

Computing Multiple-Valued Logic Programs *

James J. Lu[†] Jacques Calmet and Joachim Schü²

¹ Department of Computer Science. Bucknell University
Lewisburg, PA 17837. U.S.A.

jameslu@bucknell.edu

² Department of Computer Science.

Institute for Algorithms and Cognitive Systems
University of Karlsruhe, 76128 Karlsruhe. Am Fasanengarten 5
{calmet,schue}@ira.uka.de

Abstract

The logic of signed formula can be used to reason about a wide variety of multiple-valued logics [Häh94b, LMR97]. The formal theoretical foundation of multiple-valued logic programming based on signed formulas is set forth in [Lu96]. The current paper is an investigation into the operational semantics of such signed logic programming. The connection of signed logic programming to constraint logic programming is presented, search space issues are briefly discussed for both general and special cases, and applications to bilattice logic programming and truth-maintenance are analyzed.

keywords: Logic for Artificial Intelligence, Multiple-valued Logic, Signed Formula, Constraint Logic Programming, Truth-Maintenance, Bilattices.

1 Introduction

The logic of signed formulas facilitates the examination of questions regarding multiple-valued logics through classical logic. As such, logic programming based on signed formulas also facilitates the analysis of multiple-valued logic programming systems through classical logic programming. The theoretical foundation and the applications of the logic of signed formulas have been investigated extensively [Häh91, Häh94a, LMR97, MR94]. On the other hand, logic programming based on signed formulas — signed formula logic programming — is only formalized recently [Lu96]. There, the semantical connections between a signed formula logic program and its associated underlying multiple-valued logic program are studied. In addition, the relationships between signed formula logic programming and the class of annotated logic programming [BS89, KS92] are established. It is shown

*An extended abstract of the preliminary version of this paper appears in ISMIS '96.

[†]Work supported in part by the NSF under Grant CCR9225037.

that signed formula logic programming and annotated logic programming together provide a basis for reasoning about “inconsistent” multiple-valued logic programs.

This paper extends the work in [Lu96] by considering first of all, the operational details of signed formula logic programming. It is demonstrated that a signed formula logic program may be formulated as an equivalent constraint logic program. From a practical point of view, this equivalence makes available to signed formula logic programming a wide variety of implementation techniques that have been developed for constraint logic programming. Moreover, the operational behavior of constraint logic programming sheds insights into the search space of signed resolution, which was a procedure proposed in [Lu96] for processing queries with respect to signed formula logic programs. For the class of *regular logics*, a technique for reducing the search space called partial solvability is introduced. We also analyze two independent applications of signed formula logic programming: bilattice logic programming [Fit91] and assumption based truth-maintenance [DeK86]. The application to bilattice logic programming demonstrates how a signed formula logic program may be used to answer questions about an underlying multiple-valued logic program. On the other hand, the application to assumption based truth-maintenance provides a semantical characterization of the popular reasoning system through signed formula logic programming.

The organization of the paper is as follows. Section 2 summarizes the theoretical foundation of signed formula logic programming as examined in detail in [Lu96]. Section 3 explores the semantical connection of signed formula logic programming and constraint logic programming. Analyses and comparisons of the operational semantics of the two formalisms are provided. Section 4 investigates the applications of signed formula logic programming to bilattice logic programming (Section 4.1) and assumption based truth maintenance system (Section 4.2). Ideas described in this paper evolved from a number of recent work [BF92, Häh93, MR94, Fr94, Lu96]. A brief examination of their relations to the current study is given in Section 5.

2 Signed Formula Logic Programming

This section recapitulates fundamental definitions and results of signed formula logic programming as presented in [Lu96]. For completeness, proofs of some of the more important results are repeated.

2.1 Signed Formula

The basic building blocks of signed formulas are a multiple-valued logic Λ and its associated (finite) set of truth values Δ . A *sign* is an expression, which may contain variables, that denotes a non-empty subset of Δ .¹ Suppose S is a sign and \mathcal{F} is a Λ -formula. Then $S : \mathcal{F}$ is a *signed atom*.² More complex formulas — *signed formulas*

¹To simplify the presentation, we blur the distinction between the language from which such an expression is constructed, and the objects in Δ over which the symbols of this language is interpreted.

²Abstractly, formulas in Λ are constructed from atomic formulas and connectives of different arities. Suppose Θ is an n -ary connective, and $\mathcal{F}_1, \dots, \mathcal{F}_n$ are Λ -formulas. Then the the expression

— may be constructed recursively using signed atoms and classical connectives by: $\neg\mathcal{G}$, $\mathcal{G}_1 \mid \mathcal{G}_2$, $\mathcal{G}_1 \& \mathcal{G}_2$, $\mathcal{G}_1 \leftarrow \mathcal{G}_2$, where $\mathcal{G}, \mathcal{G}_1, \mathcal{G}_2$ are signed formulas.³ If $S : \mathcal{F}$ is a signed atom in which \mathcal{F} contains no occurrences of Λ -connectives, then $S : \mathcal{F}$ is said to be Λ -atomic.

