Strong Equivalence of Logic Programs with Abstract Constraint Atoms

Guohua Liu[1], Randy Goebel[2], Tomi Janhunen[1],
Ilkka Niemelä[1], and Jia-Huai You[2]

[1] Aalto University, Department of Information and Computer Science
{Guohua.Liu,Tomi.Janhunen,Ilkka.Niemela}@aalto.fi
[2] University of Alberta, Department of Computing Science
{goebel,you}@cs.ualberta.ca

Abstract. Logic programs with abstract constraint atoms provide a unifying framework for studying logic programs with various kinds of constraints. Establishing strong equivalence between logic programs is a key property for program maintenance and optimization, and for guaranteeing the same behavior for a revised original program in any context. In this paper, we study strong equivalence of logic programs with abstract constraint atoms. We first give a general characterization of strong equivalence based on a new definition of program reduct for logic programs with abstract constraints. Then we consider a particular kind of program revision—constraint replacements addressing the question: under what conditions can a constraint in a program be replaced by other constraints, so that the resulting program is strongly equivalent to the original one.

1 Introduction

Logic programming interpreted with *answer set* semantics or answer set programming (ASP), is a declarative programming paradigm for knowledge representation, designed for characterizing and solving computationally hard problems [1,2]. In ASP, a problem is represented by a logic program and the answer sets of the program correspond to the solutions to the problem. Answer set programming with *abstract constraint atoms* (*c-atoms*) [3,4,5] provides a unifying framework for the study of logic programs with various constraints, such as weight constraints [6], aggregates [7,8], and global constraints [9].

Strong equivalence within this kind of semantics [10] is one of the key concepts of logic programming. A program (a set of rules) P is *strongly equivalent* to a program Q if, for any other program R, the programs $P \cup R$ and $Q \cup R$ have the same answer sets. In order to see whether a set of rules in a program can always be replaced by another set of rules, regardless of other program components, one needs to check whether the two sets of rules are strongly equivalent. Strongly equivalent programs are guaranteed to have the same behavior in any context. Uniform equivalence [11] is a special case of strong equivalence. Uniformly equivalent programs have the same behavior in any context of facts. Lifschitz et. al. [10] developed a characterization of strong equivalence using the

J. Delgrande and W. Faber (Eds.): LPNMR 2011, LNAI 6645, pp. 161–173, 2011.

logic of here-and-there. Lin [12] presented a transformation by which the strong equivalence of logic programs is converted to classical entailment. Turner [13] provided a model-theoretical characterization of the strong equivalence, where two programs are strongly equivalent if and only if they have the same set of SE-models. Liu and Truszczyński [11] extended this approach to logic programs with monotone constraints.

In this paper, we study the characterization of strong equivalence for logic programs with arbitrary abstract constraint atoms, under the semantics based on conditional satisfaction [5]. We extend the concept of program *reduct* to logic programs with abstract constraint atoms. Using the notion of program reduct, we define SE-models and UE-models in the standard way employed in [11,13] and characterize strong and uniform equivalence by SE-models and UE-models, respectively. Then, we study strong equivalence of a particular class of program revisions, viz. *constraint replacements*, that amount to replacing a constraint in a program by another constraint or a combination of constraints. Constraint replacements can be used as program transformations to decompose a complicated constraint into simpler parts when doing program development or optimization. We note that constraint replacements are also a standard technique to implement complicated constraints in current ASP systems: typically the inference engine of the system supports a limited set of basic constraints and more involved constraints are compiled to basic ones during grounding [6,14]. Here, we are interested in replacements that can be applied in any context, i.e., where the original program and the modified program are strongly equivalent. Strong equivalence is particularly valuable because it allows replacements to be done in either the whole or a part of the program range, while (weak) equivalence only consider the former. We provide fundamental results on replacement operations by presenting criteria under which a constraint can be replaced with a conjunction, a disjunction, or a combination of constraints while preserving strong equivalence. An observation is that replacements with disjunctions are more involved and require an extra condition compared to those with conjunctions.

This paper is organized as follows. The next section reviews the basic definitions of programs with general abstract constraints under the semantics based on conditional satisfaction [5]. In Section 3, we characterize strong equivalence by the SE-models of constraint programs and show that the characterization generalizes to uniform equivalence. Section 4 applies strong equivalence in the study of constraint replacements. In Section 5, we address the interconnections of our results on constraint replacement to existing transformations. Related work is described in Section 6. Finally, we conclude the paper in Section 7.

2 Preliminaries

We review the answer set semantics for logic programs with arbitrary constraint atoms, as defined in [5]. The semantics should also be contributed to [15], where it is defined as a fix-point construction using 3-valued logic. We assume a propositional language with a countable set of propositional *atoms*.

