Journal of Artificial Intelligence Research 31 (2008) 205–216 Submitted 10/07; published 01/08

Sound and Complete Inference Rules for SE-Consequence

Ka-Shu Wong KSWONG@CSE.UNSW.EDU.AU
University of New South Wales and National ICT Australia
Sydney, NSW 2052, Australia

Abstract

The notion of strong equivalence on logic programs with answer set semantics gives rise to a consequence relation on logic program rules, called *SE-consequence*. We present a sound and complete set of inference rules for SE-consequence on disjunctive logic programs.

1. Introduction

In recent years there has been much research on various notions of equivalence between two logic programs. In particular, the notion of *strong equivalence* of logic programs with answer set semantics (Lifschitz, Pearce, & Valverde, 2001; Turner, 2001, 2003; Cabalar, 2002; Lin, 2002) has received much attention. We say that two logic programs P and Q are strongly equivalent iff for any set of rules R, $P \cup R$ and $Q \cup R$ have the same answer sets.

Recent work in this area (Eiter, Fink, Tompits, & Woltran, 2004; Turner, 2003; Osorio, Navarro, & Arrazola, 2001) has focused on the simplification of logic programs under strong equivalence. This has resulted in a number of logic program transformation rules which preserve strong equivalence. In these transformations, logic program rules are identified which can be removed while maintaining strong equivalence with the original program.

In this paper we look at a different but related aspect of strong equivalence. The notion of strong equivalence on logic programs gives rise to a consequence relation $\models_{\mathrm{s}}$ on logic program rules, called *SE-consequence* (Eiter et al., 2004), which can be defined by saying that a rule r is a consequence of a logic program P iff P and $P \cup r$ are strongly equivalent[1]. This consequence relation is useful in testing for strong equivalence, as well as in identifying redundant rules for logic program simplification.

In this paper, we present a set of inference rules on logic program rules and show that they are sound and complete for SE-consequence. Our set of inference rules consists of adaptations of several well-known logic program simplification rules, together with a new rule which we call S-HYP. The main contribution of this paper is the new inference rule S-HYP and a completeness result. The completeness proof makes use of the construction used in the reduction of strong equivalence testing to classical logic by Lin (2002), and applies to it a restricted form of resolution called *lock resolution* by Boyer (1971).

1. Eiter et al. (2004) uses a different definition of SE-consequence based on Turner's SE-models (Turner, 2001, 2003). The equivalence of the two definitions is proved in Section 3.

2. Definitions

We deal with propositional disjunctive logic programs with negation-as-failure, where each rule is of the form

$$a_1; a_2; \cdots; a_k \leftarrow b_1, b_2, \cdots, b_m, \mathit{not}\ c_1, \mathit{not}\ c_2, \cdots, \mathit{not}\ c_n.$$

where $a_1, \cdots, a_k$, $b_1, \cdots, b_m$ and $c_1, \cdots, c_n$ are from a set $\mathcal{A}$ of atoms, We assume that the set of atoms $\mathcal{A}$ is fixed. Given a rule r in this form, we denote $H(r) = \{a_1, \cdots, a_k\}$ (*head* of r), $B^+(r) = \{b_1, \cdots, b_m\}$ (*positive part* of r), and $B^-(r) = \{c_1, \cdots, c_n\}$ (*negative part* of r).

We ignore the order of atoms within a rule; therefore, a rule can be considered as a triple of atom sets. As an abbreviation, when we include atom sets in a rule, it means that the atoms in the set are in the corresponding part of the rule. In particular, if $X = \{x_1, \cdots, x_k\}$, then $\mathit{not}\ X$ in the body of a rule is an abbreviation for $\mathit{not}\ x_1, \cdots, \mathit{not}\ x_k$. Applied to the rule r from above, if $A = H(r)$, $B = B^+(r)$ and $C = B^-(r)$, then the rule can be abbreviated as

$$A \leftarrow B, \mathit{not}\ C.$$

For a set X of atoms and a logic program P, we use the notation $X \models P$ to mean that X is a model of P in the classical sense: For each $r \in P$, if $B^+(r) \subseteq X$ and $B^-(r) \cap X = \emptyset$, then $H(r) \cap X$ is non-empty. We say that X is a *minimal model* of P if X is minimal by set inclusion among all the models of P, i.e. $X \models P$ and there is no X' such that $X' \subset X$ and $X' \models P$.

The *Gelfond-Lifschitz reduct* (1988) P^X of a program P with respect to a set of atoms X is defined by $P^X = \{H(r) \leftarrow B^+(r) \mid r \in P \text{ and } X \cap B^-(r) = \emptyset\}$. We say that X is an *answer set* of P if X is a minimal model of P^X.

