λ *is separable on* $\mathfrak{G}_\Lambda(\Pi)$*; and*
- *each* Π_i *is negative on* λ_j *for all* $j \neq i$*.*

Then, for any interpretation I, the following two statements are equivalent

- *I is a λ-stable model of Π, and*
- *I is a λ_i-stable model of Π_i for all $1 \leq i \leq n$.*

Recall that partition Λ_{block} is separable on $\mathfrak{G}_{\Lambda_{block}}(\Pi'_{block})$ and that programs $\Pi^{<}_{block}$ and $\Pi^{>}_{block}$ are negative on β_2 and β_1, respectively. Therefore, the theorem on Splitting disjunctive programs allows us to conclude that the β-stable models of Π'_{block} are those interpretations that are at the same time β_1-stable models of $\Pi^{<}_{block}$ and β_2-stable models of $\Pi^{>}_{block}$. As a final note, recall that Π_{block} and Π'_{block} are β-strongly equivalent and, thus, they have the same β-stable models. Hence, the statement that we can split program Π_{block} into programs $\Pi^{<}_{block}$ and $\Pi^{>}_{block}$ is a consequence of these two facts stemming from presented Theorem 1 and Proposition 1, respectively.

Splitting Arbitrary Theories

We now generalize Theorem 1 on Splitting disjunctive programs to the case of arbitrary sets of sentences. We start by motivating this generalization and then proceed to its formalization. Complex formulas not fitting into the syntax of disjunctive rules naturally appear as a result of translating some common constructs of logic programs into first-order formulas. These constructs include basic arithmetic operations such as division, which are part of the ASP-Core-2 (Calimeri et al. 2012), and more advanced features such as intervals, or conditional literals Intervals and conditional literals are part of the language of the solver `clingo` (Gebser et al. 2015) and are commonly used in practice. For instance, the following rule contains a conditional literal $holds(B) : body(R, B)$ in its body:

$$holds(H) \leftarrow head(R, H),\ holds(B) : body(R, B). \quad (11)$$

Hansen and Lierler (2022) showed that this rule can be understood as an abbreviation for the following first-order sentence containing a nested universal quantifier and a nested implication

$$\forall RH\big(head(R, H) \wedge \\ \forall B\big(body(R, B) \rightarrow holds(B)\big) \rightarrow holds(H)\big). \quad (12)$$

Rules of this kind are common in what is called meta-programming (Kaminski et al. 2021), where reification of constructs is utilized to build ASP-based reasoning engines that *may* go beyond ASP paradigm itself. Kaminski et al. (2021) presented multiple examples of meta-programming. Here, we use rule (11) to showcase a simple use of this technique. Indeed, it can be used to express the meaning of a *definite* rule – a disjunctive rule whose $Head$ consists of single atomic formula and $Body$ contains no occurrences of negation not – encoded as a set of facts. For example, definite program

$$\begin{array}{l} a \leftarrow b \\ b \leftarrow c \end{array} \quad (13)$$

can be encoded by the following facts complemented with rule (11)

$$head(r1, a) \quad body(r1, b) \quad (14)$$
$$head(r2, b) \quad body(r2, c). \quad (15)$$

Clearly, we can split the program listed in (13) into two subprograms, each consisting of a single rule. We may expect the possibility of splitting its respective meta encoding as well. Consider sentences

$$\forall X\big(head(r1, X) \wedge \\ \forall W\big(body(r1, W) \rightarrow holds(W)\big) \rightarrow holds(X)\big) \quad (16)$$

$$\forall X\big(head(r2, X) \wedge \\ \forall W\big(body(r2, W) \rightarrow holds(W)\big) \rightarrow holds(X)\big) \quad (17)$$

corresponding to a "partially instantiated" portion of meta encoding of program (13). Ideally, the program corresponding to sentences (14-17) would be "identified" with subprograms (14,16) and (15,17) by splitting. Yet, Theorem 1 cannot support such a claim as (16) and (17) are not disjunctive rules. This claim is also not supported by other Splitting Theorems in the literature (Ferraris et al. 2009; Harrison and Lifschitz 2016) due to the "positive nonnegated" dependency induced by these rules in the predicate $hold/1$ (we define the concept of positive nonnegated dependency below).