An *abstract constraint atom* (*c-atom*) is a construct of the form (D, C) where D is the *domain* of the c-atom and C the *admissible solution set* of the c-atom. The domain D is a finite set of atoms and the admissible solution set C is a set of subsets of D, i.e., $C \subseteq 2^D$. Given a c-atom $A = (D, C)$, we use A_d and A_c to refer to sets D and C, respectively. Certain special c-atoms have been distinguished. A c-atom of the form $(\{a\}, \{\{a\}\})$ simply denotes a propositional atom a. A c-atom A is *monotone* if for every $X \subseteq Y \subseteq A_d$, $X \in A_c$ implies that $Y \in A_c$, *antimonotone* if for every $X \subseteq Y \subseteq A_d$, $Y \in A_c$ implies that $X \in A_c$, and *convex* if for every $X \subseteq Y \subseteq Z \subseteq A_d$, $X \in A_c$ and $Z \in A_c$ implies $Y \in A_c$.

A logic program with c-atoms, also called a *constraint program* (or *program* for short), is a finite set of rules of the form

$$A \leftarrow A_1, \ldots, A_n. \tag{1}$$

where A and A_i's are c-atoms.

For a program P, we denote by $At(P)$ the set of atoms appearing in P. In general, negative atoms of the form `not` A may appear in a rule. Following [5], a negative c-atom `not` A in a program is interpreted as, and substituted by, its *complement* c-atom $\overline{A}$, where $\overline{A}_d = A_d$ and $\overline{A}_c = 2^{A_d} \setminus A_c$. Due to this assumption, we consider the programs where no c-atoms appear negatively.

For a rule r of the form (1), we define $hd(r) = A$ and $bd(r) = \{A_1, ..., A_n\}$, which are called the *head* and the *body* of r, respectively. A rule r is said to be *basic* if $hd(r)$ is a propositional atom[1]. A program P is *basic* if every rule in it is basic and *normal* if every c-atom in it is a propositional atom.

A set of atoms M *satisfies* a c-atom A, written $M \models A$, if $M \cap A_d \in A_c$. Otherwise M does not satisfy A, written $M \not\models A$. Satisfaction naturally extends to conjunctions and disjunctions of c-atoms.

Answer sets for constraint programs are defined in two steps. First, answer sets for basic programs are defined, based on the notion *conditional satisfaction*. Then the answer sets for general programs are defined.

Definition 1. *Let S and M be sets of atoms such that $S \subseteq M$. The set S* conditionally satisfies *a c-atom A, w.r.t. M, denoted by $S \models_M A$, if $S \models A$ and for every $I \subseteq A_d$ such that $S \cap A_d \subseteq I$ and $I \subseteq M \cap A_d$, we have that $I \in A_c$.*

Example 1. Let A be the c-atom $(\{a, b\}, \{\emptyset, \{a\}, \{a, b\}\})$ and $S_1 = \emptyset$, $S_2 = \{a\}$, and $M = \{a, b\}$. Then, $S_1 \not\models_M A$ and $S_2 \models_M A$. □

Conditional satisfaction extends naturally to conjunctions and disjunctions of c-atoms. Whenever it is clear by the context, we may use a set of c-atoms to denote a conjunction or a disjunction of c-atoms.

An operator T_P is defined as follows: for any sets S, M, and a basic program P, $T_P(S, M) = \{a \mid \exists r \in P,\ hd(r) = a,\ \text{and}\ \ S \models_M bd(r)\}$. The operator T_P is monotonic w.r.t its first argument (given that the second argument is fixed). Answer sets of a basic program P are defined as the (least) fixpoint of T_P.

[1] The head can also be $\bot$, which denotes the c-atom $(D, \emptyset)$. Such a rule serves as a constraint [5]. Rules of this kind are irrelevant for the purposes of this paper.

Definition 2. *Let P be a basic program and M a set of atoms. The set M is an* answer set *of P iff M is a model of P and $M = T_P^\infty(\emptyset, M)$, where $T_P^0(\emptyset, M) = \emptyset$ and $T_P^{i+1}(\emptyset, M) = T_P(T_P^i(\emptyset, M), M)$, for all $i \geq 0$.*

The answer sets of a (general) program are defined on the basis of the answer sets of a basic program—the *instance* of the general program. Let P be a constraint program, r a rule in P of the form (1), and M a set of atoms. The instance of r, with respect to M, is

$$inst(r, M) = \begin{cases} \{a \leftarrow bd(r) \mid a \in M \cap hd(r)_d\}, & \text{if } M \models hd(r); \\ \emptyset, & \text{otherwise.} \end{cases}$$

The instance of P with respect to M, denoted $inst(P, M)$, is the program

$$inst(P, M) = \cup_{r \in P} inst(r, M)$$

Definition 3. *Let P be a program and M a set of atoms. The set M is an answer set of P iff M is an answer set of $inst(P, M)$.*

3 Characterization of Strong and Uniform Equivalence

We first define the reduct of c-atoms and general constraint programs. Then, using the reduct, we define SE-models and characterize the strong equivalence of programs. Finally, we show how these results extend to uniform equivalence.

3.1 Program Reduct

The program reduct plays a very central role in the definition of answer sets for normal programs [1]. However, it is non-trivial to generalize the reduct (e.g. [16]). In what follows, we propose a new way of reducing c-atoms themselves, establish a close connection between conditional satisfaction of c-atoms and satisfaction of reduced c-atoms, and then extend these ideas to cover rules and programs.

