Splitting a Logic Program Revisited

Jianmin Ji
School of Computer Science and Technology
University of Science and Technology of China
Hefei 230027, China
jianmin@ustc.edu.cn

Hai Wan* and **Ziwei Huo** and **Zhenfeng Yuan**
School of Software
Sun Yat-sen University
Guangzhou 510006, China
wanhai@mail.sysu.edu.cn

Abstract

Lifschitz and Turner introduced the notion of the splitting set and provided a method to divide a logic program into two parts. They showed that the task of computing the answer sets of the program can be converted into the tasks of computing the answer sets of these parts. However, the empty set and the set of all atoms are the only two splitting sets for many programs, then these programs cannot be divided by the splitting method. In this paper, we extend Lifschitz and Turner's splitting set theorem to allow the program to be split by an arbitrary set of atoms, while some new atoms may be introduced in the process. To illustrate the usefulness of the result, we show that for some typical programs the splitting process is efficient and the program simplification problem can be investigated using the concept of splitting.

Introduction

The notion of the splitting set was first introduced by Lifschitz and Turner (1994). Using a splitting set, a logic program can be divided into two parts, *i.e.*, the "bottom" part and the "top" part, the "bottom" part of which only mentions atoms in the splitting set. Moreover, the answer sets of the program can be computed by first computing the answer sets of the "bottom" part and then using them in the "top" part to determine values of the rest atoms.

The idea of splitting has been proved to be a useful method to decompose programs and a helpful tool to investigate answer set semantics (Dao-Tran et al. 2009). Lifschitz and Turner's splitting set theorem has been considered as the theoretical foundation for the incremental ASP solver, iclingo (Gebser et al. 2008). Beside, this idea has been extended to logic programs with nested expressions (Oikarinen and Janhunen 2008) and arbitrary first-order formulas with stable model semantics (Ferraris et al. 2009).

However, in many applications, the empty set and the set of all atoms are the only two splitting sets of the program. Then the program cannot be divided using Lifschitz and Turner's splitting method. In this paper, we propose a new splitting method that allows the program to be split into two parts by an arbitrary set of atoms, while one of them, the "top" part, may introduce some new atoms. We show that the task of computing the answer sets of the program can be converted into the tasks of computing the answer sets of these parts. The result extends Lifschitz and Turner's splitting set theorem. When a splitting set is used to split a program, the splitting result in our method is the same as the result in Lifschitz and Turner's splitting method.

The usefulness of the result is illustrated through two aspects. First, we discuss some computational complexity issues related to the splitting method. We show that for some typical programs the splitting process is efficient, which implies some potential applications. Second, we use the concept of splitting to investigate the program simplification problem, *i.e.*, how to simplify a program by a set of atoms that are satisfied by every answer set of the program. We show that this problem can be regarded as a splitting problem, which results a new approach to simplify a program.

Preliminaries

Logic Programs

In this paper, we consider only fully grounded finite logic programs. A *(disjunctive) logic program (DLP)* is a finite set of (disjunctive) rules of the form

$$a_1 \vee \cdots \vee a_k \leftarrow a_{k+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n. \quad (1)$$

where $n \geq m \geq k \geq 1$ and $a_1, \ldots, a_n$ are atoms. If $k = 1$, it is a *normal rule*. In particular, a *normal logic program (NLP)* is a finite set of normal rules. With a slight abuse of the notion, a formula of the form

$$\leftarrow a_{k+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n.$$

is considered as an abbreviation of the rule

$$f \leftarrow not\, f, a_{k+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n.$$

where f is a new atom that does not appear in other rules.

We will also write rule r of form (1) as

$$head(r) \leftarrow body(r).$$

where $head(r)$ is $a_1 \vee \cdots \vee a_k$, $body(r) = body^+(r) \wedge body^-(r)$, $body^+(r)$ is $a_{k+1} \wedge \cdots \wedge a_m$, and $body^-(r)$ is $\neg a_{m+1} \wedge \cdots \wedge \neg a_n$, and we identify $head(r)$, $body^+(r)$, $body^-(r)$ with their corresponding sets of atoms, and $body(r)$ the set $\{a_{k+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n\}$. We denote $Atoms(r) = head(r) \cup body^+(r) \cup body^-(r)$. Let R be a set of rules, we denote $head(R) = \bigcup_{r \in R} head(r)$, $body^+(R) = \bigcup_{r \in R} body^+(r)$, and $Atoms(R) = \bigcup_{r \in R} Atoms(r)$.

A set S of atoms *satisfies* a propositional formula ϕ, written as $S \models \phi$, is defined following the usual recursive

*Corresponding author

conditions. We say S *satisfies* a rule r, if $S \models body(r)$ implies $S \models head(r)$. S *satisfies* a program P, if S satisfies all rules in P.

The *answer sets* of a DLP are defined in (Gelfond and Lifschitz 1991). Given a DLP P and a set S of atoms, the GL transformation of P on S, written P^S, is obtained from P by deleting:

1. each rule that has $not\, p$ in its body with $p \in S$, and
2. all $not\, p$ in the bodies of the remaining rules.

For any S, P^S has a set of minimal models, denoted $\Gamma(P^S)$. Now a set S of atoms is an *answer set* of P iff $S \in \Gamma(P^S)$.

Loops and Loop Formulas

The notions of loops and loop formulas were first proposed by Lin and Zhao (2004) for NLPs. They showed that a set of atoms is an answer set of an NLP iff it satisfies both the loop formulas and the program. Lee and Lifschitz (2003) extend the notions and result to DLPs.