3. Strong Equivalence

The notion of *strong equivalence* (Lifschitz et al., 2001) describes the property that two programs remain equivalent regardless of what additional rules are added, and is defined as follows:

Definition 1. *Logic programs P and Q are strongly equivalent, iff for all sets R of rules, the programs $P \cup R$ and $Q \cup R$ have the same answer sets.*

Lifschitz et al. (2001) showed that strong equivalence can be reduced to equivalence in the logic of here-and-there. Based on this result, Turner (2003) gave the following definition of *SE-models*, which characterises strong equivalence in the sense that two programs are strongly equivalent iff they have the same SE-models:

Definition 2. *Let P be a logic program, and let $X, Y \subseteq \mathcal{A}$ be sets of atoms. We say the pair (X, Y) is a SE-model of P, written $(X, Y) \models P$, if $X \subseteq Y$, $Y \models P$ and $X \models P^Y$. For a set M of SE-models, we write $M \models P$ to mean $(X, Y) \models P$ for all $(X, Y) \in M$. Let $\mathrm{M_s}(P)$ denote the set of all SE-models of P.*

SE-models have the property that a pair (X, Y) is a SE-model of P iff it is a SE-model of every rule $r \in P$. This implies $\mathrm{M_s}(P \cup Q) = \mathrm{M_s}(P) \cap \mathrm{M_s}(Q)$.

The notion of strong equivalence gives rise to a consequence relation on logic program rules, called *SE-consequence* and denoted by $\models_{\mathrm{s}}$. SE-consequence is defined by:

Definition 3. *Let P, Q be logic programs, and r be a logic program rule. We say $P \models_{\mathrm{s}} r$ iff $\mathrm{M_s}(P) \models r$, i.e. every $(X, Y) \in \mathrm{M_s}(P)$ is a SE-model of r. Furthermore, we write $P \models_{\mathrm{s}} Q$ iff $P \models_{\mathrm{s}} r$ for every $r \in Q$.*

There is an equivalent definition of SE-consequence which does not make use of SE-models:

Proposition 1. *Let P be a logic program and r be a logic program rule. Then $P \models_{\mathrm{s}} r$ iff P and $P \cup \{r\}$ are strongly equivalent.*

Proof. P and $P \cup \{r\}$ are strongly equivalent iff $\mathrm{M_s}(P) = \mathrm{M_s}(P \cup \{r\})$ $(= \mathrm{M_s}(P) \cap \mathrm{M_s}(r))$. This holds iff $\mathrm{M_s}(P) \subseteq \mathrm{M_s}(r)$, i.e. every $(X, Y) \in \mathrm{M_s}(P)$ is a SE-model of r. □

The relation $\models_{\mathrm{s}}$ has the properties of a consequence relation:

Proposition 2. *Let P, Q be logic programs, and r be a logic program rule.*

- *If $r \in P$, then $P \models_{\mathrm{s}} r$*
- *If $P \subseteq Q$ and $P \models_{\mathrm{s}} r$ then $Q \models_{\mathrm{s}} r$*
- *If $P \models_{\mathrm{s}} r$ and $Q \models_{\mathrm{s}} P$ then $Q \models_{\mathrm{s}} r$*

The proofs follow easily from the fact that $P \models_{\mathrm{s}} Q$ iff $\mathrm{M_s}(P) \subseteq \mathrm{M_s}(Q)$.

There exist binary resolution-style calculi for the logic of here-and-there (also known as Gödel's 3-valued logic). However, Example 1 suggests that this cannot be applied to SE-consequence, seemingly because it takes one outside the logic fragment corresponding to disjunctive logic programs. This is supported by the fact that the construction in Section 6.1 may produce clauses which do not correspond to logic program rules. This is currently being investigated.

4. Inference Rules for Strong Equivalence

The consequence relation $\vdash_{\mathrm{s}}$ is defined by the following rules of inference:

$$\textbf{(TAUT)}\ \frac{}{x \leftarrow x.} \qquad \textbf{(CONTRA)}\ \frac{}{\leftarrow x, not\ x.}$$

$$\textbf{(NONMIN)}\ \frac{A \leftarrow B, not\ C.}{A; X \leftarrow B, Y, not\ C, not\ Z.}$$

$$\textbf{(WGPPE)}\ \frac{A_1 \leftarrow B_1, x, not\ C_1. \qquad A_2; x \leftarrow B_2, not\ C_2.}{A_1; A_2 \leftarrow B_1, B_2, not\ C_1, not\ C_2.}$$

$$\textbf{(S-HYP)}\ \frac{\begin{array}{c} A_1 \leftarrow B_1, not\ x_1, not\ C_1. \\ \vdots \\ A_n \leftarrow B_n, not\ x_n, not\ C_n. \\ A \leftarrow x_1, \cdots, x_n, not\ C. \end{array}}{A_1; \cdots; A_n \leftarrow B_1, \cdots, B_n, not\ C_1, \cdots, not\ C_n, not\ A, not\ C.}$$