Definition 4. *Let A be a c-atom and M a set of atoms. The* reduct *of A, w.r.t. M, denoted A^M, is the c-atom (A_d^M, A_c^M), where $A_d^M = A_d$ and the set of admissible solutions $A_c^M = \{S \mid S \in A_c,\ S \subseteq M,\ \text{and } S \models_M A\}$.*

Proposition 1. *Let A be a c-atom and S and M be sets of atoms such that $S \subseteq M$. Then $S \models_M A$ iff $S \models A^M$.*

Example 2. Let $A = (\{a, b, c\}, \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\})$ be a c-atom. Then, given an interpretation $M = \{a, b\}$, we have $A^M = (\{a, b, c\}, \{\{a\}, \{a, b\}\})$. □

Definition 5. *Let P be a basic program and M a set of atoms. The* reduct *of P, w.r.t. M, denoted P^M, is the program obtained by:*

1. *removing from P any rules whose bodies are not satisfied by M;*
2. *replacing each c-atom with the reduct of the c-atom w.r.t. M, in the bodies of the remaining rules.*

Definition 6. *Let P be a program and M a set of atoms. The reduct of P, w.r.t M, denoted P^M, is the reduct of the instance of P w.r.t. M, i.e., $inst(P, M)^M$.*

3.2 Strong Equivalence

Strong equivalence can be defined in the standard way using the notion of answer sets from Definition 3, independently of the class of programs. Similarly, given Definition 6, the notion of SE-models can be adopted—paving the way for Theorem 1 which characterizes strong equivalence in terms of SE-models.

Definition 7. *Programs P and Q are* strongly equivalent*, denoted $P \equiv_s Q$, iff, for any program R, the programs $P \cup R$ and $Q \cup R$ have the same answer sets.*

Definition 8. *Let P be a program. A pair of sets (X, Y) is a* strong equivalence model (SE-model) *of P if the following conditions hold: (1) $X \subseteq Y$; (2) $Y \models P$; and (3) $X \models P^Y$. The set of SE-models of P is denoted by* $\mathrm{SE}(P)$.

Theorem 1. *Let P and Q be two programs. Then, $P \equiv_s Q$ iff* $\mathrm{SE}(P) = \mathrm{SE}(Q)$.

Proof Sketch. We use $\mathrm{AS}(P)$ to denote the set of answer sets of a program P.

($\Longrightarrow$) Let (X, Y) be an SE-model of P. We show that (X, Y) is also an SE-model of Q, by contradiction. Assume that $Y \not\models Q$. Consider the program $R = \{a \mid a \in Y\}$. We can show that $Y \in \mathrm{AS}(P \cup R)$ and $Y \notin \mathrm{AS}(Q \cup R)$, contradicting $P \equiv_s Q$. Assume $X \not\models Q^Y$. Consider the program $R = \{a \mid a \in X\} \cup \{b \leftarrow c \mid b \in Y \text{ and } c \in S \setminus X\}$ where $S = \{a \mid \text{there is } r \in Q^Y \text{ such that } hd(r) = a \text{ and } X \models bd(r)\}$. We can show that $Y \notin \mathrm{AS}(P \cup R)$ and $Y \in \mathrm{AS}(Q \cup R)$, contradicting $P \equiv_s Q$. So, $(X, Y) \in \mathrm{SE}(Q)$. It follows by symmetry that any SE-model of Q is also an SE-model of P. Therefore $\mathrm{SE}(P) = \mathrm{SE}(Q)$.

($\Longleftarrow$) It is easy to show the following statement: for any programs P and Q, $\mathrm{SE}(P \cup Q) = \mathrm{SE}(P) \cap \mathrm{SE}(Q)$, and if $\mathrm{SE}(P) = \mathrm{SE}(Q)$, then $\mathrm{AS}(P) = \mathrm{AS}(Q)$. So, given $\mathrm{SE}(P) = \mathrm{SE}(Q)$, we have for all programs R, $\mathrm{SE}(P \cup R) = \mathrm{SE}(Q \cup R)$ and $\mathrm{AS}(P \cup R) = \mathrm{AS}(Q \cup R)$. Therefore $P \equiv_s Q$. □

3.3 Uniform Equivalence

The concept of uniform equivalence is closely related to strong equivalence (cf. Definition 7). The essential difference is that in uniform equivalence, the context program R is restricted to be a set of facts. To formalize this, we use a special rule $r_D = (D, \{D\}) \leftarrow$ for any set of atoms D [11]. Adding the rule r_D to a program is then equivalent to adding each atom $a \in D$ as a fact $(\{a\}, \{\{a\}\}) \leftarrow$.

Definition 9. *Programs P and Q are* uniformly equivalent*, denoted $P \equiv_u Q$, iff, for any set of atoms D, $P \cup \{r_D\}$ and $Q \cup \{r_D\}$ have the same answer sets.*

The uniform equivalence of finite programs can be characterized similarly as that in [11] using *uniform equivalence models* which are a special class of SE-models. We consider finite programs only as the uniform equivalence of infinite programs cannot be captured by any class of SE-models in general [17].