Given a DLP P, the *positive dependency graph* of P, written G_P, is the directed graph whose vertices are atoms in P, and there is an arc from p to q if there is a rule $r \in P$ *s.t.* $p \in head(r)$ and $q \in body^+(r)$. A set L of atoms is said to be a *loop* of P if the L-induced subgraph of G_P is strongly connected. Note that, every singleton whose atom occurs in P is also a loop of P.

Example 1 *Consider the logic program* P_1*:*

$a \leftarrow not\, d. \quad d \leftarrow not\, c. \quad a \leftarrow c, d. \quad c \leftarrow a.$

Figure 1 shows the positive dependency graph G_{P_1}*.* $\{a, c\}$*,* $\{a\}$*,* $\{c\}$*, and* $\{d\}$ *are loops of* P_1*.*

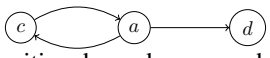

Figure 1: The positive dependency graph of program P_1

Given a loop L, a rule r is an *external support* of L if $head(r) \cap L \neq \emptyset$ and $L \cap body^+(r) = \emptyset$. In the following, let $R^-(L, P)$ be the set of external support rules of L. The function R^- can be defined for any set E of atoms, *i.e.*, $R^-(E, P) = \{r \in P \mid head(r) \cap E \neq \emptyset,\ E \cap body^+(r) = \emptyset\}$. Given a set X of atoms, the set of *external support rules of* E *under* X, denoted $R^-(E, P, X)$, is the set of rules r *s.t.* $r \in R^-(E, P)$ and $X \models body(r) \wedge \bigwedge_{q \in head(r) \setminus E} \neg q$.

The (conjunctive) *loop formula* of a set L of atoms under a program P, written $LF(L, P)$, is the following implication

$$\bigwedge_{p \in L} p \supset \bigvee_{r \in R^-(L,P)} \left(body(r) \wedge \bigwedge_{q \in head(r) \setminus L} \neg q \right).$$

Theorem 1 (Theorem 1 in (Lee and Lifschitz 2003)) *Let* P *be a logic program and* S *a set of atoms. If* S *satisfies* P*, then the following conditions are equivalent:*

1. S *is an answer set of* P*;*
2. S *satisfies* $LF(L, P)$ *for all loops* L *of* P*.*
3. S *satisfies* $LF(E, P)$ *for all sets* E *of atoms of* P*.*

Splitting Sets

Lifschitz and Turner (1994) introduced the notion of the splitting set and provided a method to split a logic program into the "bottom" part and the "top" part based on the notion. They showed that the answer sets of the program can be computed by first computing the answer sets of the "bottom" part and then using them in the "top" part to determine values of the rest atoms.

A set U of atoms is called a *splitting set* of a program P, if for each $r \in P$, $head(r) \cap U \neq \emptyset$ implies $Atoms(r) \subseteq U$. The set of rules $r \in P$ *s.t.* $head(r) \cap U \neq \emptyset$ is called the *bottom* of P *w.r.t.* U and denoted by $b_U(P)$. The set $P \setminus b_U(P)$ is the *top* of P *w.r.t.* U. Note that, $b_U(P)$ for a splitting set U of a program P mentions only atoms in U.

Example 2 *Consider the logic program* P_2*:*

$a \leftarrow not\, d. \quad d \leftarrow not\, c. \quad a \leftarrow c, d. \quad c \leftarrow .$

$\{c, d\}$ *is a splitting set of* P_2 *and the bottom of* P *w.r.t.* $\{c, d\}$ *is* $b_{\{c,d\}}(P_2) = \{d \leftarrow not\, c.\ \ c \leftarrow .\}$.

Let U, X be sets of atoms, and P a program, we denote $e_U(P, X)$ the set of rules obtained from P by deleting:

1. each rule r *s.t.* $head(r) \cap X \neq \emptyset$, $body^+(r) \cap U \not\subseteq X$, or $(body^-(r) \cap U) \cap X \neq \emptyset$;
2. all formulas of the form a or $not\, a$ where $a \in U$ in the bodies of the remaining rules.

For example, $e_{\{c,d\}}(P_2 \setminus b_{\{c,d\}}(P_2), \{c\}) = \{a \leftarrow .\}$.

Let U be a splitting set of a program P. A *solution* of P *w.r.t.* U is a pair $\langle X, Y \rangle$ of sets of atoms *s.t.*

- X is an answer set of $b_U(P)$,
- Y is an answer set of $e_U(P \setminus b_U(P), X)$.

For example, $\langle \{c\}, \{a\} \rangle$ is a solution of P_2 *w.r.t.* $\{c, d\}$.

Theorem 2 (Splitting Set Theorem) *Let* U *be a splitting set of a program* P*. A set* S *is an answer set of* P *iff* $S = X \cup Y$ *for some solution* $\langle X, Y \rangle$ *of* P *w.r.t.* U*.*

Splitting a Normal Logic Program

Lifschitz and Turner's splitting set theorem requires the program to be split by a splitting set. However in many applications, the empty set and the set of all atoms are the only two splitting sets of the program, like program P_1 in Example 1, then the program cannot be divided with the splitting method, which limits the applicability of the theorem.

Here we extend the splitting set theorem for NLPs to allow the program to be split by an arbitrary set of atoms. We want to show that an NLP P can be split by a set U of atoms into the "bottom" part P_b and the "top" part P_t, while new atoms might be introduced in P_t. Then an answer set of P can be computed from an answer set X of P_b and an answer set Y of a program constructed from P_t using X.

