Symmetric Splitting in the General Theory of Stable Models

Paolo Ferraris[1], Joohyung Lee[2], Vladimir Lifschitz[3], and Ravi Palla[2]

[1]Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
otto@cs.utexas.edu

[2]Computer Sci. and Eng.
Arizona State University
Tempe, AZ 85281, USA
{joolee, Ravi.Palla}@asu.edu

[3]Dept. of Computer Sciences
University of Texas at Austin
1 University Station C0500
Austin, TX 78705, USA
vl@cs.utexas.edu

Abstract

Splitting a logic program allows us to reduce the task of computing its stable models to similar tasks for smaller programs. This idea is extended here to the general theory of stable models that replaces traditional logic programs by arbitrary first-order sentences and distinguishes between intensional and extensional predicates. We discuss two kinds of splitting: a set of intensional predicates can be split into subsets, and a formula can be split into its conjunctive terms.

1 Introduction

This paper contributes to the theory of stable models, which is the mathematical basis of answer set programming (ASP) [Marek and Truszczyński, 1999; Niemelä, 1999; Lifschitz, 2008].

The splitting method [Lifschitz and Turner, 1994] can be sometimes used to reduce the task of computing the stable models of a logic program to similar tasks for smaller programs. Consider, for instance, the program

$$\begin{array}{l} p \leftarrow \mathit{not}\ q \\ q \leftarrow \mathit{not}\ p \\ r \leftarrow q. \end{array} \qquad (1)$$

Its stable models can be generated as follows. We concentrate first on the program consisting of the first two rules of (1):

$$\begin{array}{l} p \leftarrow \mathit{not}\ q \\ q \leftarrow \mathit{not}\ p. \end{array} \qquad (2)$$

In one of its stable models, p is true and q is false; call that model M_1. In the other, p is false and q is true; call it M_2. Then we look at the third rule of (1) and determine how this rule instructs us to "update" each of the models M_1, M_2 by assigning a truth value to r. Since its body q is false in M_1, the head r becomes false in this model as well. Since q is true in M_2, r becomes true in M_2. Thus the stable models of the given program are $\{p\}$ and $\{q, r\}$.

Splitting (1) into parts is possible because r does not occur in the rules that "define" p and q, so that p and q do not "depend" on r. Accordingly, determining the truth value of r can be postponed until the time when the truth values of p and q have been determined. Program (2), on the other hand, cannot be further split in the sense of [Lifschitz and Turner, 1994], because it defines p and q in terms of each other.

A more general version of splitting is described in [Oikarinen and Janhunen, 2008]. In the framework of that paper, some atoms occurring in a logic program can be designated as its "input" atoms, and that may affect the meaning of the program. For instance, the only stable model of the one-rule program

$$p \leftarrow \mathit{not}\ q \qquad (\text{no input atoms}) \qquad (3)$$

is M_1 (p is true, q is false). Intuitively, q is false in the stable model of (3) because its definition is empty: q does not occur in the head of any rule. But the program

$$p \leftarrow \mathit{not}\ q \qquad (q \text{ is input}) \qquad (4)$$

has both M_1 and M_2 as its stable models, according to Oikarinen and Janhunen. Intuitively, by saying that q is an input atom we assert that its truth value can be chosen arbitrarily; it is not supposed to be determined by the rules of the program.[1]

The version of the splitting theorem established in [Oikarinen and Janhunen, 2008] allows us to split (2) into two programs: (4) and

$$q \leftarrow \mathit{not}\ p \qquad (p \text{ is input}). \qquad (5)$$

That theorem shows that the stable models of (2) can be characterized as the common stable models of these two one-rule programs. Since M_1 is a stable model of each of the programs (4), (5), it is a stable model of (2) as well, and the same can be said about M_2. This example illustrates the power of "symmetric splitting" described in [Oikarinen and Janhunen, 2008] in comparison with "top-bottom splitting" proposed in [Lifschitz and Turner, 1994]. Symmetric splitting is extended to disjunctive programs in [Janhunen *et al.*, 2007].

The existing work on symmetric splitting does not cover, however, logic programs with variables, and it is not applicable to some types of rules that are frequently used in ASP, such as choice rules and cardinality constraints [Simons *et al.*, 2002]. In this paper, we extend symmetric splitting to the general theory of stable models in which traditional logic

[1]Saying that q is an input atom is similar to augmenting the program with the LPARSE choice rule $\{q\}$. (For a description of the language of LPARSE, see `http://www.tcs.hut.fi/Software/smodels/lparse.ps`.)

programs are replaced by arbitrary first-order sentences. This general approach to stable models is introduced in [Ferraris *et al.*, 2010][2] and reviewed in Section 2 below. It covers the programming constructs mentioned above, and other useful constructs, by treating them as abbreviations for first-order formulas. For instance, the LPARSE program Π consisting of three rules

```
{in(X)} :- vertex(X).
:- vertex(X;Y), in(X;Y),
   not edge(X,Y), not edge(Y,X), X!=Y.
:- {in(X):vertex(X)} n.
```

corresponds to the formula

$$\begin{array}{l}\forall x(\mathit{vertex}(x) \to (\mathit{in}(x) \vee \neg \mathit{in}(x)) \\ \wedge \forall xy \neg(\mathit{vertex}(x) \wedge \mathit{vertex}(y) \wedge \mathit{in}(x) \wedge \mathit{in}(y) \\ \qquad \wedge \neg \mathit{edge}(x,y) \wedge \neg \mathit{edge}(y,x) \wedge x \neq y) \\ \wedge \neg\neg \exists_{n+1} x(\mathit{in}(x) \wedge \mathit{vertex}(x)),\end{array} \tag{6}$$

where $\exists_n x F(x)$ is shorthand for

$$\exists x_1 \cdots x_n \left(\bigwedge_{1 \le i \le n} F(x_i) \wedge \bigwedge_{1 \le i < j \le n} x_i \neq x_j \right).$$