Many of these rules are well-known: *tautological rules* (TAUT), *contradiction* (CONTRA), *non-minimal rules* (NONMIN), and *weak partial evaluation* (WGPPE) (also called *partial deduction*, Sakama & Seki, 1997). These have been shown to be strong equivalence preserving (Brass & Dix, 1999; Osorio et al., 2001; Eiter et al., 2004). The new rule S-HYP can be thought of as a form of hyper-resolution. To our knowledge it has not been considered before in the literature.

Instead of S-HYP, one might expect a more general rule S-HYP+ which allows additional positive atoms B in the final rule:

$$\textbf{(S-HYP+)}\ \frac{\begin{array}{c} A_1 \leftarrow B_1, \mathit{not}\ x_1, \mathit{not}\ C_1. \\ \vdots \\ A_n \leftarrow B_n, \mathit{not}\ x_n, \mathit{not}\ C_n. \\ A \leftarrow x_1, \cdots, x_n, B, \mathit{not}\ C. \end{array}}{A_1; \cdots ; A_n \leftarrow B, B_1, \cdots, B_n, \mathit{not}\ C_1, \cdots, \mathit{not}\ C_n, \mathit{not}\ A, \mathit{not}\ C.}$$

However, it can be shown that replacing S-HYP with S-HYP+ does not change the consequence relation:

Proposition 3. *Let $\vdash$ be a consequence relation satisfying CONTRA. Then S-HYP and S-HYP+ are interchangeable.*

Proof. Since S-HYP+ is a more general form of S-HYP, it suffices to show that S-HYP+ can be simulated using S-HYP and CONTRA. Suppose we have

$$\begin{array}{c} A_1 \leftarrow B_1, \mathit{not}\ x_1, \mathit{not}\ C_1. \\ \vdots \\ A_n \leftarrow B_n, \mathit{not}\ x_n, \mathit{not}\ C_n. \\ A \leftarrow x_1, \cdots, x_n, b_1, \cdots, b_k, \mathit{not}\ C. \end{array}$$

For each b_i, CONTRA gives us $\leftarrow b_i, \mathit{not}\ b_i$. By using S-HYP on these rules, plus the above, we get

$$A_1; \cdots ; A_n \leftarrow B_1, \cdots, B_n, b_1, \cdots, b_k, \mathit{not}\ C_1, \cdots, \mathit{not}\ C_n, \mathit{not}\ A, \mathit{not}\ C.$$

This is the result of applying S-HYP+ to our initial set of rules. □

Example 1. *We now consider the possibility of using only binary inference rules on the logic program P:*

$$\begin{array}{rl} r_1\text{:} & a \leftarrow \mathit{not}\ x. \\ r_2\text{:} & a \leftarrow \mathit{not}\ y. \\ r_3\text{:} & \leftarrow x, y. \end{array}$$

The following rule is an SE-consequence of P, and can be derived using S-HYP on r_1, r_2 and r_3:

$$s\text{:} \quad a \leftarrow .$$

Now suppose we restrict ourselves to binary inference rules by replacing S-HYP with the binary variant of S-HYP+. Applying S-HYP+ to r_1 and r_3 gives:

$$r_4\text{:} \quad a \leftarrow y.$$

and then using S-HYP+ on r_2 and r_4 gives:

$$t{:}\quad a \leftarrow not\ a.$$

We observe that t is weaker than s. This suggests that it may not be possible to derive s using only binary inference rules. However, it is still an open question as to whether n-ary rules are indeed required.

Observe that the rule CONTRA can be replaced by *s-implication* (S-IMP) (Wang & Zhou, 2005):

Proposition 4. *Let $\vdash$ be a consequence relation satisfying TAUT and WGPPE. Then CONTRA can be replaced by the following inference rule:*

$$\textbf{(S-IMP)}\ \frac{A; X \leftarrow B, not\ C.}{A \leftarrow B, not\ C, not\ X.}$$

Proof. (S-IMP $\Rightarrow$ CONTRA) The rule $\leftarrow x, not\ x.$ can be formed by applying TAUT followed by S-IMP.
(CONTRA $\Rightarrow$ S-IMP) Suppose we have the rule

$$A; x \leftarrow B, not\ C.$$

By applying CONTRA to get $\leftarrow x, not\ x.$ followed by WGPPE on these two rules, we get

$$A \leftarrow B, not\ C, not\ x.$$

By repeating these steps we can derive new rules by moving any set of atoms from the head to the negative part of the rule. □

We note that S-IMP is a special case of the solution of the 1-1-0 problem of Lin and Chen (2007). In addition, WGPPE as well as the binary variant of S-HYP+ are contained in the solution to Lin and Chen's 2-1-0 problem.