First, we introduce some notions. Let U, X be sets of atoms, P an NLP, $b_U(P)$ and $e_U(P, X)$ defined the same as the notions in the previous section. $EC_U(P)$ denotes the set

$$\{p \leftarrow not\, p'.\ \ p' \leftarrow not\, p. \mid p \in Atoms(b_U(P)) \setminus U\},$$

where p' is a new atom *w.r.t.* the atom p. $ECC_U(P, X)$ denotes the set

$$\{\leftarrow not\, p. \mid p \in Atoms(b_U(P)) \setminus U \text{ and } p \in X\}$$
$$\cup \{\leftarrow p. \mid p \in Atoms(b_U(P)) \setminus U \text{ and } p \notin X\}$$

Note that, if U is a splitting set, then $Atoms(b_U(P)) \setminus U = \emptyset$ and $EC_U(P) = ECC_U(P, X) = \emptyset$.

Intuitively, for every atom p that occurs in $b_U(P)$ but not in U, $EC_U(P)$ acts as the set of disjunctions, $p \vee not\, p$. $ECC_U(P, X)$ states that, for such atom p, p is required to be true iff p appears in X. Then we have the proposition.

Proposition 1 *Let P be an NLP and U a set of atoms. A set $S \subseteq Atoms(P)$ satisfies P iff $S = (X \cup Y) \cap Atoms(P)$ for some sets X and Y s.t.*

- *X satisfies $b_U(P) \cup EC_U(P)$, and*
- *Y satisfies $e_U(P \setminus b_U(P), X) \cup ECC_U(P, X)$.*

Example 1 (Continued) *We use $U = \{a\}$ to split the program P_1, then*

$$b_U(P_1) = \{a \leftarrow not\, d.\ \ a \leftarrow c, d.\},$$

$$EC_U(P_1) = \{c \leftarrow not\, c'.\ \ c' \leftarrow not\, c.$$

$$d \leftarrow not\, d'.\ \ d' \leftarrow not\, d.\}.$$

Let $X = \{a, c, d'\}$ which satisfies $b_U(P_1) \cup EC_U(P_1)$, then

$$e_U(P_1 \setminus b_U(P_1), X) = \{d \leftarrow not\, c.\ \ c \leftarrow .\},$$

$$ECC_U(P_1, X) = \{\leftarrow not\, c.\ \ \leftarrow d.\}.$$

Let $Y = \{c\}$ which satisfies $e_U(P_1 \setminus b_U(P_1), X) \cup ECC_U(P_1, X)$, then the set $X \cup Y = \{a, c, d'\}$ satisfies P_1.

Now we consider how loop formulas of loops are influenced by splitting the program.

Let P be an NLP and U a set of atoms, the *in-rules* of P *w.r.t.* U, denoted by $in_U(P)$, is the set of rules $r \in P$ *s.t.* $head(r) \cap U \neq \emptyset$ and $(body^+(r) \cup head(r)) \not\subseteq U$; the *out-rules* of P *w.r.t.* U, denoted by $out_U(P)$, is the set of rules $r \in P$ *s.t.* $(body^+(r) \cup head(r)) \cap U \neq \emptyset$ and $head(r) \not\subseteq U$. P is an NLP, then for each $r \in P$ either $head(r) \cap U \neq \emptyset$ or $head(r) \not\subseteq U$, but not both. So $in_U(P) \subseteq b_U(P)$ and $out_U(P) \subseteq P \setminus b_U(P)$.

A nonempty set E of atoms is a *semi-loop* of P *w.r.t.* U if there exists a loop L of P *s.t.* $E = L \cap U$ and $E \subset L$. Let X be a set of atoms, we denote

$$SL_U(P, X) = \{E \mid E \text{ is a semi-loop of } P \text{ w.r.t. } U,$$

$$E \subseteq X \text{ and } R^-(E, P, X) \subseteq in_U(P)\}.$$

Intuitively, a semi-loop E is in $SL_U(P, X)$, if there exists a loop L of P *s.t.* $E \subset L$ and $LF(L, P)$ is not satisfied by X.

Example 1 (Continued) *Let $U = \{a\}$, $X = \{a, c, d'\}$, and $X' = \{a, c, d\}$,*

$$in_U(P_1) = \{a \leftarrow c, d.\}, \ \ out_U(P_1) = \{c \leftarrow a.\}.$$

$\{a\}$ is the only semi-loop of P_1 w.r.t. U, $R^-(\{a\}, P_1, X) = \{a \leftarrow not\, d.\}$ and $R^-(\{a\}, P_1, X') = \{a \leftarrow c, d.\}$, then $SL_U(P_1, X) = \emptyset$ and $SL_U(P_1, X') = \{\{a\}\}$.

Let P be an NLP and U, X sets of atoms, the *top* of P *w.r.t.* U under X, denoted by $t_U(P, X)$, is the union of the following sets of rules:

- $P \setminus (b_U(P) \cup out_U(P))$,
- $\{\, x_E \leftarrow body(r) \mid r \in in_U(P)$ and $r \in R^-(E, P, X)\,\}$, for each $E \in SL_U(P, X)$,
- $\{\, head(r) \leftarrow x_{E_1}, \ldots, x_{E_t}, body(r) \mid r \in out_U(P)$, for all possible $E_i \in SL_U(P, X)$ $(1 \leq i \leq t)$ *s.t.* $body^+(r) \cap E_i \neq \emptyset\,\}$,

where x_E's are new atoms *w.r.t.* $E \in SL_U(P, X)$. Note that, if $SL_U(P, X) = \emptyset$, then $t_U(P, X) = P \setminus b_U(P)$. A *solution* of P *w.r.t.* U is a pair $\langle X, Y\rangle$ of sets of atoms *s.t.*

- X is an answer set of $b_U(P) \cup EC_U(P)$,
- Y is an answer set of $e_U(t_U(P, X), X) \cup ECC_U(P, X)$.