(Program Π describes large cliques in a graph, as discussed in Section 7.) Note that the first conjunctive term of (6) is logically valid, so that dropping it would not affect the class of models of (6). But the class of *stable* models of this formula, as defined in the next section, would be affected; in this sense, the first conjunctive term of (6) is essential. For the same reason, we do not drop $\neg\neg$ in front of $\exists_{n+1} x$.

The distinction between extensional and intensional predicates in [Ferraris *et al.*, 2010] is similar to the distinction between input and non-input atoms in [Oikarinen and Janhunen, 2008]. As discussed below, stable models of a first-order sentence F relative to a list $\mathbf{p}$ of intensional predicates are defined as models of the sentence obtained from F by a certain syntactic transformation, denoted by $\mathrm{SM}_{\mathbf{p}}$. (This transformation, like circumscription, makes a formula stronger, so that all stable models of F in the sense of that definition are indeed models of F.) For instance, if F is

$$(\neg q \to p) \wedge (\neg p \to q) \wedge (q \to r) \tag{7}$$

(which is program (1) written as a formula) then $\mathrm{SM}_{pqr}[F]$ can be equivalently rewritten as

$$(\neg q \leftrightarrow p) \wedge (q \leftrightarrow r).$$[3]

This formula has two models—the stable models of (1).

Our version of the splitting theorem asserts that under certain syntactic conditions

$$\mathrm{SM}_{\mathbf{pq}}[F \wedge G] \quad \text{is equivalent to} \quad \mathrm{SM}_{\mathbf{p}}[F] \wedge \mathrm{SM}_{\mathbf{q}}[G].$$

[2]That article is the journal version of the IJCAI-07 paper [Ferraris *et al.*, 2007]. In the conference paper, all predicates are implicitly assumed to be intensional.

[3]This is essentially the completion of (1) in the sense of [Clark, 1978]. In this example, the stable model semantics produces the same result as the completion semantics.

The example of splitting program (1) above corresponds to

$$\begin{array}{ll}(\neg q \to p) \wedge (\neg p \to q) & \text{as } F, \\ q \to r & \text{as } G, \\ pq & \text{as } \mathbf{p}, \\ r & \text{as } \mathbf{q}.\end{array}$$

To split program (2), we take

$$\begin{array}{ll}\neg q \to p & \text{as } F, \\ \neg p \to q & \text{as } G, \\ p & \text{as } \mathbf{p}, \\ q & \text{as } \mathbf{q}.\end{array}$$

The new splitting theorem is applicable also in many situations that are not covered by the version due to Oikarinen and Janhunen, including, as we will see, some that are important from the perspective of answer set programming and knowledge representation.

The splitting theorem easily follows from a lemma that may be of interest in its own right. The lemma asserts that under certain syntactic conditions

$$\mathrm{SM}_{\mathbf{pq}}[F] \quad \text{is equivalent to} \quad \mathrm{SM}_{\mathbf{p}}[F] \wedge \mathrm{SM}_{\mathbf{q}}[F].$$

This "splitting lemma" is more general than the splitting theorem in the sense that it is applicable, in principle, to formulas of any syntactic form, not only to conjunctions. On the other hand, it does not split the formula itself; it only splits the list of intensional predicates.

We will see that the assumption about F, $\mathbf{p}$ and $\mathbf{q}$ in the statement of the splitting lemma is related to the concept of a loop, introduced in [Lin and Zhao, 2004] for traditional logic programs and generalized to arbitrary first-order formulas in [Lee and Meng, 2008].

This paper includes proofs of the splitting lemma and the splitting theorem, and also a brief discussion of the applications of splitting that motivated this work.

2 Review: Operator SM

This review follows [Ferraris *et al.*, 2010]. Notation: if p and q are predicate constants of the same arity then $p \le q$ stands for the formula

$$\forall \mathbf{x}(p(\mathbf{x}) \to q(\mathbf{x})),$$

where $\mathbf{x}$ is a tuple of distinct object variables. If $\mathbf{p}$ and $\mathbf{q}$ are tuples $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ of predicate constants then $\mathbf{p} \le \mathbf{q}$ stands for the conjunction

$$(p_1 \le q_1) \wedge \cdots \wedge (p_n \le q_n),$$

and $\mathbf{p} < \mathbf{q}$ stands for $(\mathbf{p} \le \mathbf{q}) \wedge \neg(\mathbf{q} \le \mathbf{p})$. In second-order logic, we apply the same notation to tuples of predicate variables.