It can be shown that the inference rules presented above are sound and complete for SE-consequence:

Theorem 1. $P \models_s r$ *iff* $P \vdash_s r$.

Proof (Soundness). Here we prove soundness only for the inference rule S-HYP, as the soundness of the other rules are already known.

The proof proceeds by contradiction. Assume the rule is not sound. Then there is a program P

$$\begin{array}{c} A_1 \leftarrow B_1, not\ x_1, not\ C_1. \\ \vdots \\ A_n \leftarrow B_n, not\ x_n, not\ C_n. \\ A \leftarrow x_1, \cdots, x_n, not\ C. \end{array}$$

and a rule r

$$A_1; \cdots; A_n \leftarrow B_1, \cdots, B_n, not\ C_1, \cdots, not\ C_n, not\ A, not\ C.$$

for which $P \vdash_{\mathrm{s}} r$ but $P \not\models_{\mathrm{s}} r$. This means $\mathrm{M}_{\mathrm{s}}(P) \not\subseteq \mathrm{M}_{\mathrm{s}}(r)$, so P has a SE-model (X, Y) which is not a SE-model of r.

(X, Y) not being a SE-model of r means either $Y \not\models r$ or $X \not\models r^Y$. We can exclude the first case since it is clear that r is a classical consequence of P. Therefore assume $X \not\models r^Y$. For each $1 \leq i \leq n$ we have $A_i \cap X = \emptyset$, $B_i \subseteq X$, and $C_i \cap Y = \emptyset$. Furthermore we have $A \cap Y = C \cap Y = \emptyset$.

But (X, Y) is a SE-model of P, hence $X \models P^Y$. For each $1 \leq i \leq n$, we have the following rule in P:

$$A_i \leftarrow B_i, \mathit{not}\ x_i, \mathit{not}\ C_i.$$

Since $A_i \cap X = \emptyset$, the body must not hold, or the rule must be eliminated in the reduct. But we know $B_i \subseteq X$ and $C_i \cap Y = \emptyset$. Therefore we must have $x_i \in Y$ so that the rule is eliminated in the reduct.

We have $A \cap Y = C \cap Y = \emptyset$ and $x_1, \cdots, x_n \in Y$, so Y is not a classical model of

$$A \leftarrow x_1, \cdots, x_n, \mathit{not}\ C.$$

and hence $Y \not\models P$, which contradicts (X, Y) being a SE-model of P. □

5. Some Background for the Completeness Proof

In this section we introduce two results which will be used in the completeness proof.

5.1 Lin's Construction

Lin (2002) presented a method of reducing strong equivalence to equivalence in classical logic. Given a logic program, this construction produces a set of clauses such that two logic programs are strongly equivalent iff the two sets of clauses are equivalent in classical logic.

Let $\{x_1, \cdots, x_n\}$ be the set of atoms. In the construction, each atom x_i is represented by two propositional letters x_i and x'_i. The logic program P consists of a set of rules of the following form:

$$a_1; a_2; \cdots; a_k \leftarrow b_1, b_2, \cdots, b_m, \mathit{not}\ c_1, \mathit{not}\ c_2, \cdots, \mathit{not}\ c_n.$$

For each such rule r, we construct two clauses $\gamma(r)$ and $\gamma'(r)$:

$$\gamma(r) := a_1 \vee \cdots \vee a_k \vee \neg b_1 \vee \cdots \vee \neg b_m \vee c'_1 \vee \cdots \vee c'_n$$

$$\gamma'(r) := a'_1 \vee \cdots \vee a'_k \vee \neg b'_1 \vee \cdots \vee \neg b'_m \vee c'_1 \vee \cdots \vee c'_n$$

Let $\Gamma(P) := \{\gamma(r) \mid r \in P\}$ and $\Gamma'(P) := \{\gamma'(r) \mid r \in P\}$. Furthermore for each atom x_i we add the clause $\neg x_i \vee x'_i$. Let Δ denote the set of all such clauses. Lin's result showed that P and Q are strongly equivalent iff

$$\bigwedge(\Gamma(P) \cup \Gamma'(P) \cup \Delta) \equiv \bigwedge(\Gamma(Q) \cup \Gamma'(Q) \cup \Delta)$$

An immediate corollary of this result is:

Proposition 5. $P \models_{\mathrm{s}} r$ *iff*

$$\bigwedge(\Gamma(P) \cup \Gamma'(P) \cup \Delta) \models \gamma(r) \wedge \gamma'(r)$$

Observe that given a clause α, we can find a corresponding rule r such that $\gamma(r) = \alpha$ if and only if there are no literals of the form $\neg x'$ in α.