Example 1 (Continued) *Let $U = \{a\}$, $X = \{a, c, d'\}$, and $X' = \{a, c, d\}$, then*

$$t_U(P_1, X) = \{d \leftarrow not\, c.\ \ c \leftarrow a.\},$$

$$t_U(P_1, X') = \{d \leftarrow not\, c.\ \ x_{\{a\}} \leftarrow c, d.\ \ c \leftarrow x_{\{a\}}, a.\}.$$

The pair $\langle X, \{c\}\rangle$ is a solution of P w.r.t. U. Meanwhile, $e_U(t_U(P, X'), X') \cup ECC_U(P, X')$ is the program:

$$d \leftarrow not\, c.\ \ x_{\{a\}} \leftarrow c, d.\ \ c \leftarrow x_{\{a\}}.\ \ \leftarrow not\, c.\ \ \leftarrow not\, d.$$

which does not have an answer set.

Lemma 1 *For any NLP P and set U of atoms, if $\langle X, Y\rangle$ is a solution of P w.r.t. U and $SL_U(P, X) = \emptyset$, then $(X \cup Y) \cap Atoms(P)$ is an answer set of P.*

Intuitively, $SL_U(P, X) = \emptyset$, then there does not exist a loop L of P *s.t.* $L \cap U \neq \emptyset$, $L \cap (Atoms(P) \setminus U) \neq \emptyset$, and $X \cup Y \not\models LF(L, P)$. So $(X \cup Y) \cap Atoms(P)$ satisfies loop formulas for all loops of P. From Proposition 1 and Lin and Zhao's theorem, $(X \cup Y) \cap Atoms(P)$ is an answer set of P.

Theorem 3 *Let P be an NLP and U a set of atoms. A set S is an answer set of P iff $S = (X \cup Y) \cap Atoms(P)$ for some solution $\langle X, Y\rangle$ of P w.r.t. U.*

We omit the proof here due to the limit of space.

If U is a splitting set of P, then for any set X of atoms, $SL_U(P, X) = \emptyset$, $t_U(P, X) = P \setminus b_U(P)$, and $EC_U(P) = ECC_U(P, X) = \emptyset$. Theorem 3 extends Lifschitz and Turner's splitting set theorem to allow the program to be split by arbitrary sets of atoms.

Computational Complexity Issues

In the previous section, we have provided a new splitting method to split an NLP into parts. The result extends Lifschitz and Turner's splitting set theorem and has its own theoretical interests in the investigation of the answer set semantics as illustrated in next sections. Here we discuss some computational complexity issues related to the splitting method.

When an NLP P is split by a set U of atoms and X is an answer set of the "bottom" part $b_U(P) \cup EC_U(P)$, two main issues that have impact on the computational complexity of the splitting process are:

(i) rules in $EC_U(P)$ would introduce $2^{|Atoms(b_U(P)) \setminus U|}$ number of answer sets for the program $b_U(P) \cup EC_U(P)$,

(ii) the size of $|SL_U(P, X)|$ would be exponential in the number of atoms in P and $t_U(P)$ would introduce an exponential number of new atoms (in the form of x_E).

For the issue (i), there are two cases that could release the computational complexity:

- the size of $|Atoms(b_U(P)) \setminus U|$ is small,
- atoms in U are satisfied by every answer set of P.

When the size of $|Atoms(b_U(P)) \setminus U|$ is small, the size of $2^{|Atoms(b_U(P)) \setminus U|}$ is not too large, then computing all answer sets for the program $b_U(P) \cup EC_U(P)$ is tractable.

When atoms in U are satisfied by every answer set of P, we only need to consider the answer set X of the program $b_U(P) \cup EC_U(P)$ *s.t.* $U \subseteq X$, which would greatly reduce the number of answer sets that need to be considered in the splitting process. Moreover, in a next section, we will show that this case is related to the problem of program simplification and a new simplifying method would be introduced using the concept of splitting.

For the issue (ii), we first introduce two useful cases where $SL_U(P, X) = \emptyset$.

A program P is *tight* if every loop of P is a singleton. A set U of atoms is called a *separating set* of a program P, if there does not exist a loop L of P *s.t.* $L \not\subseteq U$ and $L \cap U \neq \emptyset$.

Proposition 2 *For any NLP P and set U of atoms, if P is tight or U is a separating set of P, then $SL_U(P, X) = \emptyset$.*

Now we show that not all semi-loops in $SL_U(P, X)$ are necessary for the splitting process. We introduce a subset of $SL_U(P, X)$ and refine the splitting method by considering these semi-loops only.

Let P be an NLP and U a set of atoms, a semi-loop E of P *w.r.t.* U is *dominated* by another semi-loop E' of P *w.r.t.* U, if $E \subset E'$, $E \cap head(in_U(P)) = E' \cap head(in_U(P))$, and $E \cap body^+(out_U(P)) = E' \cap body^+(out_U(P))$. Intuitively, if E is dominated by E', then in the "top" part of the program, the loop formulas of loops relative to E could be entailed from the loop formulas of loops relative to E'.

Let P be a program, U and X sets of atoms, we denote

$DSL_U(P, X) = \{E \mid E \in SL_U(P, X)$ and there does not exist another $E' \in SL_U(P, X)$ s.t. E is dominated by $E'\}$.

Clearly, $DSL_U(P, X) \subseteq SL_U(P, X)$. Moreover, we have the following proposition.