We will define the *stable model operator with the intensional predicates* $\mathbf{p}$, denoted by $\mathrm{SM}_{\mathbf{p}}$. Some details of the definition depend on which propositional connectives and quantifiers are treated as primitives, and which of them are viewed as abbreviations. We assume that

$$\bot \text{ (falsity)}, \ \wedge, \ \vee, \ \to, \ \forall, \ \exists$$

are the primitives; $\neg F$ stands for $F \to \bot$, $\top$ is $\bot \to \bot$, and $F \leftrightarrow G$ is $(F \to G) \wedge (G \to F)$.

Let $\mathbf{p}$ be a list of distinct predicate constants $p_1, \ldots, p_n$ other than equality. For any first-order sentence F, by $\mathrm{SM}_{\mathbf{p}}[F]$ we denote the second-order sentence

$$F \wedge \neg\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where $\mathbf{u}$ is a list of n distinct predicate variables $u_1, \ldots, u_n$, and $F^*(\mathbf{u})$ is defined recursively:

- $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ for any tuple $\mathbf{t}$ of terms;
- $F^* = F$ for any atomic F that does not contain members of $\mathbf{p}$;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \to G)^* = (F^* \to G^*) \wedge (F \to G)$;
- $(\forall x F)^* = \forall x F^*$;
- $(\exists x F)^* = \exists x F^*$.

A model of F is *stable* (relative to the set $\mathbf{p}$ of intensional predicates) if it satisfies $\mathrm{SM}_{\mathbf{p}}[F]$.

For instance, let F be the formula

$$\forall x(p(x) \to (q(x) \vee \neg q(x)), \tag{8}$$

representing the choice rule

```
{q(X)} :- p(X).
```

If we take q to be the only intensional predicate then $F^*(u)$ is

$$\begin{array}{l}\forall x((p(x) \to (u(x) \vee (\neg u(x) \wedge \neg q(x))))\\ \quad \wedge (p(x) \to (q(x) \vee \neg q(x)))),\end{array}$$

so that $\mathrm{SM}_q[F]$ is

$$\begin{array}{l}\forall x(p(x) \to (q(x) \vee \neg q(x)))\\ \quad \wedge \neg\exists u((u < q)\\ \qquad \wedge \forall x((p(x) \to (u(x) \vee (\neg u(x) \wedge \neg q(x))))\\ \qquad\quad \wedge (p(x) \to (q(x) \vee \neg q(x))))).\end{array} \tag{9}$$

This sentence is equivalent to the first-order formula

$$\forall x(q(x) \to p(x)),$$

which reflects the intuitive meaning of choice: q is an arbitrary subset of p.

3 Splitting Lemma

The statement of the splitting lemma refers to the predicate dependency graph of a formula, defined in [Ferraris *et al.*, 2007] and [Ferraris *et al.*, 2010].[4] This definition, reproduced below, generalizes the idea that if a logic program contains, say, the rule

$$p(x) \leftarrow q(x), \mathit{not}\ r(x) \tag{10}$$

then p "positively depends" on q in this program.

Recall that an occurrence of a predicate constant, or any other subexpression, in a formula is called *positive* if the number of implications containing that occurrence in the antecedent is even, and *strictly positive* if that number is 0. We say that an occurrence of a predicate constant in a formula is *negated* if it belongs to a subformula of the form $\neg F$ (that is, $F \to \bot$), and *nonnegated* otherwise. For instance, in the formula

$$q(x) \wedge \neg r(x) \to p(x), \tag{11}$$

corresponding to rule (10), both p and r are positive, p is strictly positive, and r is negated.

A *rule* of a first-order formula F is a strictly positive occurrence of an implication in F. For instance, the rules of (8) are $p(x) \to (q(x) \vee \neg q(x))$ and $\neg q(x)$; the only rule of (11) is (11) itself.

For any first-order formula F, the *predicate dependency graph* of F (relative to the list $\mathbf{p}$ of intensional predicates) is the directed graph that

- has all intensional predicates as its vertices, and
- has an edge from p to q if, for some rule $G \to H$ of F,
  - p has a strictly positive occurrence in H, and
  - q has a positive nonnegated occurrence in G.

For instance, the predicate dependency graph of (11) relative to pqr has one edge, from p to q. We will denote the predicate dependency graph of F relative to $\mathbf{p}$ by $\mathrm{DG}_{\mathbf{p}}[F]$.

Splitting Lemma, Version 1 *Let F be a first-order sentence, and let $\mathbf{p}$, $\mathbf{q}$ be disjoint tuples of distinct predicate constants. If each strongly connected component of $\mathrm{DG}_{\mathbf{pq}}[F]$ is a subset of $\mathbf{p}$ or a subset of $\mathbf{q}$ then*

$$\mathrm{SM}_{\mathbf{pq}}[F] \quad \textit{is equivalent to} \quad \mathrm{SM}_{\mathbf{p}}[F] \wedge \mathrm{SM}_{\mathbf{q}}[F].$$

Note that the condition on $\mathrm{DG}_{\mathbf{pq}}[F]$ in the statement of the lemma holds trivially if all strongly connected components of this graph are singletons.