Formalization The first thing to note is that one direction of the Splitting Theorem always holds without need to inspect the dependency graph.

Proposition 2 *Let* $\Gamma = \Gamma_1 \cup \ldots \cup \Gamma_n$ *be a theory and let* $\Lambda = \{\gamma_1, \ldots, \gamma_n\}$ *be a partition of* λ*. If* I *is a* λ*-stable model of* Γ*, then* I *is a* γ_i*-stable model of* Γ_i *for all* $1 \leq i \leq n$*.*

For the other direction, we have to generalize the notions of separability and of being negative to the case of arbitrary sentences. Similar to the Splitting Theorem by Ferraris et al. (2009), this generalization relies on the notions of *strictly positive*, *positive nonnegated*, and *negative nonnegated* occurrence of an expression. An occurrence of an expressions is called *positive* if the number of implications containing that occurrence in the antecedent is even; and *strictly positive* if that number is 0. It is called *negative* if that number is odd. As in the case of logic programs, an occurrence of an expressions is called *negated* if it belongs to a subformula of the form $\neg F$ (that is, $F \rightarrow \bot$); and *nonnegated* otherwise. A *rule* is a strictly positive occurrence of a formula of the form $B \rightarrow H$. Therefore, the rules of a set of disjunctive rules are exactly all its disjunctive rules, but this definition also covers arbitrary nested rules. For instance, sentence $(c \rightarrow (a \rightarrow b)) \vee d)$ contains two occurrences of rules, namely, $a \rightarrow b$ and $c \rightarrow (a \rightarrow b)$.

To extend Theorem 1 to arbitrary formulas, we use these notions to make the construction of a counterpart of sentence (10) recursive over the formula. In addition, we ought to incorporate in this construction a *context* that carries information about the rest of the program. To observe the need for this context, note that whether rules (16) and (17) can be separated into differ subprograms depends on the extension of $head/2$ and $body/2$. For instance, adding the fact $body(r2, a)$ to our program creates a dependency that cannot be broken. In fact, this new program has the same meaning as a non-splittable program consisting of rules $a \leftarrow b$ and $b \leftarrow c, a$.

Given a formula F with free variables $\mathbf{X}$ and a theory Ψ, by F^{Ψ} we denote formula F itself if $\Psi \cup \{\exists \mathbf{X}\, F\}$ is satisfiable; and $\bot$ otherwise. Let now $p(\mathbf{t})$ be a strictly positive (resp. positive nonnegated or negative nonnegated) "distinguished" occurrence of predicate p in F and $\mathbf{Y}$ a list of variables not occurring in F of the same length and sorts as $\mathbf{t}$. We recursively build formula $Sp_{\Psi}(F)$ (resp. $Pnn_{\Psi}(F)$ and $Nnn_{\Psi}(F)$) for this occurrence as described next; the construction of these three formulas only differs in the case of implication, so we use the metavariable $Fun_{\Psi}(F)$ for the common cases:

- If $\Psi \cup \{\exists \mathbf{X}\, F\}$ is unsatisfiable, then $Fun_{\Psi}(F) = \bot$.
- If $\Psi \cup \{\exists \mathbf{X}\, F\}$ is satisfiable and the distinguished occurrence $p(\mathbf{t})$ does not occur in F, then $Fun_{\Psi}(F) = F$.