Proposition 3 *Let P be an NLP and U, X sets of atoms. A semi-loop $E \in DSL_U(P, X)$ iff E is the union of all possible semi-loop $E' \in SL_U(P, X)$ s.t. $E \cap head(in_U(P)) = E' \cap head(in_U(P))$ and $E \cap body^+(out_U(P)) = E' \cap body^+(out_U(P))$.*

Then we provide a new splitting method based on $DSL_U(P, X)$. Let P be an NLP and U, X sets of atoms, the *dominative top* of P *w.r.t.* U under X, denoted by $dt_U(P, X)$, is the union of the following sets of rules:

- $P \setminus (b_U(P) \cup out_U(P))$,
- $\{\, x_E \leftarrow body(r) \mid r \in in_U(P)$ and $r \in R^-(E, P, X)\,\}$, for each $E \in DSL_U(P, X)$,
- $\{\, head(r) \leftarrow x_{E_1}, \ldots, x_{E_t}, body(r) \mid r \in out_U(P)$, for all possible $E_i \in DSL_U(P, X)$ $(1 \leq i \leq t)$ *s.t.* $body^+(r) \cap E_i \neq \emptyset\,\}$,

where x_E's are new atoms *w.r.t.* $E \in DSL_U(P, X)$.

A *dominative solution* of P *w.r.t.* U is a pair $\langle X, Y\rangle$ of sets of atoms *s.t.*

- X is an answer set of $b_U(P) \cup EC_U(P)$,
- Y is an answer set of $e_U(dt_U(P, X), X) \cup ECC_U(P, X)$.

Theorem 4 *Let P be an NLP and U a set of atoms. A set S is an answer set of P iff $S = (X \cup Y) \cap Atoms(P)$ for some dominative solution $\langle X, Y\rangle$ of P w.r.t. U.*

Note that $|DSL_U(P, X)| \leq (2^{|head(in_U(P))|} - 1) \times (2^{|body^+(out_U(P))|} - 1)$. $dt_U(P, X)$ would introduce an exponential number of new atoms in the worst case. However, if $|head(in_U(P))|$ and $|body^+(out_U(P))|$ are small, then the size of $|DSL_U(P, X)|$ would not be too large and computing the answer sets of the program $e_U(dt_U(P, X), X) \cup ECC_U(P, X)$ would be tractable.

We provide Algorithm 1 for computing $DSL_U(P, X)$. Intuitively, for every possible combination of nonempty subsets of $head(in_U(P))$ and $body^+(out_U(P))$, $dsl_U(P, X)$ tries to find out the largest semi-loop $E \in SL_U(P, X)$ that contains the combination.

Algorithm 1: $dsl_U(P, X)$

```
dsl := ∅;
for each pair of nonempty sets S1 ⊆ head(in_U(P))
      and S2 ⊆ body+(out_U(P)) do
    G_P^S := the S1 ∪ S2 ∪ (Atoms(P) \ (head(in_U(P)) ∪
    body+(out_U(P)))) induced subgraph of G_P;
    L := the Strongly Connected Component (SCC) of G_P^S
    s.t. S1 ∪ S2 ⊆ L;
    if L does not exist then
        break
    if L ∩ U ⊆ X and R−(L ∩ U, P, X) ⊆ in_U(P) then
        append L ∩ U to dsl
    else
        S := head(R−(L ∩ U, P, X) \ in_U(P)) ∪
        ((L ∩ U) \ X);
        G_P^S := the L \ S induced subgraph of G_P;
        goto 5
return dsl
```

Proposition 4 *Let P be an NLP and U, X sets of atoms. $dsl_U(P, X)$ returns $DSL_U(P, X)$ in $O(m2^c)$, where m is the number of atoms in U and c is the number of atoms in $head(in_U(P)) \cup body^+(out_U(P))$.*

Once the splitting process for a set U of a program P is efficient, instead of directly computing an answer set of P, we can first compute an answer set X of $b_U(P) \cup EC_U(P)$ and an answer set Y of $e_U(dt_U(P, X), X) \cup ECC_U(P, X)$, then construct the answer set of P from $X \cup Y$. We show that for some typical programs the later approach is more efficient.

We try Niemelä's (1999) encoding of the Hamiltonian Circuit (HC) problem[1] and consider graphs with two parts, A and B, *s.t.* there is exactly one arc from a node in A to a node in B, and one arc from a node in B to a node in A. Then any HC of the graph must go through these two arcs.

Let P be the resulting program of the HC problem for such a graph, and U be the set of atoms corresponding to nodes and arcs in part A. Then the splitting process for P under U is efficient.[2] In particular, $|Atoms(b_U(P)) \setminus U| = 3$, every atom in $Atoms(b_U(P)) \setminus U$ is satisfied in every answer set of P, and $|DSL_U(P, X)| = 1$ for any possible X. Moreover, every answer set X of $b_U(P) \cup \{p \leftarrow . \mid p \in Atoms(b_U(P)) \setminus U\}$ would lead to a dominative solution of P *w.r.t.* U.

Table 1 contains the running times for these programs split in such way.[3] We consider 2-N as graphs with 2 copies of the complete graph with N nodes and with exactly one arc from each other. For each 2-N entry in the table, we randomly create 10 different such graphs, and the reported times refer the average times for the resulting 10 programs. The numbers under "whole" refer to the running times (in

[1] The structure of a graph for the HC problem is similar to the structure of the positive dependency graph of the logic program.

[2] http://ss.sysu.edu.cn/%7ewh/splitting.html

[3] Our experiments were done on a Linux machine with AMD A10-5800K (3.8GHz) CPU and 3.3GB RAM. The times are in CPU seconds as reported by Linux "usr/bin/time" command.

seconds) of clasp (version 3.1.0 (Gebser et al. 2007)) for the whole programs. The numbers under "bottom" (resp. "top") refer to the running times of clasp for the bottom (resp. top) part of the programs split by the corresponding set U. Clearly, the numbers under "bottom + top" refer to the running times of computing an answer set of the programs using the splitting approach. As can be seen, it is more efficient to use the splitting approach for these programs.