2.2 Logic Programming

We are interested in signed clauses — signed formulas of the form

$$S_0 : A \leftarrow S_1 : \mathcal{F}_1 \& \dots \& S_n : \mathcal{F}_n$$

where $S_0 : A$ is a Λ -atomic signed atom, and each $S_1 : \mathcal{F}_1, \dots, S_n : \mathcal{F}_n$ is a signed atom. A finite set of signed clauses is called a *signed formula logic program* (SFLP). In a signed clause, the conjunction appearing on the right hand side of the \leftarrow symbol is called the *body* of the clause, and the single signed atom to the left of the \leftarrow is called the *head* of the clause. Variables that occur in the clause, whether they appear in formulas over Λ or in signs, are assumed to stand for all possible ground instantiations, under the restriction that variables appearing in signs are substituted with subsets of Δ , and variables that appear in atoms are substituted with terms in Λ . A bodiless signed clause is sometimes called a *signed fact*, or a *signed unit clause*. A headless signed clause is a *signed query*.

Interpretations over the logic Λ map ground atoms to Δ , and are extended to Λ -formulas according to the meaning of the connectives that appear in the formulas. Intuitively, a signed formula $S : \mathcal{F}$ may be thought of as representing the query: “Can \mathcal{F} evaluate to some element in S ?” [LMR97].

Definition 1 (Satisfaction) A Λ -interpretation I *satisfies* a variable free signed atom $S : \mathcal{F}$ iff $I(\mathcal{F}) \in S$.⁴ Satisfaction is extended to arbitrary signed formulas in the usual way. A signed clause is satisfied by an interpretation I if each ground instance of the clause is satisfied by I . An SFLP P is satisfied by an interpretation I if each signed clause is satisfied by I ; I is said to be a *model* of P .

We write $\text{Mod}(S : \mathcal{F})$ to denote the collection of all Λ -interpretations that satisfy the signed atom $S : \mathcal{F}$.

Proposition 2 Mod extends to arbitrary signed formulas as follows.

- $\text{Mod}(\mathcal{F}_1 \& \mathcal{F}_2) = \text{Mod}(\mathcal{F}_1) \cap \text{Mod}(\mathcal{F}_2)$.
- $\text{Mod}(\mathcal{F}_1 \mid \mathcal{F}_2) = \text{Mod}(\mathcal{F}_1) \cup \text{Mod}(\mathcal{F}_2)$.
- $\text{Mod}(\neg\mathcal{F}) = \Omega - \text{Mod}(\mathcal{F})$, where Ω is the set of all Λ -interpretations.
- $\text{Mod}(\mathcal{F}_1 \leftarrow \mathcal{F}_2) = \text{Mod}(\mathcal{F}_1) \cup (\Omega - \text{Mod}(\mathcal{F}_2))$

$\Theta(\mathcal{F}_1, \dots, \mathcal{F}_n)$ is also a Λ -formula.

³We use \mid and $\&$ to denote classical or and and respectively. The symbols \vee and \wedge will be used in Section 4.1 to denote connectives in Λ .

⁴This reflects the intuitive reading of $S : \mathcal{F}$ since I is a witness to the question “Can \mathcal{F} evaluate to a value in S ?”

The classical notion of logical consequence applies, and is written with the usual notation \models . The collection of all models of an SFLP P is denoted $\text{Mod}(P)$.

Clearly, there will be SFLPs for which no models exist. Consider for example the SFLP P over the $\Delta = \{0, 0.2, 0.5, 0.8, 1\}$.⁵

$$\begin{aligned} \{1\} : A &\leftarrow \\ \{0\} : A &\leftarrow \end{aligned}$$

P possesses no model since no Λ -interpretation can assign both 1 and 0 to the proposition A . The existence of such an *inconsistent* program does not concern us. It simply indicates that there exist formulas in the underlying multiple valued logic Λ for which certain assignment of truth values is impossible. Indeed, their existence give rise to the interesting possibility of using signed formula logic programs in conjunction with *annotated logic* to reason about inconsistent multiple valued knowledge bases [LR95].

2.3 Semantics

An important property of classical logic programming is that a program P possesses a unique minimal model (with respect to an appropriate ordering). In the case of an SFLP, this property does not hold. For instance, using $\Delta = \{0, 0.2, 0.5, 0.8, 1\}$ again as our truth values, if we have the program P that contains the single unit signed atom $\{0, 1\} : A \leftarrow$, then P has two models:

$$\begin{aligned} I_1(A) &= 1 \\ I_2(A) &= 0 \end{aligned}$$

If we regard Δ as ordered according to the usual less than relation, then a reasonable choice for a minimal model is I_2 since $0 \leq 1$. However, as the truth value set Δ is not in general assumed to be equipped with any ordering, consequently if we treat the elements in Δ as being independent of one another, then I_1 and I_2 are incomparable models.