Example 1: F is $\neg p \wedge r \to q$, $\mathbf{p}$ is p, $\mathbf{q}$ is q. In this case, the graph $\mathrm{DG}_{\mathbf{pq}}[F]$ has two vertices p, q, and no edges, so that its strongly connected components are singletons. The splitting lemma asserts that

$$\mathrm{SM}_{pq}[\neg p \wedge r \to q] \tag{12}$$

is equivalent to the conjunction of

$$\mathrm{SM}_{p}[\neg p \wedge r \to q] \tag{13}$$

and

$$\mathrm{SM}_{q}[\neg p \wedge r \to q]. \tag{14}$$

Each of these three expressions can be rewritten as a propositional formula using the methods described in [Ferraris *et al.*, 2010]. Formula (12) becomes

$$(p \leftrightarrow \bot) \wedge (q \leftrightarrow \neg p \wedge r), \tag{15}$$

(13) becomes

$$(\neg p \wedge r \to q) \wedge \neg p, \tag{16}$$

and (14) turns into

$$q \leftrightarrow \neg p \wedge r. \tag{17}$$

It is clear that (15) is indeed equivalent to the conjunction of (16) and (17).

[4] We follow here the version given in the second paper, which is simpler but can introduce some unnecessary dependencies when applied to deeply nested implications.

Example 2: F is $r \to p \vee q$, $\mathbf{p}$ is p, $\mathbf{q}$ is q. The graph $\mathrm{DG}_{\mathbf{pq}}[F]$ is the same as in Example 1, and the splitting lemma asserts that

$$\mathrm{SM}_{pq}[r \to p \vee q] \tag{18}$$

is equivalent to the conjunction of

$$\mathrm{SM}_{p}[r \to p \vee q] \tag{19}$$

and

$$\mathrm{SM}_{q}[r \to p \vee q]. \tag{20}$$

The methods for simplifying $\mathrm{SM}_{\mathbf{p}}[F]$ described in [Ferraris *et al.*, 2010] are not directly applicable to (18), but they allow us to simplify (19) and (20). The version of program completion presented in that paper turns the former into $p \leftrightarrow \neg q \wedge r$ and the latter into $q \leftrightarrow \neg p \wedge r$. Consequently (18) is equivalent to the conjunction of these two formulas.

Example 2 shows that the splitting lemma allows us to expand the power of completion, as a method for describing stable models, to some disjunctive programs. This is similar to the generalization of completion to disjunctive programs described in [Lee and Lifschitz, 2003]; the advantage of the splitting lemma is that it is applicable to programs with variables. For instance, using the same argument as in Example 2 we can check that

$$\mathrm{SM}_{pq}[\forall xy(r(x,y) \to p(x) \vee q(y))]$$

is equivalent to the conjunction of

$$\forall x(p(x) \leftrightarrow \exists y(\neg q(y) \wedge r(x,y)))$$

and

$$\forall y(q(y) \leftrightarrow \exists x(\neg p(x) \wedge r(x,y))).$$

To illustrate the role of the condition on the predicate dependency graph in the statement of the splitting lemma, take F to be $p \leftrightarrow q$, with p as $\mathbf{p}$ and q as $\mathbf{q}$. The graph $\mathrm{DG}_{\mathbf{pq}}[F]$ in this case has two edges, from p to q and from q to p. The strongly connected component $\{p, q\}$ of this graph has a common element with $\mathbf{p}$ and a common element with $\mathbf{q}$, so that the splitting lemma is not applicable. Accordingly, the formulas $\mathrm{SM}_{pq}[p \leftrightarrow q]$ and $\mathrm{SM}_{p}[p \leftrightarrow q] \wedge \mathrm{SM}_{q}[p \leftrightarrow q]$ are not equivalent to each other. Indeed, the former can be rewritten as $\neg p \wedge \neg q$, and each conjunctive term of the latter is equivalent to $p \leftrightarrow q$.

The splitting lemma as stated above can be equivalently reformulated as follows:

Splitting Lemma, Version 2 *Let F be a first-order sentence, and let $\mathbf{p}$ be a tuple of distinct predicate constants. If $\mathbf{c}^1, \dots, \mathbf{c}^n$ are all the strongly connected components of $\mathrm{DG}_{\mathbf{p}}[F]$ then*

$$\mathrm{SM}_{\mathbf{p}}[F] \quad \textit{is equivalent to} \quad \mathrm{SM}_{\mathbf{c}^1}[F] \wedge \cdots \wedge \mathrm{SM}_{\mathbf{c}^n}[F].$$

A *loop* of a first-order formula F (relative to a list $\mathbf{p}$ of intensional predicates) is a nonempty subset $\mathbf{l}$ of $\mathbf{p}$ such that the subgraph of $\mathrm{DG}_{\mathbf{p}}[F]$ induced by $\mathbf{l}$ is strongly connected. It is clear that the strongly connected components of $\mathrm{DG}_{\mathbf{p}}[F]$ can be characterized as the maximal loops of F.

Splitting Lemma, Version 3 *Let F be a first-order sentence, and let $\mathbf{p}$ be a tuple of distinct predicate constants. If $\mathbf{l}^1, \dots, \mathbf{l}^n$ are all the loops of F relative to $\mathbf{p}$ then*

$$\mathrm{SM}_{\mathbf{p}}[F] \quad \textit{is equivalent to} \quad \mathrm{SM}_{\mathbf{l}^1}[F] \wedge \cdots \wedge \mathrm{SM}_{\mathbf{l}^n}[F].$$

The last two versions of the splitting lemma are equivalent to each other in view of the fact that the operator $\mathrm{SM}_{\mathbf{p}}$ is monotone with respect to $\mathbf{p}$: if $\mathbf{p}$ contains $\mathbf{q}$ then $\mathrm{SM}_{\mathbf{p}}[F]$ entails $\mathrm{SM}_{\mathbf{q}}[F]$.