Table 1: Comparing two ways of computing an answer set

Problem	whole	bottom	top	bottom+top
2-10	0.03	0.01	0.02	0.03
2-15	0.53	0.03	0.16	0.19
2-20	1.84	0.03	0.68	0.71
2-25	5.56	0.07	2.25	2.31
2-30	14.07	0.15	5.66	5.81
2-35	27.16	0.23	12.61	12.83
2-40	52.53	0.32	23.61	23.93
2-45	106.93	0.59	48.73	49.32
2-50	171.94	0.75	80.95	81.70

Splitting a Disjunctive Logic Program

In this section, we extend the splitting method to DLPs. Let P be a DLP and U, X sets of atoms, the notions of $b_U(P)$, $e_U(P,X)$, $EC_U(P)$, $ECC_U(P,X)$, $in_U(P)$, $out_U(P)$, $SL_U(P,X)$, and $DSL_U(P,X)$ are defined the same. Different from NLPs, for some DLPs P, $in_U(P) \cap out_U(P) \neq \emptyset$.

The *top* of a DLP P *w.r.t.* U under X, denoted by $t_U(P,X)$, is the union of the following sets of rules:

- $P \setminus (b_U(P) \cup out_U(P))$,
- $\{\{x_E\} \cup head(r) \setminus E \leftarrow body(r) \mid r \in in_U(P)$ and $r \in R^-(E,P,X)\}$, for each $E \in SL_U(P,X)$,
- $\{\, head(r) \leftarrow x_{E_1}, \ldots, x_{E_t}, body(r) \mid r \in out_U(P)$, for all possible $E_i \in SL_U(P,X)$ $(1 \leq i \leq t)$ *s.t.* $body^+(r) \cap E_i \neq \emptyset\,\}$,

where x_E's are new atoms *w.r.t.* $E \in SL_U(P,X)$.

Example 3 *Let $U = \{a\}$ and P_3 be the program:*

$$a \vee d \leftarrow . \qquad d \leftarrow not\ c. \qquad a \leftarrow c, d. \qquad c \leftarrow a.$$

G_{P_3} is the same as G_{P_1} in Figure 1. Then

$$b_U(P_3) = in_U(P_3) = \{a \vee d \leftarrow .\ \ a \leftarrow c, d.\},$$

$$out_U(P_3) = \{a \vee d \leftarrow .\ \ c \leftarrow a.\}.$$

$X = \{a, c, d\}$ is an answer set of $b_U(P_3) \cup EC_U(P_3)$, then $R^-(\{a\}, P_3, X) = \{a \leftarrow c, d\}$, $SL_U(P_3, X) = \{\{a\}\}$. So $t_U(P_3, X) = \{d \leftarrow not\ c.\ \ x_{\{a\}} \leftarrow c, d.\ \ c \leftarrow x_{\{a\}}, a.\}$ and $e_U(t_U(P_3, X), X) \cup ECC_U(P_3, X)$ is the program: $\{d \leftarrow not\ c.\ \ x_{\{a\}} \leftarrow c, d.\ \ c \leftarrow x_{\{a\}}.\quad \leftarrow not\ c.\quad \leftarrow not\ d.\}$ which does not have an answer set.

The *dominative top* of a DLP P *w.r.t.* U under X, denoted by $dt_U(P,X)$, is the union of above sets of rules by replacing every occurrence of $SL_U(P,X)$ with $DSL_U(P,X)$. Then the notions of *solutions* and *dominative solutions* of P w.r.t U are defined the same as in previous sections.

Theorem 5 *Let P be a DLP and U a set of atoms. A set S is an answer set of P iff $S = (X \cup Y) \cap Atoms(P)$ for some solution $\langle X, Y\rangle$ of P w.r.t. U iff $S = (X \cup Y) \cap Atoms(P)$ for some dominative solution $\langle X, Y\rangle$ of P w.r.t. U.*

The computational complexity issues related to the splitting method for DLPs are also similar to the issues discussed above. Proposition 3 and 4 still hold for DLPs.

Strong Splitting a Logic Program

In (Lifschitz and Turner 1994), a logic program P is split by a splitting set U into two parts, $b_U(P)$ and $P \setminus b_U(P)$. For any program P' that does not contain atoms in U, the answer sets of $P \cup P'$ can be computed from the answer sets of $b_U(P)$ and the answer sets of a program constructed from $(P \setminus b_U(P)) \cup P'$. However, the property does not hold for arbitrary sets of atoms in our splitting method. In this section, we propose a new splitting method in which the property holds for all possible sets of atoms.

In particular, when U is a splitting set of the program, we have the following proposition.

Proposition 5 *Let U be a splitting set of a program P and P' a program s.t. $Atoms(P') \cap U = \emptyset$. A set S is an answer set of $P \cup P'$ iff $S = X \cup Y$ for some answer set X of $b_U(P)$ and some answer set Y of $e_U(P \setminus b_U(P), X) \cup P'$.*

Now we introduce the *strong splitting method*. Let P be a DLP and U, X sets of atoms, we denote

$$SS_U(P,X) = \{E \mid E \text{ is a nonempty subset of } U,$$
$$E \subseteq X \text{ and } R^-(E,P,X) \subseteq in_U(P)\}.$$

Intuitively, $SS_U(P,X)$ extends $SL_U(P,X)$ from semi-loops of P *w.r.t.* U to all possible subsets of U. Note that, for every program P' *s.t.* $Atoms(P') \cap U = \emptyset$, $b_U(P \cup P') = b_U(P)$ and for every set X of atoms, $SL_U(P \cup P', X) \subseteq SS_U(P,X)$. If U is a splitting set of P, then for any set X, $SS_U(P,X) = \emptyset$.