This leaves us with a rather undesirable situation. An SFLP may be *disjunctive*, and this complicates computational issues since it may be necessary to answer queries with respect to multiple models — a difficult problem well-known in the research on disjunctive logic programming [LMR92]. Fortunately we may obtain a good approximation to the models $\text{Mod}(P)$ via an extension to the notion of interpretation. Intuitively, extended interpretations can be thought of as functions that measure the “indefiniteness” of each proposition in an SFLP.

Definition 3 (Extended Interpretation) An *extended interpretation* I of Λ is a mapping from ground atoms to subsets of Δ . It extends to arbitrary variable-free Λ -formulas as follows: Suppose Θ is an n -ary connective in Λ , and $\mathcal{F}_1, \dots, \mathcal{F}_n$ are variable free Λ -formulas. Then

$$I(\Theta(\mathcal{F}_1, \dots, \mathcal{F}_n)) = \{\Theta(\mu_1, \dots, \mu_n) \mid \mu_i \in I(\mathcal{F}_i), \forall 1 \leq i \leq n\}$$

⁵The truth value set $\{0, 0.2, 0.5, 0.8, 1\}$ has been applied in fuzzy reasoning [WTL93].

Definition 4 (Extended Satisfaction) An extended interpretation I *e-satisfies* (extended satisfies) a variable free signed atom $S : \mathcal{F}$ if $I(\mathcal{F}) \subseteq S$. E-satisfaction for arbitrary signed formula is defined in the usual way. The collection of all extended interpretations that e-satisfy $S : \mathcal{F}$ is denoted $\text{EMod}(S : \mathcal{F})$. An extended interpretation that e-satisfies an SFLP P is called an e-model of P , and the collection of all e-models is denoted $\text{EMod}(P)$.

For a given logic of signed formulas, the class of all extended interpretations forms a complete lattice under the ordering \sqsubseteq given by:

$$I_1 \sqsubseteq I_2 \text{ iff } I_2(A) \subseteq I_1(A) \text{ for any ground atom } A.$$

Care must be taken to observe that the ordering \sqsubseteq “reverses” the ordering \subseteq . This does not go against intuition. Since a sign S is interpreted disjunctively, i.e. can a formula evaluate to *one of* the values in S , the ordering \sqsubseteq is, in some sense, modeling definiteness. In other words, an extended interpretation is more definite than another if the first assigns a smaller set of truth values to each formula. The next lemmas are immediate.

Lemma 5 Suppose $I_2(A) \subseteq I_1(A)$ for any ground atom A . Then $I_2(\mathcal{F}) \subseteq I_1(\mathcal{F})$ for any ground Λ -formula \mathcal{F} .

Lemma 6 Suppose $I_1 \sqsubseteq I_2$. Then for any signed atom $S : \mathcal{F}$, $I_1 \in \text{EMod}(S : \mathcal{F})$ implies $I_2 \in \text{EMod}(S : \mathcal{F})$.

Various standard results of classical logic programming can now be proven for signed formula logic programs with respect to extended models, including the existence of a unique minimal e-model, and the existence of a monotone operator whose post-fixpoints coincide with the e-models of the program. We quickly state them for the sake of completeness in Theorems 7 and 8. The interesting non-standard result is Theorem 9, where the connection between models of P (i.e. $\text{Mod}(P)$), and the e-models of P (i.e. $\text{EMod}(P)$) is established.

Theorem 7 Suppose P is an SFLP. Then there is a unique minimal e-model \mathcal{E}_P of P under the ordering \sqsubseteq , given by

$$\mathcal{E}_P(A) = \bigcup_{I \in \text{EMod}(P)} I(A)$$

for any ground atom A . Moreover, \mathcal{E}_P corresponds to the least fixpoint of the operator W_P which maps from and to extended interpretations of P :

$$W_P(I)(A) = \bigcap \{S \mid S : A \leftarrow S_1 : \mathcal{F}_1 \ \& \ \dots \ \& \ S_n : \mathcal{F}_n \text{ is a ground instance of a clause in } P \text{ and } I \in \text{EMod}(S_i : \mathcal{F}_i), \text{ for each } 1 \leq i \leq n\}$$

The least fixpoint of W_P can be approximated by iterating W_P starting with the least extended interpretation that maps every Λ -formula to Δ . We use the following notation.⁶

⁶We will use the symbol \uparrow , which deviates slightly from the well-known \uparrow notation used in the logic programming literature, because in Section 4.1, \uparrow is used to denote upsets of partially ordered sets [DP90].

$$\begin{aligned}
W_P^{\uparrow 0} &= I_\Delta, \text{ where } I_\Delta(A) = \Delta \text{ for any ground atom } A. \\
W_P^{\uparrow n} &= W_P(W_P^{\uparrow n-1}), \text{ for } n \text{ a successor ordinal.} \\
W_P^{\uparrow n} &= \sqcup_{m \leq n} W_P^{\uparrow m}, \text{ for } n \text{ a limit ordinal.}
\end{aligned}$$

The symbol \sqcup denotes the least upper bound with respect to \sqsubseteq .