Definition 10. *Let P be a program. A pair (X, Y) is a* uniform equivalence model (UE-model) *of P if the following conditions hold: (1) (X, Y) is an SE-model of P; (2) for every SE-model (X', Y) of P such that $X \subseteq X'$, either $X' = X$ or $X' = Y$. The set of UE-models of P is denoted by* $\mathrm{UE}(P)$.

Theorem 2. *For any finite programs P and Q, $P \equiv_u Q$ iff* $\mathrm{UE}(P) = \mathrm{UE}(Q)$.

4 Constraint Replacements

In this section we consider a particular kind of program revision—*constraint replacement*. The idea is that a constraint represented by a c-atom in a logic program is replaced by either (i) a conjunction of constraints, (ii) a disjunction of constraints, or (iii) a combination of them. It is then natural to use strong equivalence as correctness criterion and we establish explicit conditions on which strong equivalence is preserved. As regards notation, we define the *Cartesian* product of two sets of interpretations S_1 and S_2, denoted $S_1 \times S_2$, as the set of interpretations $\{T_1 \cup T_2 \mid T_1 \in S_1 \text{ and } T_2 \in S_2\}$. Using this notion, we are able to define a basic operation for constructing sets of admissible solutions.

Definition 11. *The* extension of the set A_c of admissible solutions *of a c-atom A over a set D of atoms, denoted by $ext(A_c, D)$, is $ext(A_c, D) = A_c \times 2^{(D \setminus A_d)}$.*

Proposition 2. *For a c-atom A, a set D of atoms, and an interpretation M, the extended projection $M \cap (A_d \cup D) \in ext(A_c, D)$ iff $M \models A$.*

Given a rule r of the form (1) and $A_k \in bd(r)$ with $1 \leq k \leq n$, we write $r[A_k/B_1, \ldots, B_m]$ for the result of substituting c-atoms $B_1, \ldots, B_m$ for A_k, i.e.,

$$A \leftarrow A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n. \tag{2}$$

4.1 Conjunctive Encoding

In a conjunctive encoding, the idea is to represent a c-atom A as a conjunction of c-atoms $A_1, \ldots, A_m$ where each A_i may have a subdomain of A_d as its domain.

Definition 12. *A conjunction of c-atoms $A_1, \ldots, A_m$ is a* conjunctive encoding *of a c-atom A, denoted $A = \mathcal{C}(A_1, \ldots, A_m)$, iff the c-atoms satisfy*

1. $A_d = \bigcup_{i=1}^{m} (A_i)_d$; *and*
2. $A_c = \bigcap_{i=1}^{m} ext((A_i)_c, A_d)$.

The conditions of Definition 12 guarantee important properties for conjunctive encodings as detailed below: The (conditional) satisfaction of c-atoms is preserved and the same can be observed for the reducts of c-atoms.

Proposition 3. *If $A = \mathcal{C}(A_1, \ldots, A_m)$, then for any M, N such that $M \subseteq N$:*

1. *$M \models A$ iff $M \models A_i$ for each $1 \leq i \leq m$;*
2. *$M \models_N A$ iff $M \models_N A_i$ for each $1 \leq i \leq m$; and*
3. *$M \models A^N$ iff $M \models (A_i)^N$ for each $1 \leq i \leq m$.*

The properties listed above guarantee that replacing a c-atom in a program by its conjunctive encoding $A_1, \ldots, A_m$ also preserves SE-models. This observation leads us to the following results, at both the rule and program levels.

Theorem 3. *Let r be a rule of the form (1) and A_k a c-atom in the body $bd(r)$. If $A_k = \mathcal{C}(B_1, \ldots, B_m)$, then $\{r\} \equiv_s \{r[A_k/B_1, \ldots, B_m]\}$.*

In the above, the rule $r[A_k/B_1, \ldots, B_m]$ coincides with (2) and we call this particular rule the *conjunctive rewrite* of r with respect to $A_k = \mathcal{C}(B_1, \ldots, B_m)$. Since $\equiv_s$ is a congruence relation, i.e., $P \equiv_s Q$ implies $P \cup R \equiv_s Q \cup R$, we can apply Theorem 3 in any context. In particular, we call a program P' a *one-step conjunctive rewrite* of P iff P' is obtained as $(P \setminus \{r\}) \cup \{r[A_k/B_1, \ldots, B_m]\}$ for $A_k \in bd(r)$ and $A_k = \mathcal{C}(B_1, \ldots, B_m)$. This idea easily generalizes for n steps.