5.2 Boyer's Lock Resolution

Resolution is the following inference rule: given two clauses $x \vee C_1$ and $\neg x \vee C_2$, derive the clause $C_1 \vee C_2$. We say that x and $\neg x$ are the literals *resolved on*, and the clause $C_1 \vee C_2$ is the *resolvent*. It is well-known that resolution is refutation-complete, that is, if a set of clauses is unsatisfiable, then the empty clause can be derived using resolution.

Definition 4. *A deduction of the clause C from a set S of clauses is a sequence of clauses $C_1, \cdots, C_n$ such that $C_n = C$ and each C_i is either a clause in S or a resolvent of clauses preceding C_i. If such a deduction exists, we say that C can be derived from S.*

Lock resolution is a restricted form of resolution introduced by Boyer (1971). A numeric label is given to each literal in each clause. Resolution is permitted only on literals with the lowest valued label in their clause. Note that in a clause there can be more than one literal with the same label: if there are many literals with the lowest valued label, then resolution on any of them is allowed. Literals in the resolvent inherit their labels from the parent clauses. If a literal in the resolvent has two possible labels, the lower value is used. A deduction which follows these restrictions is called a *lock deduction*.

Example 2. *Consider the following clauses:*

$$C_1\text{: } a_{(1)} \vee b_{(2)} \qquad C_2\text{: } \neg a_{(2)} \vee b_{(3)} \qquad C_3\text{: } \neg b_{(1)} \vee c_{(2)}$$

We can resolve C_1 and C_2 on $a_{(1)}$ and $\neg a_{(2)}$ to form the following clause:

$$b_{(2)}$$

Note that b is labelled 2 in C_1 and 3 in C_2, so the lower value is used. However we cannot resolve C_2 and C_3 on $b_{(3)}$ and $\neg b_{(1)}$ since 3 is not the minimum label in C_2.

Boyer showed that lock resolution is refutation-complete.

6. The Completeness Proof

Proof of Theorem 1 (Completeness). To prove completeness, we need to show that $P \models_s r$ implies $P \vdash_s r$. We do this by showing the existence of a lock deduction of $\gamma(r')$ where r' is a subset of r (in the sense that $H(r') \subseteq H(r)$, $B^+(r') \subseteq B^+(r)$, and $B^-(r') \subseteq B^-(r)$) from which we can construct a deduction of r from P using the inference rules.

6.1 Lock Deduction of $\gamma(r')$ from $\Gamma(P) \cup \Gamma'(P) \cup \Delta$

From Proposition 5 we know that $\gamma(r)$ is a logical consequence of $\bigwedge(\Gamma(P) \cup \Gamma'(P) \cup \Delta)$. Ideally, we want a lock deduction of $\gamma(r)$ from $\Gamma(P) \cup \Gamma'(P) \cup \Delta$. However, this may not be possible as resolution is only refutation-complete. But in fact we can show that a lock deduction of $\gamma(r')$ does exist for some r', provided that r does not contain the same atom in its head and positive body. We do this by first obtaining a lock deduction D' of the empty clause from $\Gamma(P) \cup \Gamma'(P) \cup \Delta$ and the negation of $\gamma(r)$. We then modify this to form the deduction D of $\gamma(r')$ from $\Gamma(P) \cup \Gamma'(P) \cup \Delta$, in such a way that the restrictions of lock deduction are preserved.

We label the literals in $\Gamma(P) \cup \Gamma'(P) \cup \Delta$ with either 1 or 0: if the literal is of the form $\neg x'$, we give it the label 0, otherwise we give it the label 1. For example, $\gamma(a \leftarrow b, \mathit{not}\ c)$ becomes

$$a_{(1)} \vee \neg b_{(1)} \vee c'_{(1)}$$

and $\gamma'(a \leftarrow b, \mathit{not}\ c)$ becomes

$$a'_{(1)} \vee \neg b'_{(0)} \vee c'_{(1)}$$

Let r be the rule

$$a_1; a_2; \cdots; a_k \leftarrow b_1, b_2, \cdots, b_m, \mathit{not}\ c_1, \mathit{not}\ c_2, \cdots, \mathit{not}\ c_n.$$

Then $\gamma(r)$ is

$$a_1 \vee \cdots \vee a_k \vee \neg b_1 \vee \cdots \vee \neg b_m \vee c'_1 \vee \cdots \vee c'_n$$