We define the *strong top* of P *w.r.t.* U under X, denoted by $st_U(P,X)$, to be the union of the following sets of rules:

- $P \setminus (b_U(P) \cup out_U(P))$,
- $\{\{x_E\} \cup head(r) \setminus E \leftarrow body(r) \mid r \in in_U(P)$ and $r \in R^-(E,P,X)\}$, for each $E \in SS_U(P,X)$,
- $\{\, head(r) \leftarrow x_{E_1}, \ldots, x_{E_t}, body(r) \mid r \in out_U(P)$, for all possible $E_i \in SS_U(P,X)$ $(1 \leq i \leq t)$ *s.t.* $body^+(r) \cap E_i \neq \emptyset\,\}$,

where x_E's are new atoms *w.r.t.* $E \in SS_U(P,X)$.

A *strong solution* of P *w.r.t.* U is a pair $\langle X, Y\rangle$ of sets of atoms *s.t.*

- X is an answer set of $b_U(P) \cup EC_U(P)$,
- Y is an answer set of $e_U(st_U(P,X), X) \cup ECC_U(P,X)$.

We show that, such strong splitting method also generalizes Lifschitz and Turner's splitting set theorem.

Theorem 6 *Let P be a DLP and U a set of atoms. A set S is an answer set of P iff $S = (X \cup Y) \cap Atoms(P)$ for some strong solution $\langle X, Y\rangle$ of P w.r.t. U.*

We show that a similar property of Proposition 5 holds for arbitrary sets U of atoms in this strong splitting method.

Theorem 7 *Let P be a DLP, U a set of atoms, and P' a program s.t. $Atoms(P') \cap U = \emptyset$. A set S is an answer set of $P \cup P'$ iff $S = (X \cup Y) \cap Atoms(P)$ for some answer set X of $b_U(P) \cup EC_U(P)$ and some answer set Y of $e_U(st_U(P,X), X) \cup ECC_U(P,X) \cup P'$.*

Application: Program Simplification

A *consequence* of a program is a set of literals that are satisfied by every answer set of the program. The problem

of program simplification considers how to simplify the program by a consequence, so that the answer sets of the program can be computed from the answer sets of the resulting program with the consequence. In this section, we investigate the problem using the concept of splitting.

Given a literal l, the *complement* of l, written $\bar{l}$ below, is $\neg a$ if l is a and a if l is $\neg a$, where a is an atom. For a set L of literals, we let $\overline{L} = \{\, \bar{l} \mid l \in L \,\}$.

Let set L of literals be a consequence of a program P, we define $tr_n(P, L)$ to be the program obtained from P by

1. deleting each rule r that has an atom $p \in body^+(r)$ with $\neg p \in L$, and
2. replacing each rule r that has an atom $p \in head(r)$ or $p \in body^-(r)$ with $\neg p \in L$ by the rule
$$head(r) \setminus \overline{L} \leftarrow body^+(r), body^-(r) \setminus \overline{L}.$$

We define $tr_p(P, L)$ to be the program obtained from P by

1. deleting each rule r that has an atom $p \in head(r)$ or $p \in body^-(r)$ with $p \in L$, and
2. replacing each rule r that has an atom $p \in body^+(r)$ with $p \in L$ by the rule
$$head(r) \leftarrow body^+(r) \setminus L, body^-(r).$$

Note that $tr_n(P, L)$ (resp. $tr_p(P, L)$) does not contain any atom p with $\neg p \in L$ (resp. $p \in L$) and $tr_p(tr_n(P, L), L)$ does not contain any atoms occurring in L.

The well-founded model (Van Gelder, Ross, and Schlipf 1991) of an NLP is a set of literals and it is also a consequence of the program.

Proposition 6 *Let L be a well-founded model of an NLP P. A set S is an answer set of P iff $S \setminus L$ is an answer set of $tr_p(tr_n(P, L), L)$.*

As a result, in all current ASP solvers, an NLP is first simplified by its well-founded model. Meanwhile, there are other consequences of a program that are larger than the well-founded model and can be computed efficiently. For instant, Chen, Ji, and Lin (2013) computes such a consequence using loop formulas of loops with at most one external support rule. However, Proposition 6 does not hold in general for these consequences.

We show that $tr_n(P, L)$ simplifies P in general and $tr_p(P, L)$ simplifies P for only some special cases.

Proposition 7 *Let L be a consequence of a program P.*

- *A set S is an answer set of P iff S is an answer set of $tr_n(P, L)$.*
- *If for every atom $p \in L$ there is a rule $p \leftarrow .$ in P, then a set S is an answer set of P iff $S \setminus L$ is an answer set of $tr_p(P, L)$.*
- *A set S is an answer set of P implies $S \setminus L$ is an answer set of $tr_p(P, L)$, but not vice versa in general.*

Intuitively, for any loop E of a program P and any consequence L of P, if E is not a loop of $tr_n(P, L)$, then the answer sets of $tr_n(P, L)$ still satisfy the loop formula of E under P; if E is not a loop of $tr_p(P, L)$, then the answer sets of $tr_p(P, L)$ may not satisfy the loop formula of E.

Example 4 *Consider the logic program P_4:*

$a \leftarrow b.$ $\quad c \leftarrow a.$ $\quad b \leftarrow c.$ $\quad c \leftarrow d.$ $\quad a \leftarrow f.$

$d \leftarrow not\, e.$ $\quad e \leftarrow not\, d.$ $\quad \leftarrow not\, a.$ $\quad f \leftarrow a.$

$L = \{a, f\}$ is a consequence of P_4, then $tr_p(P_4, L)$ is:

$c \leftarrow .$ $\quad b \leftarrow c.$ $\quad c \leftarrow d.$ $\quad d \leftarrow not\, e.$ $\quad e \leftarrow not\, d.$

The only answer set of P_4 is $\{a, b, c, d, f\}$. However, $tr_p(P_4, L)$ has two answer sets: $\{b, c, d\}$ and $\{b, c, e\}$.