Otherwise,

- $Fun_{\Psi}(p(\mathbf{t})) = p(\mathbf{t}) \wedge \mathbf{Y} = \mathbf{t}$;
- $Fun_{\Psi}(F_1 \wedge F_2) = Fun_{\Psi}(F_1) \wedge Fun_{\Psi}(F_2)$;
- $Fun_{\Psi}(F_1 \vee F_2) = Fun_{\Psi}(F_i)$ with F_i containing the occurrence (i is 1 or 2);
- $Fun_{\Psi}(\forall X F) = Fun_{\Psi}(\exists X F) = \exists X \big(Fun_{\Psi}(F)\big)$;
- $Sp_{\Psi}(F_1 \rightarrow F_2) = F_1^{\Psi} \wedge Sp_{\Psi}(F_2)$;
- $Pnn_{\Psi}(F_1 \rightarrow F_2) =$
$$\begin{cases} Nnn_{\Psi}(F_1) & \text{if } F_1 \text{ contains the occurrence} \\ F_1^{\Psi} \wedge Pnn_{\Psi}(F_2) & \text{otherwise} \end{cases};$$
- $Nnn_{\Psi}(F_1 \rightarrow F_2) =$
$$\begin{cases} Pnn_{\Psi}(F_1) & \text{if } F_1 \text{ contains the occurrence} \\ F_1^{\Psi} \wedge Nnn_{\Psi}(F_2) & \text{otherwise} \end{cases}.$$

Note that $Pnn_{\Psi}(F)$ and $Nnn_{\Psi}(F)$ are mutually recursive due to the case of implication. Examples illustrating the construction of these formulas follow the next definition.

Given theories Γ and Ψ and a partition Λ of some intensionality statement λ, the *(directed) graph of positive dependencies with respect to* Λ *and under context* Ψ, denoted $\mathfrak{G}_{\Lambda,\Psi}(\Gamma)$, is defined as follows:

- Its vertices are pairs (p, λ_i) such that p is a predicate symbol, $\lambda_i \in \Lambda$ is an intensionality statement and theory $\Psi \cup \{\exists \mathbf{X}\, \lambda_i^p(\mathbf{X})\}$ is satisfiable.
- It has an edge from (p, λ_i) to (q, λ_j) when for some rule $B \rightarrow H$ of Γ, the following conditions hold
  - there is a strictly positive occurrence $p(\mathbf{t})$ in H,
  - there is a positive nonnegated occurrence of $q(\mathbf{r})$ in B,
  - the theory below is satisfiable,
  $$\Psi \cup \big\{\exists \mathbf{XYZ}\big(Pnn_{\Psi}(B) \wedge Sp_{\Psi}(H) \wedge \lambda_j^q(\mathbf{Y}) \wedge \lambda_i^p(\mathbf{Z})\big)\big\},$$
  where $\mathbf{X}$ are the free variables in $B \rightarrow H$, and $\mathbf{Y}$ and $\mathbf{Z}$ respectively are the free variables in formulas $Pnn_{\Psi}(B)$ and $Sp_{\Psi}(H)$ that are not in $\mathbf{X}$; in the construction of $Sp_{\Psi}(H)$ and $Pnn_{\Psi}(B)$ we consider occurrences $p(\mathbf{t})$ in H and $q(\mathbf{r})$ in B, respectively.

Take sentence $F_1 = q \vee (r \wedge p)$ and rule $F_1 \rightarrow p$. Note that p occurs positively nonnegated in F_1. We consider this occurrence p when constructing formulas $Pnn_{(\cdot)}(F_1)$ below. Assume that p is intensional in λ (i.e., $\lambda^p \equiv \top$) and that Λ is a partition of λ. On the one hand, if we consider the empty context, then $Pnn_{\emptyset}(F_1)$ is $r \wedge p$ and the dependency graph of $F_1 \rightarrow p$ with respect to Λ and the empty context contains a reflexive edge on vertex (p, λ). On the other hand, if we take context $\Psi_1 = \{\neg r\}$, then $Pnn_{\Psi_1}(F_1)$ is $\bot$ and the dependency graph with respect to Λ and Ψ_1 is empty. This example shows that, even on propositional formulas, the use of context leads to less dependencies than previous approaches such as (Ferraris et al. 2009; Harrison and Lifschitz 2016). This leads to the fact that the Splitting Theorem introduced in the sequel is applicable to more theories as it relies on this new notion of dependencies. This difference may appear even using the empty context. Take a theory consisting of formula $\big(q \vee (\bot \wedge p)\big) \rightarrow p$. It turns out that p never contributes any edge in our approach under any context. For the same theory, p always forms a dependency in the earlier papers. Consider now a small variation. Let p be a unary predicate and let $F_2 = q \vee \exists X\big(r \wedge p(X)\big)$. Note that $Pnn_{\emptyset}(F_2) = \exists X(r \wedge p(X) \wedge Y = X)$ and $Pnn_{\Psi_1}(F_2) = \bot$, where we construct these formulas for occurrence $p(X)$. Under the empty context, analogously to the previous example, the dependency graph of $F_2 \rightarrow p(1)$

contains a reflexive edge on vertex (p, λ). Under context Ψ_1, the dependency graph is empty.