Corollary 1. *For an n-step conjunctive rewrite P' of a program P, $P \equiv_s P'$.*

It is also worth pointing out special cases of conjunctive encodings. If each domain $(A_i)_d$ coincides with A_d, then $ext((A_i)_c, A_d) = (A_i)_c$ and the second condition of Definition 12 implies $A_c = \bigcap_{i=1}^{m} (A_i)_c$. On the other hand, if the domains of each A_i and A_j with $i \neq j$ are mutually disjoint, then the second condition reduces to a Cartesian product $A_c = (A_1)_c \times \ldots \times (A_m)_c$. In other intermediate cases, we obtain a *natural join* $A_c = (A_1)_c \bowtie \ldots \bowtie (A_m)_c$ condition, which was introduced by Janhunen et al. [18] to relate the set of answer sets associated with an entire logic program with those of its component programs.

4.2 Disjunctive Encoding

The idea of a disjunctive encoding is to represent a c-atom A as a disjunction of c-atoms $A_1, \ldots, A_m$. However, in contrast with Definition 12, an additional condition becomes necessary in order to preserve conditional satisfaction of c-atoms and their reducts.

Definition 13. *A disjunction of c-atoms $A_1, \ldots, A_m$ is a* disjunctive encoding *of a c-atom A, denoted $A = \mathcal{D}(A_1, \ldots, A_m)$, iff the c-atoms satisfy*

1. $A_d = \bigcup_{i=1}^{m} (A_i)_d$;
2. $A_c = \bigcup_{i=1}^{m} ext((A_i)_c, A_d)$; *and*
3. *for any subset M of A_d, $A_c^M = \bigcup_{i=1}^{m} (A_i)_c^M$.*

Proposition 4. *If $A = \mathcal{D}(A_1, \ldots, A_m)$, then for any M, N such that $M \subseteq N$:*

1. *$M \models A$ iff $M \models A_i$ for some $1 \leq i \leq m$;*
2. *$M \models_N A$ iff $M \models_N A_i$ for some $1 \leq i \leq m$; and*
3. *$M \models A^N$ iff $M \models (A_i)^N$ for some $1 \leq i \leq m$.*

Because of the general properties of disjunction, we need to be careful about disjunctive encodings when replacing c-atoms in rules. Rewriting a rule r of the form (1) with respect to a disjunctive encoding $A_k = \mathcal{D}(B_1, \ldots, B_m)$ results in m rules $r[A_k/B_1], \ldots, r[A_k/B_m]$ obtained by substituting A_k by each B_i in turn. Proposition 4 guarantees the preservation of strong equivalence.

Theorem 4. *Let r be a rule of the form (1) and A_k a c-atom in the body $bd(r)$. If $A_k = \mathcal{D}(B_1, \ldots, B_m)$, then $\{r\} \equiv_s \{r[A_k/B_1], \ldots, r[A_k/B_m]\}$.*

Hence, in *one-step disjunctive rewriting* based on Theorem 4, a program P with $r \in P$ would be rewritten as $P' = (P \setminus \{r\}) \cup \{r[A_k/B_1], \ldots, r[A_k/B_m]\}$. This also preserves strong equivalence by Theorem 4. In general, we obtain:

Corollary 2. *For an n-step disjunctive rewrite P' of a program P, $P \equiv_s P'$.*

The condition 3 of Definition 13 reveals that, in contrast with conjunctive encodings, conditional satisfaction is not automatically preserved in the disjunctive case. The next example illustrates that the first two conditions that preserve the satisfaction of a c-atom are insufficient to preserve strong equivalence.

Example 3. Let P be the following program with an *aggregate* denoted by A:

$$p(2) \leftarrow A. \qquad p(1) \leftarrow . \qquad p(-3) \leftarrow p(2).$$

The intuitive reading of A is $\mathsf{SUM}(\{X \mid p(X)\}) \neq -1$ and following [5], it corresponds to a c-atom with $A_d = \{p(1), p(3), p(-3)\}$ and $A_c = 2^{A_d} \setminus \{\{p(2), p(-3)\}\}$. It may seem natural to replace A by the disjunction of $A_1 = \mathsf{SUM}(\{X \mid p(X)\}) > -1$ and $A_2 = \mathsf{SUM}(\{X \mid p(X)\}) < -1$ and, therefore, to rewrite P as P':

$$p(2) \leftarrow A_1. \qquad p(2) \leftarrow A_2. \qquad p(1) \leftarrow . \qquad p(-3) \leftarrow p(2).$$

However, the programs P and P' are not strongly equivalent. To check this, consider $M = \{p(1), p(2), p(-3)\}$ which is an answer set of P but not that of P'. This aspect is captured by the third condition of Definition 13 because A_c^M differs from $(A_1)_c^M \cup (A_2)_c^M$ for the interpretation $M = \{p(1), p(2), p(-3)\}$. □

There is also one interesting special case of disjunctive encodings conforming to Definition 13. If the domain of each c-atom A_i coincides with A_d, i.e., $(A_i)_d = A_d$ for each $1 \leq i \leq m$, then $ext((A_i)_c, A_d) = (A_i)_c$ for each $1 \leq i \leq m$ as well. Thus, the sets of admissible solutions are simply related by $A_c = \bigcup_{i=1}^{m} (A_i)_c$.