4 Proof of the Splitting Lemma

Lemmas 1 and 2 below can be easily proved by induction [Ferraris *et al.*, 2010].

Lemma 1 *Formula*

$$(\mathbf{u} \le \mathbf{p}) \wedge F^*(\mathbf{u}) \to F$$

is logically valid.

About a formula F we say that it is *negative* on a tuple $\mathbf{p}$ of predicate constants if members of $\mathbf{p}$ have no strictly positive occurrences in F.

Lemma 2 *If F is negative on $\mathbf{p}$ then*

$$(\mathbf{u} \le \mathbf{p}) \to (F^*(\mathbf{u}) \leftrightarrow F)$$

is logically valid.

The following lemma extends Lemma 3 from [Ferraris *et al.*, 2006] to first-order formulas.

Lemma 3 *Let $\mathbf{p}_1$, $\mathbf{p}_2$ be disjoint lists of distinct predicate constants, and let $\mathbf{u}_1$, $\mathbf{u}_2$ be disjoint lists of distinct predicate variables of the same length as $\mathbf{p}_1$, $\mathbf{p}_2$ respectively.*

(a) If every positive occurrence of every predicate constant from $\mathbf{p}_2$ in F is negated then

$$((\mathbf{u}_1, \mathbf{u}_2) \le (\mathbf{p}_1, \mathbf{p}_2)) \wedge F^*(\mathbf{u}_1, \mathbf{p}_2) \to F^*(\mathbf{u}_1, \mathbf{u}_2)$$

is logically valid.

(b) If every nonpositive occurrence of every predicate constant from $\mathbf{p}_2$ in F is negated then

$$((\mathbf{u}_1, \mathbf{u}_2) \le (\mathbf{p}_1, \mathbf{p}_2)) \wedge F^*(\mathbf{u}_1, \mathbf{u}_2) \to F^*(\mathbf{u}_1, \mathbf{p}_2)$$

is logically valid.

Proof. Both parts are proved simultaneously by induction on F. Consider the case when F is $G \to H$; the other cases are straightforward. Then $F^*(\mathbf{u}_1, \mathbf{u}_2)$ is

$$(G^*(\mathbf{u}_1, \mathbf{u}_2) \to H^*(\mathbf{u}_1, \mathbf{u}_2)) \wedge (G \to H). \tag{21}$$

(a) Every nonpositive occurrence of every predicate constant from $\mathbf{p}_2$ in G is negated, and so is every positive occurrence of every predicate constant from $\mathbf{p}_2$ in H. By the induction hypothesis, it follows that the formulas

$$((\mathbf{u}_1, \mathbf{u}_2) \le (\mathbf{p}_1, \mathbf{p}_2)) \wedge G^*(\mathbf{u}_1, \mathbf{u}_2) \to G^*(\mathbf{u}_1, \mathbf{p}_2) \tag{22}$$

and

$$(\mathbf{u}_1, \mathbf{u}_2) \le (\mathbf{p}_1, \mathbf{p}_2) \wedge H^*(\mathbf{u}_1, \mathbf{p}_2) \to H^*(\mathbf{u}_1, \mathbf{u}_2) \tag{23}$$

are logically valid. Assume $(\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)$,

$$(G^*(\mathbf{u}_1, \mathbf{p}_2) \to H^*(\mathbf{u}_1, \mathbf{p}_2)) \wedge (G \to H) \qquad (24)$$

and $G^*(\mathbf{u}_1, \mathbf{u}_2)$. By (22), we conclude $G^*(\mathbf{u}_1, \mathbf{p}_2)$. Then, by (24), we conclude $H^*(\mathbf{u}_1, \mathbf{p}_2)$. Then, by (23), we conclude $H^*(\mathbf{u}_1, \mathbf{u}_2)$. (b) Similar. ∎

The following assertion is a generalization of Lemma 5 from [Ferraris *et al.*, 2006].

Lemma 4 *Let $\mathbf{p}_1$, $\mathbf{p}_2$ be disjoint lists of distinct predicate constants such that $\mathrm{DG}_{\mathbf{p}_1\mathbf{p}_2}[F]$ has no edges from predicate constants in $\mathbf{p}_1$ to predicate constants in $\mathbf{p}_2$, and let $\mathbf{u}_1$, $\mathbf{u}_2$ be disjoint lists of distinct predicate variables of the same length as $\mathbf{p}_1$, $\mathbf{p}_2$ respectively. Formula*

$$((\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge F^*(\mathbf{u}_1, \mathbf{u}_2) \to F^*(\mathbf{u}_1, \mathbf{p}_2)$$

is logically valid.