Negating $\gamma(r)$ gives us the following set of clauses, which we will call $N(r)$:

$$\neg a_1, \cdots, \neg a_k, b_1, \cdots, b_m, \neg c'_1, \cdots, \neg c'_n$$

We label these clauses in the same way as above. Note that $\gamma(r)$ contain only literals with label 1, while $N(r)$ is a set of single-literal clauses, where some of them may also have label 0. Since $\gamma(r)$ is a consequence of $\bigwedge(\Gamma(P) \cup \Gamma'(P) \cup \Delta)$, adding the negation of $\gamma(r)$ makes it unsatisfiable. Therefore there is a lock deduction D' of the empty clause from $\Gamma(P) \cup \Gamma'(P) \cup \Delta$ plus $N(r)$.

In the next step, we construct a new lock deduction D which does not contain any clauses from $N(r)$. Let $C'_1, \cdots, C'_n$ be the clauses in D'. We construct inductively the clauses $C_1, \cdots, C_n$:

- C'_i is not a resolvent. In this case we set $C_i := C'_i$. Note that if C'_i is from $N(r)$, it will be removed in the next step of the construction.
- C'_i is a resolvent of C'_j and C'_k on the literals x and $\neg x$, where neither C'_j or C'_k is from $N(r)$. We set C_i to be the resolvent of C_j and C_k on x and $\neg x$.
- C'_i is a resolvent of C'_j and C'_k, one of which is from $N(r)$. Without loss of generality, assume C'_j is from $N(r)$. In this case we set $C_i := C_k$.

To complete the construction, we remove every C_i which is from $N(r)$ and is not a resolvent. The remaining clauses form the deduction D. Note that it is not possible for C'_i to be a resolvent of two clauses from $N(r)$. This is because r does not contain the same atom in both its head and positive body, and hence $N(r)$ cannot contain a pair of complementary literals. Note also that the final clause C_n is never removed, since C'_n is the empty clause, which is not in $N(r)$.

Example 3. *Suppose P is the program consisting of the single rule*

$$a \leftarrow b.$$

and r is the rule

$$\leftarrow b, \mathit{not}\ a.$$

Then $\Gamma(P) \cup \Gamma'(P) \cup \Delta$ consists of the clauses

$$a_{(1)} \vee \neg b_{(1)} \qquad a'_{(1)} \vee \neg b'_{(0)} \qquad \neg a_{(1)} \vee a'_{(1)} \qquad \neg b_{(1)} \vee b'_{(1)}$$

$\gamma(r)$ is $\neg b_{(1)} \vee a'_{(1)}$ (after labelling). The negation of $\gamma(r)$, denoted $N(r)$, consists of the clauses

$$b_{(1)} \qquad \neg a'_{(0)}$$

Here is an example of a lock deduction D' for the empty clause from $\Gamma(P) \cup \Gamma'(P) \cup \Delta \cup N(r)$:

(1)	$b_{(1)}$	*from $N(r)$*
(2)	$\neg b_{(1)} \vee b'_{(1)}$	*from Δ*
(3)	$b'_{(1)}$	*resolvent of (1), (2)*
(4)	$a'_{(1)} \vee \neg b'_{(0)}$	*from $\Gamma'(P)$*
(5)	$a'_{(1)}$	*resolvent of (3), (4)*
(6)	$\neg a'_{(0)}$	*from N(r)*
(7)	$\perp$	*resolvent of (5), (6)*

The construction produces the following sequence of clauses:

$(1)*$	$b_{(1)}$	
(2)	$\neg b_{(1)} \vee b'_{(1)}$	*from Δ*
(3)	$\neg b_{(1)} \vee b'_{(1)}$	*from Δ—copy of (2)*
(4)	$a'_{(1)} \vee \neg b'_{(0)}$	*from $\Gamma'(P)$*
(5)	$a'_{(1)} \vee \neg b_{(1)}$	*resolvent of (3), (4)*
$(6)*$	$\neg a'_{(0)}$	
(7)	$a'_{(1)} \vee \neg b_{(1)}$	*resolvent of (3), (4)—copy of (5)*

The clauses marked with $*$ are removed, and the resulting deduction D is formed by the remaining clauses.

By construction, D is a deduction. We need to show that D satisfies the restrictions of lock deduction. In addition, we show that each clause C_i in D consists of C'_i with zero or more literals from $\gamma(r)$ added, i.e. $C_i = C'_i \vee \alpha_i$ where α_i is a disjunction of zero or more literals from $\gamma(r)$. The proof is by induction.