4.3 Shannon Encodings

Any Boolean function $f(a_1, \ldots, a_n)$ can be expanded with respect to its argument a_i using Shannon's partial evaluation principle:

$$f(a_1, \ldots, a_n) = (a_i \wedge f(a_1, \ldots, a_{i-1}, \top, a_{i+1}, \ldots, a_n)) \vee (\neg a_i \wedge f(a_1, \ldots, a_{i-1}, \bot, a_{i+1}, \ldots, a_n)). \quad (3)$$

The objective of this section is to present Shannon expansion for *monotone* c-atoms. The reason for this restriction is that Shannon's principle cannot be applied to arbitrary c-atoms in a natural way (see Example 5 for details). In the case of monotone c-atoms, however, the following can be established.

Proposition 5. *If A is a monotone c-atom and $a \in A_d$, then it holds that $A = \mathcal{D}(\mathcal{C}(a, A^+(a)), A^-(a))$ where a stands for the c-atom $(\{a\}, \{\{a\}\})$, and*

1. *$A^+(a) = (A_d \setminus \{a\}, \{T \setminus \{a\} \mid T \in A_c$ and $a \in T\})$ and*
2. *$A^-(a) = (A_d \setminus \{a\}, \{T \mid T \in A_c$ and $a \notin T\})$.*

Given this relationship we may call $\mathcal{S}(A,a) = \mathcal{D}(\mathcal{C}(a, A^+(a)), A^-(a))$ as the *Shannon encoding* of A with respect to an atom $a \in A_d$. Intuitively, the Shannon encoding $\mathcal{S}(A,a)$ builds on a case analysis. The part $\mathcal{C}(a, A^+(a))$ captures admissible solutions of A where a is true. The part $A^-(a)$ covers cases where a is false (by default) and hence $(\{a\},\{\emptyset\})$ is not incorporated. We call $A^+(a)$ and $A^-(a)$ the respective *positive* and *negative* encodings of A given $a \in A_d$.

Example 4. Consider a monotone c-atom $A = (\{a,b\},\{\{a\},\{a,b\}\})$ for which $\mathcal{S}(A,a) = \mathcal{D}(\mathcal{C}(a, A^+(a)), A^-(a))$ where the respective positive and negative encodings of A are $A^+(a) = (\{b\},\{\emptyset,\{b\}\})$ and $A^-(a) = (\{b\},\{\})$. It is worth noting that the latter c-atom is never satisfied and if it is used to rewrite any rule body, the resulting rule can be directly omitted due to inapplicability. □

Given a monotone c-atom A, any atom $a \in A_d$ can be used to do the Shannon encoding. When the identity of a is not important, we simply use $\mathcal{S}(A,\cdot)$ to denote the appropriate construction, the properties of which are as follows.

Proposition 6. *If A is a monotone c-atom, then so are $A^+(\cdot)$ and $A^-(\cdot)$.*

Proposition 7. *If A is a monotone c-atom and $a \in A_d$, then for any M, N such that $M \subseteq N$:*

1. *$M \models A$ iff $M \models a \wedge A^+(a)$ or $M \models A^-(a)$;*
2. *$M \models_N A$ iff $M \models_N a \wedge A^+(a)$ or $M \models_N A^-(a)$; and*
3. *$M \models A^N$ iff $M \models a \wedge A^+(a)^N$ or $M \models A^-(a)^N$.*

We stress that the Shannon encoding $\mathcal{S}(A,a)$ is not even satisfaction preserving if applied to other than monotone c-atoms. This is illustrated below.

Example 5. Consider the antimonotone c-atom $A = (\{a\},\{\emptyset\})$. We have that $A^+(a) = (\emptyset,\emptyset)$, $A^-(a) = (\emptyset,\{\emptyset\})$, and $\mathcal{S}(A,a) = \mathcal{D}(\mathcal{C}(a, A^+(a)), A^-(a)) \equiv (\emptyset,\{\emptyset\})$. Let $M = \{a\}$. It is easy to see that $M \models \mathcal{S}(A,a)$. But, on the other hand, we have $M \not\models A$. □

However, for monotone c-atoms, strong equivalence is additionally preserved under Shannon encodings. Since $\mathcal{S}(A,a)$ is a combination of disjunctive and conjunctive encodings, our preceding results on rewriting rules and programs apply. Given a rule r of the form (1), a monotone c-atom A_k in the body $bd(r)$, and an atom $a \in (A_k)_d$, the *Shannon rewrite* of r consists of two rules $r[A_k/(\{a\},\{\{a\}\}), A^+(a)]$ and $r[A_k/A^-(a)]$. Such replacements are highly beneficial if either $A^+(a)$ or $A^-(a)$ becomes trivial in one sense (cf. Example 4). If not, then repeated Shannon rewritings can lead to an exponential expansion.

Theorem 5. *Let r be a rule of the form (1), A_k a monotone c-atom in the body $bd(r)$, and $a \in (A_k)_d$ an atom. Then $\{r\} \equiv_s \{r[A_k/a, A^+(a)], r[A_k/A^-(a)]\}$ where $A^+(a)$ and $A^-(a)$ are the respective positive and negative encodings of A.*

Corollary 3. *For an n-step Shannon rewrite P' of a program P, $P \equiv_s P'$.*

It is also possible to rewrite a program P by mixing conjunctive, disjunctive, and Shannon rewriting (Shannon rewriting can be only done for monotone c-atoms). Corollaries 1, 2, and 3 guarantee, on their behalf, that the resulting program will be strongly equivalent with the original one.