Construction of $Sp_\Psi(F)$, $Pnn_\Psi(F)$ and $Nnn_\Psi(F)$ reference a "distinguished" occurrence, as there may be multiple occurrences of the same atomic formula. For instance, there are two occurrences of formula $p(X)$ in sentence F_3 defined as $\exists X(X = a \wedge p(X)) \vee (X = b \wedge p(X))$; for this sentence $Pnn_\emptyset(F_3)$ is $\exists X(X = a \wedge p(X) \wedge Y = X)$, when the first occurrence of $p(X)$ is considered; and $\exists X(X = b \wedge p(X) \wedge Y = X)$, when the second one is considered. Take Ψ_2 to be the theory containing sentence $\neg p(b)$ only, then $Pnn_{\Psi_2}(F_3)$ remains unmodified for the first occurrence of p, but becomes $\bot$ for the second one. As a consequence, in a context where $p(b)$ is false, the first occurrence may generate an edge in the dependency graph, while the second occurrence does not generate any. Note that in all considered examples so far, $Sp_{(\cdot)}(\cdot)$ is the same as $Pnn_{(\cdot)}(\cdot)$ due to the lack of implications.

We now explore the meta encoding example introduced earlier. Let B denote the antecedent of rule (16) and H denote its consequent $holds(X)$. Under the empty context, $Pnn_\emptyset(B)$ for the only occurrence of $holds/1$ in B is

$$head(r1, X) \wedge \exists W\big(body(r1, W) \wedge holds(W) \wedge Y = W\big);$$

and $Sp_\emptyset(H)$ for the only occurrence of $holds/1$ in H is $holds(X) \wedge X = Z$. Let γ_1 and γ_2 be intensionality statements where $head$ and $body$ are extensional and

$$\gamma_1^{holds}(X_1) \text{ is } X_1 = a \qquad \gamma_2^{holds}(X_1) \text{ is } X_1 = b.$$

Then, the singleton theory with the existential closure of

$$Pnn_\emptyset(B) \wedge Sp_\emptyset(H) \wedge \gamma_j^{holds}(Y) \wedge \gamma_i^{holds}(Z) \qquad (18)$$

is satisfiable for any $i, j \in \{1, 2\}$. For a theory containing rule (16), the graph of positive dependencies with respect to $\{\gamma_1, \gamma_2\}$ and under the empty context contains an edge from vertex $(holds, \gamma_1)$ to vertex $(holds, \gamma_2)$, and vice-versa; together with reflexive edges in both vertices. Let Ψ_3 be the theory consisting of the universal closures of formulas

$$head(r1, X) \leftrightarrow X = a \qquad head(r2, X) \leftrightarrow X = b$$
$$body(r1, X) \leftrightarrow X = b \qquad body(r2, X) \leftrightarrow X = c.$$

Then, $Sp_{\Psi_3}(H) = Sp_\emptyset(H)$ and $Pnn_{\Psi_3}(B) = Pnn_\emptyset(B)$. The union of Ψ_3 and the singleton theory consisting of the existential closure of formula (18) is satisfiable only with $i = 1$ and $j = 2$. Therefore, under this context, the graph of positive dependencies from the previous example contains an edge from $(holds, \gamma_1)$ to $(holds, \gamma_2)$ but not vice-versa. There are no reflexive edges either. In fact, under this context the positive dependency graph of (16-17) only contains this edge.