Proof. By induction on F. Consider the case when F is $G \to H$, so that $F^*(\mathbf{u}_1, \mathbf{u}_2)$ is (21); the other cases are straightforward. Assume $(\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)$ and $F^*(\mathbf{u}_1, \mathbf{u}_2)$. Our goal is to prove

$$G^*(\mathbf{u}_1, \mathbf{p}_2) \to H^*(\mathbf{u}_1, \mathbf{p}_2).$$

Assume $G^*(\mathbf{u}_1, \mathbf{p}_2)$. By Lemma 1, the formula

$$((\mathbf{u}_1, \mathbf{p}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge G^*(\mathbf{u}_1, \mathbf{p}_2) \to G \qquad (25)$$

is logically valid. Consequently, from the assumptions above we can conclude G, and, by (21), H. *Case 1:* H is negative on $\mathbf{p}_1$. It follows from Lemma 2 that the formula

$$((\mathbf{u}_1, \mathbf{p}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \to (H^*(\mathbf{u}_1, \mathbf{p}_2) \leftrightarrow H)$$

is logically valid, and we can conclude that $H^*(\mathbf{u}_1, \mathbf{p}_2)$. *Case 2:* H is not negative on $\mathbf{p}_1$, that is to say, H contains a strictly positive occurrence of a predicate constant from $\mathbf{p}_1$. Then every positive occurrence of every predicate constant from $\mathbf{p}_2$ in G is negated, because otherwise there would exist an edge from $\mathbf{p}_1$ to $\mathbf{p}_2$ in $\mathrm{DG}_{\mathbf{p}_1\mathbf{p}_2}[F]$. By Lemma 3(a), the formula

$$((\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge G^*(\mathbf{u}_1, \mathbf{p}_2) \to G^*(\mathbf{u}_1, \mathbf{u}_2)$$

is logically valid. Consequently from the assumptions above we can conclude that $G^*(\mathbf{u}_1, \mathbf{u}_2)$. By (21), it follows that $H^*(\mathbf{u}_1, \mathbf{u}_2)$. Since every edge in $\mathrm{DG}_{\mathbf{p}_1\mathbf{p}_2}[H]$ belongs to $\mathrm{DG}_{\mathbf{p}_1\mathbf{p}_2}[F]$, by the induction hypothesis applied to H, the formula

$$((\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge H^*(\mathbf{u}_1, \mathbf{u}_2) \to H^*(\mathbf{u}_1, \mathbf{p}_2)$$

is logically valid. We can thus conclude that $H^*(\mathbf{u}_1, \mathbf{p}_2)$. ∎

Lemma 5 *For any formula F and any nonempty set Y of intensional predicates, there exists a subset Z of Y such that*

(a) Z is a loop of F, and

(b) the predicate dependency graph of F has no edges from predicate constants in Z to predicate constants in $Y \setminus Z$.

The proof is essentially the same as the proof of Lemma 4 in [Ferraris *et al.*, 2006].

Proof of Version 3 of the Splitting Lemma. It is sufficient to prove the logical validity of the formula

$$\begin{aligned}\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})) \\ \leftrightarrow\ \exists \mathbf{u}^1((\mathbf{u}^1 < \mathbf{l}^1) \wedge F^*(\widetilde{\mathbf{u}^1})) \\ \vee \cdots \vee \exists \mathbf{u}^n((\mathbf{u}^n < \mathbf{l}^n) \wedge F^*(\widetilde{\mathbf{u}^n})),\end{aligned}$$

where each $\mathbf{u}^i$ is the part of $\mathbf{u}$ that corresponds to the part $\mathbf{l}^i$ of $\mathbf{p}$, and $\widetilde{\mathbf{u}^i}$ is the list of symbols obtained from $\mathbf{p}$ by replacing every intensional predicate p that belongs to $\mathbf{l}^i$ with the corresponding predicate variable u. *Right to left:* Clear. *Left to right:* Assume $\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u}))$ and take $\mathbf{u}$ such that $(\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})$. Consider several cases, each corresponding to a nonempty subset Y of $\mathbf{p}$. The assumption characterizing each case is that $u < p$ for each member p of $\mathbf{p}$ that belongs to Y, and that $u = p$ for each p that does not belong to Y. By Lemma 5, there is a loop $\mathbf{l}^i$ of F that is contained in Y such that the dependency graph has no edges from predicate constants in $\mathbf{l}^i$ to predicate constants in $Y \setminus \mathbf{l}^i$. Since $\mathbf{l}^i$ is contained in Y, from the fact that $u < p$ for each p in Y we can conclude that

$$\mathbf{u}^i < \mathbf{l}^i. \qquad (26)$$

Let $\mathbf{u}'$ be the list of symbols obtained from $\mathbf{p}$ by replacing every member p that belongs to Y with the corresponding variable u. Under the assumption characterizing each case, $\mathbf{u} = \mathbf{u}'$, so that $F^*(\mathbf{u}) \leftrightarrow F^*(\mathbf{u}')$. Consequently, we can derive $F^*(\mathbf{u}')$. It follows from Lemma 4 that the formula

$$(\mathbf{u}' \leq \mathbf{p}) \wedge F^*(\mathbf{u}') \to F^*(\widetilde{\mathbf{u}^i})$$

is logically valid, so that we further conclude that $F^*(\widetilde{\mathbf{u}^i})$. In view of (26), it follows that $\exists \mathbf{u}^i((\mathbf{u}^i < \mathbf{l}^i) \wedge F^*(\widetilde{\mathbf{u}^i}))$. ∎

5 Application: Dropping Double Negations

The semantics of the answer set programming language RASPL-1 is defined in [Lee *et al.*, 2008] using a syntactic translation that turns every RASPL-1 program into a first-order sentence, called its *FOL-representation*. An answer set of a RASPL-1 program Π is defined as an arbitrary Herbrand stable model of the FOL-representation of Π.