- C'_i is not a resolvent. In this case C_i is the same as C'_i.
- C'_i is a resolvent of clauses C'_j and C'_k on the literals x and $\neg x$, with neither being from $N(r)$. Then C_i is the resolvent of C_j and C_k on the literals x and $\neg x$. By induction, $C_j = C'_j \vee \alpha_j$ and $C_k = C'_k \vee \alpha_k$. If x does not occur in α_j and α_k, then $C_i = C'_i \vee \alpha_j \vee \alpha_k$. Otherwise $C_i = C'_i \vee \alpha_i$ where α_i contains the literals in $\alpha_j \vee \alpha_k$ except possibly for x or $\neg x$. In both cases, C_i consists of C'_i plus zero or more literals from $\gamma(r)$.

  This resolution step satisfies the restriction of lock deduction because the added literals are all labelled 1, which is the highest label value that we use. Thus if x has the lowest label value in C'_j and $\neg x$ in C'_k, the same must hold for x in C_j and $\neg x$ in C_k.
- C'_i is a resolvent of clauses C'_j and C'_k, and C'_j is from $N(r)$. Since each clause in $N(r)$ is the negation of a literal found in $\gamma(r)$ (note they might carry different labels), there must be some literal α in $\gamma(r)$ such that $C'_j = \neg\alpha$ and $C'_k = C'_i \vee \alpha$. The construction of C_i gives us $C_i = C_k$, and by induction $C_k = C'_k \vee \alpha_k$. Therefore $C_i = C'_i \vee \alpha \vee \alpha_k$.

We have shown that D is a lock deduction, and that each clause C_i in D consists of C_i' with zero or more literals from $\gamma(r)$ added. Recall that the final clause in D' is the empty clause. Therefore the final clause in D is a clause which contains some subset of the literals of $\gamma(r)$, and hence D is a lock deduction for some $\gamma(r')$ where r' is a subset of r.

6.2 Existence of a Deduction of r from P

Suppose we have a lock deduction of $\gamma(r)$ from $\Gamma(P) \cup \Gamma'(P) \cup \Delta$ with the labelling described above. We prove by induction that $P \vdash_{\mathrm{s}} r$.

BASE CASE

$\gamma(r)$ is either in $\Gamma(P)$ or $\Gamma'(P)$ or Δ.

- $\gamma(r)$ is in $\Gamma(P)$. Then $r \in P$, therefore $P \vdash_{\mathrm{s}} r$.
- $\gamma(r)$ is in $\Gamma'(P)$. This means $\gamma(r) = \gamma'(s)$ for some $s \in P$. $B^+(s)$ must be empty since there can be no literals of the form $\neg x'$ in $\gamma(r)$. Hence we can write s as

  $$a_1; \cdots ; a_k \leftarrow \mathit{not}\ c_1, \cdots \mathit{not}\ c_n.$$

  and r is

  $$\leftarrow \mathit{not}\ a_1, \cdots , \mathit{not}\ a_k, \mathit{not}\ c_1, \cdots \mathit{not}\ c_n.$$

  Since $\vdash_{\mathrm{s}} \leftarrow a_i, \mathit{not}\ a_i$, we can "move" atoms from the head to the negative part using WGPPE. Therefore $P \vdash_{\mathrm{s}} r$.
- $\gamma(r)$ is in Δ. In this case r is

  $$\leftarrow x, \mathit{not}\ x.$$

  for some atom x, and $\vdash_{\mathrm{s}} r$ by applying CONTRA.

INDUCTION STEP

$\gamma(r)$ is the resolvent of clauses α and β. The literal resolved on is either a and $\neg a$ or a' and $\neg a'$ for some atom a. Assume without loss of generality that the positive literal is in α and the negative literal is in β.

- a in α and $\neg a$ in β is the literal resolved on. Since there is no literal of the form $\neg x'$ in the resolvent, there cannot be any literal of that form in α or β. Therefore we can find logic program rules s and t such that $\alpha = \gamma(s)$ and $\beta = \gamma(t)$. The inference rule WGPPE gives us

  $$s, t \vdash_{\mathrm{s}} r$$

  and $P \vdash_{\mathrm{s}} s, t$ by induction. Therefore $P \vdash_{\mathrm{s}} r$.
- a' in α and $\neg a'$ in β is the literal resolved on. Because $\neg a'$ is in β, there is no logic program rule t such that $\beta = \gamma(t)$. However, we can find a logic program rule s such that $\alpha = \gamma(s)$, because of the lock resolution property: a' is labelled 1 so there can be no literal of the form $\neg x'$, which is labelled 0, in α.