Let us now generalize the notion of being a negative occurrence to the case of arbitrary theories. We say that a theory Γ is *Ψ-negative* on some intensionality statement λ if, for every rule $B \to H$ of Γ and every strictly positive occurrence $p(\mathbf{t})$ in H, the following theory is unsatisfiable $\Psi \cup \{\exists \mathbf{XY}\,(B \wedge Sp_\Psi(H) \wedge \lambda^p(\mathbf{Y}))\}$, where $\mathbf{X}$ consists of the free variables in $B \to H$; $\mathbf{Y}$ consists of the free variables in $Sp_\Psi(H)$ that do not occur in $\mathbf{X}$; and we consider occurrence $p(\mathbf{t})$ in H in the construction of $Sp_\Psi(H)$. We also say that theory Ψ is an *λ-approximator* of Γ if all the λ-stable models of Γ are models of Ψ.

Theorem 2 (Splitting Theorem) *Let $\Gamma = \Gamma_1 \cup \ldots \cup \Gamma_n$, λ be an intensionality statement, and Ψ be an λ-approximator of Γ. Let $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ be a partition of λ such that*
- *Λ is separable on $\mathfrak{G}_{\Lambda,\Psi}(\Gamma)$; and*
- *each Γ_i is Ψ-negative on λ_j for all $j \neq i$.*

Then, the following two statements are equivalent
- *I is a λ-stable model of Π, and*
- *I is a model of Ψ and a λ_i-stable model of Π_i for all $i \in \{1, 2, 3\}$.*

Continuing with our meta encoding example, in addition to γ_1 and γ_2, we define γ_3 to be the intensionality statement where $head/2$, $body/2$ are intensional and $holds/1$ is extensional. Let γ be $\gamma_1 \sqcup \gamma_2 \sqcup \gamma_3$. It is easy to see that $\{\gamma_1, \gamma_2, \gamma_3\}$ is a partition of γ. Let Γ_1 and Γ_2 be the singleton theories consisting of (16) and (17), respectively; let Γ_3 consist of facts in (14-15), and let Γ denote $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$. Partition $\Lambda = \{\gamma_1, \gamma_2, \gamma_3\}$ is separable on $\mathfrak{G}_{\Lambda,\Psi_3}(\Gamma)$. We also can see that Γ_i is negative on γ_j under context Ψ_3 for all $j \neq i$. Therefore, by Theorem 2, the γ-stable models of Γ are the models of Ψ_3 that are γ_i-stable models of Γ_i for all $i \in \{1, 2, 3\}$. The intent of context Ψ_3 is to carry information from one part of the theory into another. In our example all models of Ψ_3 are γ_3-stable models of Γ_3. Hence, we can simply say that the γ-stable models of Γ are the interpretations that are γ_i-stable models of Γ_i for all $i \in \{1, 2, 3\}$.

It is worth noting that the empty theory approximates any theory. When such theory is considered in the presented theorem, it more closely resembles the Splitting Theorem by Ferraris et al. (2009). In general, we can use the theory itself, its completion (Ferraris, Lee, and Lifschitz 2011; Fandinno and Lifschitz 2022) or the completion of a part of it as a "more precise" approximator.

Conclusions

The concept of intensionality statements introduced here provides us with a new granularity on considering semantics of logic programs and its subcomponents. It also paves a way to the refinement of earlier versions of the Spltting method. We generalized the conditions under which this method can be applied to first-order theories and show how the resulting approach covers more programs commonly used in practice. This generalization comes at a price. The conditions of the Splitting Theorem by Ferraris et al. (2009) are syntactic, while our conditions rely on verification of semantic properties. In fact, deciding whether there is an edge in our dependency graph is, in general, undecidable. For instance, a program containing rules $p \leftarrow q, t = 0$ and $q \leftarrow p$, where t is a polynomial, can be split into subprograms each containing of one of these rules only if the Diophantine equation in the body has no solutions. However, we illustrated that for many practical problems the Splitting result of this paper is applicable. In the future, we will investigate the possibility to utilize first-order theorem provers for checking the required semantic conditions.

Acknowledgements

We are thankful to Vladimir Lifschitz, Torsten Schaub and the antonymous reviewers for their comments on earlier versions of this draft. The first author is partially supported by the Nebraska EPSCoR 95-3101-0060-402 grant.