To relate RASPL-1 to cardinality constraint programs in the sense of [Syrjänen, 2004], the authors defined a class of *strongly regular* RASPL-1 programs. The answer sets of any program in this class are identical to its answer sets in the sense of Syrjänen [Lee *et al.*, 2008, Proposition 5]. This fact is stated in the paper without proof, and the key part of the proof involves checking that, under certain conditions, dropping a double negation from a first-order sentence does not affect its stable models. One such case is described in the following proposition:

Theorem on Double Negations *Let H be a sentence, F a subformula of H, and H^- the sentence obtained from H by inserting $\neg\neg$ in front of F. If F is contained in a subformula G of H that is negative on $\mathbf{p}$ then $\mathrm{SM}_{\mathbf{p}}[H^-]$ is equivalent to $\mathrm{SM}_{\mathbf{p}}[H]$.*

Proof. Let G^- be the formula obtained from G by inserting $\neg\neg$ in front of F. By Lemma 2, the formulas

$$\mathbf{u} \leq \mathbf{p} \to (G^*(\mathbf{u}) \leftrightarrow G)$$

and

$$\mathbf{u} \leq \mathbf{p} \to ((G^-)^*(\mathbf{u}) \leftrightarrow G^-)$$

are logically valid. Consequently

$$\mathbf{u} \leq \mathbf{p} \to (G^*(\mathbf{u}) \leftrightarrow (G^-)^*(\mathbf{u}))$$

is logically valid also, and so is

$$\mathbf{u} \leq \mathbf{p} \to (H^*(\mathbf{u}) \leftrightarrow (H^-)^*(\mathbf{u})).$$

It follows that $\mathrm{SM}_{\mathbf{p}}[H^-]$ is equivalent to $\mathrm{SM}_{\mathbf{p}}[H]$. ∎

By itself, this theorem does not help us prove Proposition 5 from [Lee *et al.*, 2008]: the condition

> *F is contained in a subformula G of H that is negative on* $\mathbf{p}$

turns out to be too restrictive. What we need to observe is that this condition can be relaxed as follows:

> *for every strongly connected component* $\mathbf{c}$ *of* $\mathrm{DG}_{\mathbf{p}}[H]$*, F is contained in a subformula G of H that is negative on* $\mathbf{c}$.

The possibility of strengthening the theorem on double negations in this way is immediate from Version 2 of the splitting lemma.

6 Splitting Theorem

Recall that a formula F is said to be *negative* on a tuple $\mathbf{p}$ of predicate constants if members of $\mathbf{p}$ have no strictly positive occurrences in F (Section 4). The importance of this class of formulas for the theory of stable models is that they are analogous to constraints in traditional answer set programming: $\mathrm{SM}_{\mathbf{p}}[F \wedge G]$ is equivalent to $\mathrm{SM}_{\mathbf{p}}[F] \wedge G$ whenever G is negative on $\mathbf{p}$ [Ferraris *et al.*, 2010].

Splitting Theorem *Let F, G be first-order sentences, and let $\mathbf{p}$, $\mathbf{q}$ be disjoint tuples of distinct predicate constants. If*

- *each strongly connected component of* $\mathrm{DG}_{\mathbf{pq}}[F \wedge G]$ *is a subset of* $\mathbf{p}$ *or a subset of* $\mathbf{q}$*,*
- *F is negative on* $\mathbf{q}$*, and*
- *G is negative on* $\mathbf{p}$

then

$$\mathrm{SM}_{\mathbf{pq}}[F \wedge G] \quad \textit{is equivalent to} \quad \mathrm{SM}_{\mathbf{p}}[F] \wedge \mathrm{SM}_{\mathbf{q}}[G].$$

Proof. By the splitting lemma, $\mathrm{SM}_{\mathbf{pq}}[F \wedge G]$ is equivalent to

$$\mathrm{SM}_{\mathbf{p}}[F \wedge G] \wedge \mathrm{SM}_{\mathbf{q}}[F \wedge G].$$

Since G is negative on $\mathbf{p}$, the first conjunctive term can be rewritten as

$$\mathrm{SM}_{\mathbf{p}}[F] \wedge G. \tag{27}$$

Similarly, the second conjunctive term can be rewritten as

$$\mathrm{SM}_{\mathbf{q}}[G] \wedge F. \tag{28}$$

It remains to observe that the second conjunctive term of each of the formulas (27), (28) is entailed by the first conjunctive term of the other. ∎

7 Application: Abstract ASP Programs

The intuition behind the rules of the program Π from the introduction is easy to explain. The first rule says that the extent of the predicate `in` is an arbitrary set of vertices. The second rule expresses the definition of a clique: a clique may not contain a pair of distinct vertices that are not adjacent to each other. The third rule expresses a restriction on the size of the clique: it is not allowed to be $\leq \mathtt{n}$.

The claim that Π describes large cliques in a graph can be made precise in two ways: by considering logic programs obtained from Π by adding descriptions of specific graphs, and in an abstract way, as a theorem about program Π itself. In both formulations below, σ is the signature consisting of the predicate constants *vertex*, *edge*, and *in*, and Γ is an arbitrary finite directed graph.