If β is a resolvent, then the literal resolved on must be of the form x' and $\neg x'$, with $\neg a'$ being from the parent clause that contains $\neg x'$. Again, this is because of the lock resolution property: the presence of $\neg a'$ in that clause prevents resolution on any literal that is not labelled 0. Let β_2 be the parent clause containing $\neg x'$, and let α_2 be the other parent clause. Now α_2 only contains literals labelled 1, so we can find a logic program rule s_2 such that $\alpha_2 = \gamma(s_2)$.

By repeating this, we form a chain of resolvents $\alpha_1 (= \alpha), \alpha_2, \cdots, \alpha_n$ and $\beta_1 (= \beta), \beta_2, \cdots, \beta_n$. Each β_i is the resolvent of α_{i+1} and β_{i+1} on the literal a'_i in α_{i+1} and $\neg a'_i$ in β_{i+1}. There is a logic program rule s_i corresponding to each α_i such that $\gamma(s_i) = \alpha_i$.

We extend this chain as far as possible, until β_n is not a resolvent. β_n contains literals of the form $\neg x'$, so it can only come from $\Gamma'(P)$. Therefore there is a $t \in P$ such that $\gamma'(t) = \beta_n$. Observe that t is the rule

$$X \leftarrow a_1, \cdots, a_n, \mathit{not}\, Y.$$

for some sets of atoms X, Y. There cannot be additional positive atoms in the body, because they correspond to literals of the form $\neg x'$ in β_n, and there are no such literals remaining the result of this chain of resolution steps, which is $\gamma(r)$.

The inference rule S-HYP gives us

$$s_1, \cdots, s_n, t \vdash_{\mathrm{s}} r$$

Now $P \vdash_{\mathrm{s}} s_1, \cdots, s_n$ by induction, and $P \vdash_{\mathrm{s}} t$ because $t \in P$. Therefore $P \vdash_{\mathrm{s}} r$.

6.3 The Final Step

We have shown that if $P \models_{\mathrm{s}} r$, then we can find r' which is a "subset" of r such that $P \vdash_{\mathrm{s}} r'$, apart from some special cases. The final step of the proof is given by applying NONMIN, which allows us the deduction $r' \vdash_{\mathrm{s}} r$ when r' is a "subset" of r.

The special case where r contains the same atom in its head and positive body is handled by observing that r can be produced using the rules TAUT followed by NONMIN, which shows $\vdash_{\mathrm{s}} r$.

Therefore $P \models_{\mathrm{s}} r$ implies $P \vdash_{\mathrm{s}} r$. $\square$

7. Conclusion

In this paper we presented a sound and complete set of inference rules for SE-consequence on disjunctive logic programs, consisting of a number of well-known logic program transformation rules, TAUT, CONTRA, NONMIN, and WGPPE, plus a new rule which we call S-HYP. We proved that this set of rules is complete for SE-consequence by using a reduction of logic programs to propositional clauses on which we apply a restricted form of resolution. This result leads to a syntactic definition of the closure operator for logic programs under strong equivalence. Future work involves applying this to construct logic program update operators that respect strong equivalence, as well as finding similar results for other notions of equivalence on logic programs.

Acknowledgments

We thank the anonymous reviewers for their many helpful suggestions which we used in revising the paper. This work was partially supported by a scholarship from National ICT Australia. NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

Boyer, R. (1971). *Locking: A Restriction of Resolution*. Ph.D. thesis, University of Texas, Austin.

Brass, S., & Dix, J. (1999). Semantics of (disjunctive) logic programs based on partial evaluation. *Journal of Logic Programming*, *38(3)*, 167–213.

Cabalar, P. (2002). A three-valued characterization for strong equivalence of logic programs. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-2002)*, pp. 106–111.

Eiter, T., Fink, M., Tompits, H., & Woltran, S. (2004). Simplifying logic programs under uniform and strong equivalence. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 87–99.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pp. 1070–1080.

Lifschitz, V., Pearce, D., & Valverde, A. (2001). Strongly equivalent logic programs. *Computational Logic*, *2*(4), 526–541.

Lin, F. (2002). Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 170–176.

Lin, F., & Chen, Y. (2007). Discovering classes of strongly equivalent logic programs. *Journal of Artificial Intelligence Research*, *28*, 431–451.

Osorio, M., Navarro, J. A., & Arrazola, J. (2001). Equivalence in answer set programming. In *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation*, pp. 57–75.

Sakama, C., & Seki, H. (1997). Partial deduction in disjunctive logic programming. *Journal of Logic Programming*, *32*(3), 229–245.

Turner, H. (2001). Strong equivalence for logic programs and default theories (made easy). In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 81–92.

Turner, H. (2003). Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming*, *3*(4-5), 609–622.

Wang, K., & Zhou, L. (2005). Comparisons and computation of well-founded semantics for disjunctive logic programs. *ACM Transactions on Computational Logic*, *6*(2), 295–327.