Let set U of atoms be a consequence of a program P, we can use U to split P that results a program that does not contain atoms in U. Then using the concept of splitting, we provide a new approach to simplify a program by a set of atoms. Let P be a DLP and U a set of atom, we denote

$$CS_U(P) = \{E \mid E \text{ is a nonempty subset of } U \text{ and } R^-(E, P) \subseteq in'_U(P)\},$$
$$in'_U(P) = \{r \in P \mid head(r) \cap U \neq \emptyset, Atoms(r) \not\subseteq U\}.$$

We define the *consequence top* of P w.r.t U, denoted by $ct_U(P)$, to be the union of the following sets of rules:

- $P \setminus (b_U(P) \cup out_U(P))$,
- $\{\{x_E\} \cup head(r) \setminus E \leftarrow body(r) \mid r \in in'_U(P)$ and $r \in R^-(E, P)\}$, for each $E \in CS_U(P)$,
- $\{head(r) \leftarrow x_{E_1}, \ldots, x_{E_t}, body(r) \mid r \in out_U(P)$, for all possible $E_i \in CS_U(P)$ $(1 \leq i \leq t)$ *s.t.* $body^+(r) \cap E_i \neq \emptyset\}$,
- $\{\leftarrow not\, x_E\}$, for each $E \in CS_U(P)$,

where x_E's are new atoms *w.r.t.* $E \in CS_U(P)$.

Proposition 8 *Let set U of atoms be a consequence of an NLP P. A set S is an answer set of P iff there is an answer set S^* of $tr_p(ct_U(P), U)$ s.t. $S \setminus U = S^* \cap Atoms(P)$.*

$tr_p(ct_U(P), U)$ may introduce new atoms for elements in $CS_U(P)$. Similar to the discussion in the section for computational complexity issues, when $|CS_U(P)|$ is small, $tr_p(ct_U(P), U)$ would be simpler than the original program P.[4] Note that, Proposition 8 may not be true for DLPs.

Example 4 (Continued) *Let $U = \{a, f\}$, then $CS_U(P_4) = \{\{a, f\}\}$ and $ct_U(P_4)$ is:*

$b \leftarrow c.$ $\quad c \leftarrow d.$ $\quad d \leftarrow not\, e.$ $\quad e \leftarrow not\, d.$

$\leftarrow not\, a.$ $\quad x_{\{a,f\}} \leftarrow b.$ $\quad c \leftarrow x_{\{a,f\}}, a.$ $\quad \leftarrow not\, x_{\{a,f\}}.$

Then $tr_p(ct_U(P_4), U)$ is:

$b \leftarrow c.$ $\quad c \leftarrow d.$ $\quad d \leftarrow not\, e.$ $\quad e \leftarrow not\, d.$

$x_{\{a,f\}} \leftarrow b.$ $\quad c \leftarrow x_{\{a,f\}}.$ $\quad \leftarrow not\, x_{\{a,f\}}.$

The only answer set of $tr_p(ct_U(P_4), U)$ is $\{b, c, d, x_{\{a,f\}}\}$.

Conclusion

We summarize the contribution of this paper here. First, we provide a new method to split a program into two parts by an arbitrary set of atoms, while new atoms might be introduced in the process. We show that the task of computing the answer sets of the program can be converted into the tasks of computing the answer sets of these parts. The result extends Lifschitz and Turner's splitting theorem. Second, we illustrate the usefulness of the splitting method through two applications. We show that for some typical programs the splitting process is efficient, which could help ASP solvers to compute an answer set of the program. We also show that the program simplification problem can be regarded as a splitting program, which results a new approach to simplify a program. We expect that, the idea of splitting will find many other uses, for instance in the investigation of incremental ASP solvers and forgetting.

[4] http://ss.sysu.edu.cn/%7ewh/splitting.html

Acknowledgments

We thank the reviewers for their comments and suggestions for improving the paper. We are grateful to Fangzhen Lin for many helpful and informative discussions. We would also like to thank Xiaoping Chen and his research group for their useful discussions. Jianmin Ji's research was partially supported by the Fundamental Research Funds for the Central Universities under grant WK0110000035, the National Natural Science Foundation of China under grant 61175057, the National Natural Science Foundation for the Youth of China under grant 61403359, as well as the USTC Key Direction Project and the USTC 985 Project. Hai Wan thanks Research Fund for the Doctoral Program of Higher Education of China (No. 20110171120041), Natural Science Foundation of Guangdong Province of China (No. S2012010009836), and Guangzhou Science and Technology Project (No. 2013J4100058) for the support of this research. We also thank supports from the National Natural Science Foundation of China under grant 61370161.

References

Chen, X.; Ji, J.; and Lin, F. 2013. Computing loops with at most one external support rule. *ACM Transactions on Computational Logic (TOCL)* 14(1):3–40.

Dao-Tran, M.; Eiter, T.; Fink, M.; and Krennwallner, T. 2009. Modular nonmonotonic logic programming revisited. In *Logic Programming*. Springer. 145–159.

Ferraris, P.; Lee, J.; Lifschitz, V.; and Palla, R. 2009. Symmetric splitting in the general theory of stable models. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, 797–803.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 386–392.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2008. Engineering an incremental asp solver. In *Logic Programming*. Springer. 190–205.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New generation computing* 9(3-4):365–385.

Lee, J., and Lifschitz, V. 2003. Loop formulas for disjunctive logic programs. In *Proceedings of the 19th International Conference on Logic Programming (ICLP-03)*, 451–465.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming (ICLP-94)*, 23–37.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1-2):115–137.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3-4):241–273.

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.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)* 38(3):619–649.