References

Cabalar, P.; Fandinno, J.; and Lierler, Y. 2020. Modular Answer Set Programming as a Formal Specification Language. *Theory and Practice of Logic Programming*, 20(5): 767–782.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Ricca, F.; and Schaub, T. 2012. ASP-Core-2: Input language format.

Fandinno, J.; Hansen, Z.; and Lierler, Y. 2022. Axiomatization of Aggregates in Answer Set Programming. In *Proceedings of the Thirty-sixth National Conference on Artificial Intelligence (AAAI'22)*, 5634–5641. AAAI Press.

Fandinno, J.; and Lifschitz, V. 2022. Verification of Locally Tight Programs. Forthcoming.

Fandinno, J.; Lifschitz, V.; Lühne, P.; and Schaub, T. 2020. Verifying Tight Logic Programs with anthem and vampire. *Theory and Practice of Logic Programming*, 20(5): 735–750.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2007. A New Perspective on Stable Models. In Veloso, M., ed., *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, 372–379. AAAI/MIT Press.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence*, 175(1): 236–263.

Ferraris, P.; Lee, J.; Lifschitz, V.; and Palla, R. 2009. Symmetric Splitting in the General Theory of Stable Models. In Boutilier, C., ed., *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*, 797–803. AAAI/MIT Press.

Gebser, M.; Harrison, A.; Kaminski, R.; Lifschitz, V.; and Schaub, T. 2015. Abstract Gringo. *Theory and Practice of Logic Programming*, 15(4-5): 449–463.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, 19(1): 27–82.

Gelfond, M.; and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In Kowalski, R.; and Bowen, K., eds., *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, 1070–1080. MIT Press.

Gelfond, M.; and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9: 365–385.

Hansen, Z.; and Lierler, Y. 2022. Semantics for Conditional Literals via the SM Operator. In *LPNMR*, volume 13416 of *Lecture Notes in Computer Science*, 259–272. Springer.

Harrison, A.; and Lifschitz, V. 2016. Stable models for infinitary formulas with extensional atoms. *Theory and Practice of Logic Programming*, 16(5-6): 771–786.

Kaminski, R.; Romero, J.; Schaub, T.; and Wanko, P. 2021. How to Build Your Own ASP-based System?! *Theory and Practice of Logic Programming*, 1–63.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2): 39–54.

Lifschitz, V. 2008. What Is Answer Set Programming? In Fox, D.; and Gomes, C., eds., *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*, 1594–1597. AAAI Press.

Lifschitz, V. 2019. *Answer Set Programming*. Springer-Verlag.

Lifschitz, V. 2021. Transforming Gringo Rules into Formulas in a Natural Way. 421–434.

Lifschitz, V.; Morgenstern, L.; and Plaisted, D. 2008. Knowledge Representation and Classical Logic. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*, 3–88. Elsevier.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4): 526–541.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2007. A Characterization of Strong Equivalence for Logic Programs with Variables. In Baral, C.; Brewka, G.; and Schlipf, J., eds., *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*, 188–200. Springer-Verlag.

Lifschitz, V.; and Turner, H. 1994. Splitting a logic program. In *Proceedings of the Eleventh International Conference on Logic Programming*, 23–37. MIT Press.

Oikarinen, E.; and Janhunen, T. 2008. Achieving Compositionality of the Stable Model Semantics for smodels Programs. *Theory and Practice of Logic Programming*, 8(5-6): 717–761.

Pearce, D.; and Valverde, A. 2004. Towards a First Order Equilibrium Logic for Nonmonotonic Reasoning. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 147–160.

Pearce, D.; and Valverde, A. 2005. A First Order Nonmonotonic Extension of Constructive Logic. *Studia Logica*, 30(2-3): 321–346.

Truszczyński, M. 2012. Connecting First-Order ASP and the Logic FO(ID) through Reducts. In Erdem, E.; Lee, J.; Lierler, Y.; and Pearce, D., eds., *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, 543–559. Springer-Verlag.