5 Interconnections to Some Existing Encodings

In this section, we work out the interconnections of some existing translations of c-atoms in the literature to conjunctive and disjunctive encodings. In this way, we can establish that these transformations preserve strong equivalence by appealing to the results of Section 4.

Liu and Truszczyński [11] propose a way of representing any convex c-atom A as a conjunction of two c-atoms A^+ and A^- that are the *upward* and *downward* closures of A, respectively. These closures are defined by

1. $A^+_d = A^-_d = A_d$,
2. $A^+_c = \{T \subseteq A_d \mid S \subseteq T \text{ for some } S \in A_c\}$, and
3. $A^-_c = \{T \subseteq A_d \mid T \subseteq S \text{ for some } S \in A_c\}$.

It is obvious that A^+ is monotone and A^- is antimonotone. In addition to this, the two conditions from Definition 12 can be verified so that $A = \mathcal{C}(A^+, A^-)$ holds in general. This justifies the statement of Proposition 8 given below. Thus it follows by Theorem 3 and Corollary 1 that when a convex c-atom A appearing in a program is replaced by its upward and downward closures A^+ and A^-, the resulting program is strongly equivalent to the original one.

Proposition 8. *The encoding of convex c-atoms in terms of their upward and downward closures [11] is a conjunctive encoding.*

As regards arbitrary c-atoms, a number of representations have been proposed such as *sampler sets* [16], the *translation* of aggregates into propositional formulas [19,20], *abstract representations* [21], and *local power sets* [22]. Given a c-atom A, these approaches are essentially based on a disjunctive encoding $A = \mathcal{D}(A_1, \ldots, A_m)$ where each A_i is defined by

1. $(A_i)_d = A_d$ and
2. $(A_i)_c = \{T \subseteq A_d \mid L_i \subseteq T \subseteq G_i\}$

where L_i and G_i are *least* and *greatest* admissible solutions from A_c such that (i) $L_i \subseteq G_i$, (ii) each T between L_i and G_i also belongs to A_c and (iii) the range induced by L_i and G_i is maximal in this sense. Intuitively, the sets L_i and $A_d \setminus G_i$ consist of atoms that have to be true and false, respectively, in order to satisfy A_i as well as A. It is obvious that each A_i is convex. For this reason we call an encoding of this kind as the *convex* encoding of A. Proposition 9 below is a consequence of verifying that A and $A_1, \ldots, A_m$ meet the requirements of Definition 13. So, referring to Theorem 4, given a rule r with an arbitrary c-atom in its body and a convex encoding $A = \mathcal{D}(A, \ldots, A_m)$ for A, the replacement of r by $r[A/A_1], \ldots, r[A/A_m]$ leads to a strongly equivalent program.

Proposition 9. *The convex encoding based on the representations of c-atoms in [16,19,20,21,22] is a disjunctive encoding.*

6 Related Work

The characterizations of strong equivalence in [10,12,11,13] provide the basis to determine the strong equivalence of normal programs under the answer set semantics [1]. For constraint programs under the answer set semantics, strong equivalence can be determined by translating the constraint programs to normal programs, then applying the previous characterizations. Given a constraint program, note that the resulting normal program may be exponential in the size of the original program [20,8,23]. The characterization of strong equivalence developed in this paper makes it possible to determine whether strong equivalence holds without such a potentially exponential translation.

Example 6. Let P be a program consisting of the rule: $b \leftarrow A$, where A is the aggregate $\mathsf{COUNT}(\{X \mid p(X)\}) \leq k$ and $p(X)$ is defined over the domain $D = \{p(1), ..., p(n)\}$.

Let N be a set of atoms such that $|N \cap D| \leq k$ and $b \in N$. Let M be any subset of N with $b \notin M$. It can be determined, without enumerating the admissible solutions of A, that $M \not\models P^N$, since $M \models_N A$ and $b \notin M$. So, (M, N) does not satisfy the third condition in the definition of SE-models. Then $(M, N) \notin \mathrm{SE}(P)$. So, for any program Q that has (M, N) as its SE-model, we can conclude that $P \not\equiv_s Q$.

On the other hand, to determine the strong equivalence of P and Q using the characterizations of strong equivalence for normal programs, one has to translate P to a normal program whose size is possibly exponential in the size of P (a straightforward translation consists of $\mathcal{O}\binom{n}{k}$ rules). □

The concept of program reduct (Definition 5) developed in this paper leads to a simple definition of answer sets for (disjunctive) constraint programs, where a rule head could be a disjunction of c-atoms. Given a (disjunctive) constraint program P and a set of atoms M, M is an answer set of P if and only if M is a minimal model of the program reduct P^M. It can be verified that the program reduct and the answer sets coincide with the generalized Gelfond-Lifschitz transformation and answer sets defined in [21]. Note, however, the simplicity of our definition.