Correctness of Π, Formulation 1. Extend σ by adding the vertices of Γ as object constants. Let I be an interpretation of the extended signature that interprets each object constant as itself, interprets *vertex* as the set of vertices of Γ, and interprets *edge* as the set of edges of Γ. Let X be the set of ground atoms that contain *vertex* or *edge* and are satisfied by I. The following conditions are equivalent:

- I is a stable model of the conjunction of (6) and the atoms X, with the intensional predicates *vertex*, *edge*, and *in*,
- the extent of *in* in I is a clique in Γ, and its cardinality is $> n$.

Correctness of Π, Formulation 2. Let I be an interpretation of σ that interprets *vertex* as the set of vertices of Γ, and interprets *edge* as the set of edges of Γ. The following conditions are equivalent:

- I is a stable model of (6) with the intensional predicate *in*,
- the extent of *in* in I is a clique in Γ, and its cardinality is $> n$.

The correctness of Π in the sense of Formulation 2 is immediate from the fact that the result of applying the operator $\mathrm{SM}_{\mathit{in}}$ to (6) can be equivalently written as

$$\begin{aligned}&\forall x(\mathit{in}(x) \to \mathit{vertex}(x))\\ &\wedge \forall xy\neg(\mathit{vertex}(x) \wedge \mathit{vertex}(y) \wedge \mathit{in}(x) \wedge \mathit{in}(y)\\ &\qquad \wedge \neg \mathit{edge}(x,y) \wedge \neg \mathit{edge}(y,x) \wedge x \neq y)\\ &\wedge \exists_{n+1} x(\mathit{in}(x) \wedge \mathit{vertex}(x)).\end{aligned}$$

This fact is easy to verify using the methods of [Ferraris *et al.*, 2010]. Formulation 1 follows from Formulation 2 by the splitting theorem, with (6) as F, the conjunction of the atoms X as G, *in* as $\mathbf{p}$, and *vertex*,*edge* as $\mathbf{q}$.

The advantage of Formulation 1 is that it describes what happens when we actually use Π to find a large clique in a graph: we run an answer set solver on a program obtained from Π by adding the definition of the graph as a set of facts. The advantage of the "abstract" Formulation 2 is that it is easier to state and to prove. The splitting theorem can be used to establish a relationship between these two kinds of correctness theorems.

8 Application: Relationship between Two Formulations of the Event Calculus

Theorem 1 from [Kim *et al.*, 2009] shows that circumscriptive event calculus [Shanahan, 1997] can be reformulated in terms of the stable model semantics in the form

$$\mathrm{SM}_{Initiates,Terminates,Releases}[\Sigma] \wedge \ \mathrm{SM}_{Happens}[\Delta] \wedge \Xi.$$

The splitting theorem stated above is used there to show that this formula can be equivalently written using a single application of the operator SM:

$$\mathrm{SM}_{Initiates,Terminates,Releases,Happens}[\Sigma \wedge \Delta \wedge \Xi].$$

A further transformation is shown in [Kim *et al.*, 2009] to turn the conjunction $\Sigma \wedge \Delta \wedge \Xi$ into a logic program that can be processed by existing answer set solvers.

9 Conclusion

The splitting lemma and the splitting theorem proved in this note appear to be valuable mathematical tools. The former helped us extend program completion to some disjunctive programs, including programs with variables, and to compare two approaches to the semantics of aggregates in ASP. The latter can be used to relate "abstract" ASP programs to programs containing specific facts and to turn one of the formulations of the event calculus into an executable ASP program. We hope that future work will bring us new applications of splitting to answer set programming and knowledge representation.

Acknowledgements

We are grateful to Michael Gelfond, Tomi Janhunen and Emilia Oikarinen for useful discussions related to the topic of this paper. This work was partially supported by the National Science Foundation under Grants IIS-0712113 and IIS-0839821.

References

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Ferraris *et al.*, 2006] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, 47:79–101, 2006.

[Ferraris *et al.*, 2007] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 372–379, 2007.

[Ferraris *et al.*, 2010] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription.[5] *Artificial Intelligence*, 2010. To appear.

[5] *http://peace.eas.asu.edu/joolee/papers/smcirc.pdf* .

[Janhunen *et al.*, 2007] Tomi Janhunen, Emilia Oikarinen, Hans Tompits, and Stefan Woltran. Modularity aspects of disjunctive stable models. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 175–187, 2007.

[Kim *et al.*, 2009] Tae-Won Kim, Joohyung Lee, and Ravi Palla. Circumscriptive event calculus as answer set programming. 2009. This volume.

[Lee and Lifschitz, 2003] Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 451–465, 2003.

[Lee and Meng, 2008] Joohyung Lee and Yunsong Meng. On loop formulas with variables. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, pages 444–453, 2008.

[Lee *et al.*, 2008] Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 472–479, 2008.

[Lifschitz and Turner, 1994] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proceedings of International Conference on Logic Programming (ICLP)*, pages 23–37, 1994.

[Lifschitz, 2008] Vladimir Lifschitz. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1594–1597, 2008.

[Lin and Zhao, 2004] Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157:115–137, 2004.

[Marek and Truszczyński, 1999] Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[Oikarinen and Janhunen, 2008] Emilia Oikarinen and Tomi Janhunen. Achieving compositionality of the stable model semantics for Smodels programs. *Theory and Practice of Logic Programming*, 5–6:717–761, 2008.

[Shanahan, 1997] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[Simons *et al.*, 2002] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.

[Syrjänen, 2004] Tommi Syrjänen. Cardinality constraint programs. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, pages 187–199, 2004.