Program revisions under strong and uniform equivalence have been studied in [24] for disjunctive logic programs. In that research, the concept of revision is motivated to remove redundant rules in a program. In contrast, we study a wider class of programs with constraint atoms and the question how such constraint atoms can be replaced by others, independently of the embedding program.

7 Conclusion and Directions of Future Work

We propose the concept of reduct for logic programs with arbitrary abstract constraint atoms (c-atoms), based on which we give a characterization of strong equivalence and uniform equivalence of logic programs with c-atoms. This extends previous work on logic programs with monotone c-atoms [11]. We study

strong equivalence of a particular class of program revisions: constraint replacements. We provide criteria under which a c-atom can be replaced by a conjunction of c-atoms, a disjunction c-atoms, or a combination of them while preserving strong equivalence. These results provide the basis to check if a constraint can be replaced with other constraints in any program context.

For the future work, it would be interesting to extend our results to programs with concrete constraints such as aggregate programs [7,25]. Another promising direction is that to investigate the strong equivalence of logic programs embedded within other reasoning mechanisms, such as constraint programming (CP) [26,27,28], description logics [29], and SAT modulo theories [30]. These embeddings make it possible to exploit the strengths of other reasoning mechanisms in ASP for solving complex real world problems. In the embedding approach, it is a common practice to replace a part of a program with components of other reasoning mechanisms to model and solve different kinds of problems. The study of strong equivalence in this context may provide insights and approaches to program optimization, system implementation, and efficient answer set computation.

References

1. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. ICLP, pp. 1070–1080 (1988)
2. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Math. and Artificial Intelligence 25(3-4), 241–273 (1999)
3. Marek, V., Niemelä, I., Truszczyński, M.: Logic programs with monotone abstract constraint atoms. TPLP 8(2), 167–199 (2008)
4. Marek, V.W., Remmel, J.B.: Set constraints in logic programming. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 167–179. Springer, Heidelberg (2003)
5. Son, T.C., Pontelli, E., Tu, P.H.: Answer sets for logic programs with arbitrary abstract constraint atoms. JAIR 29, 353–389 (2007)
6. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence 138(1-2), 181–234 (2002)
7. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
8. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. TPLP 7, 355–375 (2006)
9. van Hoeve, W.J., Katriel, I.: Global constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
10. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic 2(4), 526–541 (2001)
11. Liu, L., Truszczyński, M.: Properties and applications of programs with monotone and convex constraints. JAIR 7, 299–334 (2006)
12. Lin, F.: Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: Proc. KR 2002, pp. 170–176 (2002)
13. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. Theory and Practice of Logic Programming 3, 609–622 (2003)

14. Gebser, M., Kaminski, R., Ostrowski, M., Schaub, T., Thiele, S.: On the input language of ASP grounder *gringo*. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 502–508. Springer, Heidelberg (2009)
15. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. TPLP 7(3), 301–353 (2007)
16. Janhunen, T.: Sampler programs: The stable model semantics of abstract constraint programs revisited. In: Proc. ICLP, pp. 94–103 (2010)
17. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Transactions on Computational Logic 8(3) (2007)
18. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity aspects of disjunctive stable models. J. Artif. Intell. Res. (JAIR) 35, 813–857 (2009)
19. Pelov, N.: Semantics of Logic Programs with Aggregates. PhD thesis, Ketholieke Universiteit Leuven (2004)
20. Pelov, N., Denecker, M., Bruynooghe, M.: Translation of aggregate programs to normal logic programs. In: Proc. ASP 2003, pp. 29–42 (2003)
21. Shen, Y., You, J., Yuan, L.: Characterizations of stable model semantics for logic programs with arbitrary constraint atoms. TPLP 9(4), 529–564 (2009)
22. You, J., Liu, G.: Loop formulas for logic programs with arbitrary constraint atoms. In: Proc. AAAI 2008, pp. 584–589 (2008)
23. You, J., Yuan, L.Y., Liu, G., Shen, Y.: Logic programs with abstract constraints: Representation, disjunction and complexities. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 228–240. Springer, Heidelberg (2007)
24. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Simplifying logic programs under uniform and strong equivalence. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 87–99. Springer, Heidelberg (2003)
25. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 119–131. Springer, Heidelberg (2005)
26. Baselice, S., Bonatti, P.A., Gelfond, M.: Towards an integration of answer set and constraint solving. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 52–66. Springer, Heidelberg (2005)
27. Gebser, M., Ostrowski, M., Schaub, T.: Constraint answer set solving. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 235–249. Springer, Heidelberg (2009)
28. Mellarkod, V.S., Gelfond, M., Zhang, Y.: Integrating answer set programming and constraint logic programming. Annals of Math. and AI 53(1-4), 251–287 (2008)
29. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artificial. Intelligence. 172(12-13), 1495–1539 (2008)
30. Niemelä, I.: Integrating answer set programming and satisfiability modulo theories. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, p. 3. Springer, Heidelberg (2009)