Theorem 8 $W_P^{\uparrow \omega} = \mathcal{E}_P$.

As with classical logic programming, $W_P^{\uparrow \omega}$ may be regarded as a bottom-up operational semantics for P . In a deductive database context (i.e. logic programs without function symbols), W_P provides the basis for view materialization. In the more general logic programming context, W_P is typically used for proving the completeness of top-down query answering procedures (See Theorem 16).

Theorem 9 $\{I(A) \mid I \in \text{Mod}(P)\}$ is an e-model of P .

Proof We define a function π that maps each interpretation of P to an extended interpretation of P as follows. For any ground atom A , $\pi(I)(A) = \{I(A)\}$.

For any ground formula \mathcal{F} , it is straightforward to verify by induction on the structure of \mathcal{F} that $\pi(I)(\mathcal{F}) = \{I(\mathcal{F})\}$. Thus:

Lemma 10 $I \in \text{Mod}(P)$ iff $\pi(I) \in \text{EMod}_s(P)$ where $\text{EMod}_s(P)$ denotes the collection of all e-models of P that assign only singleton sets to ground atoms.

Proof In the only if direction, let $C \equiv S : A \leftarrow S_1 : \mathcal{F}_1 \ \& \ \dots \ \& \ S_n : \mathcal{F}_n$ be a ground instance of a signed clause in P , and let $\pi(I)$ e-satisfy each $S_i : \mathcal{F}_i$, for $1 \leq i \leq n$. $\pi(I)(\mathcal{F}_i) \subseteq S_i$. Then $I(\mathcal{F}_i) \in S_i$. As $I \in \text{Mod}(P)$, it follows that $I(A) \in S$. Hence $\pi(I)(A) \subseteq S$ and that $\pi(I)$ is an e-model of C .

In the if direction, suppose $C \equiv S : A \leftarrow S_1 : \mathcal{F}_1 \ \& \ \dots \ \& \ S_n : \mathcal{F}_n$ is a ground instance of a signed clause in P , and suppose I satisfies each $S_i : \mathcal{F}_i$, for $1 \leq i \leq n$. $I(\mathcal{F}_i) \in S_i$. Then $\pi(I)(\mathcal{F}_i) \subseteq S_i$. As $\pi(I) \in \text{EMod}_s(P)$, $\pi(I)(A) \subseteq S$. Suppose $\pi(I) = \{\alpha\}$. Then $I(A) = \alpha \in S$. \square

The lemma proves, in other words, that π is a satisfiability preserving bijection from $\text{Mod}(P)$ to $\text{EMod}_s(P)$. A simple corollary to the lemma is that $\pi(I)$ is an e-model of P if I is a model of P .

We define Mod_P^* to be the extended interpretation that maps each ground atom A to $\{I(A) \mid I \in \text{Mod}(P)\}$. In view of Lemma 10, Mod_P^* may be expressed equivalently as $\text{Mod}_P^*(A) = \bigcup_{I \in \text{EMod}_s(P)} I(A)$, for any ground atom A .

Then, for any ground atom A ,

$$\text{Mod}_P^*(A) = \bigcup_{I \in \text{EMod}_s(P)} I(A) \subseteq \mathcal{E}_P(A).$$

This completes the proof of Theorem 9. \square

The theorem tells us that if we collect, for each ground atom A , the set of all truth values assigned by the original semantics of P , then the result is an extended interpretation that also e-satisfies P . The following corollary is immediate.

Corollary 11 $\mathcal{E}_P \sqsubseteq \{I(A) \mid I \in \text{Mod}(P)\}$ for any ground atom A .

Computationally, there are several important implications from the above discussion. First, query processing with respect to extended interpretations is no harder than classical logic programming due to the existence of a unique minimal e-model. This is a major advantage. In particular, the adaptation of constraint logic programming deduction to SFLP relies on the existence of \mathcal{E}_P . Second, the answers thus obtained is a conservative extension of the original semantics — the set $\text{Mod}(P)$ — since the set of truth values prescribed by the models of P are all contained in \mathcal{E}_P , for each ground atom A .

In general, an SFLP P may be translated into an equivalent Λ -atomic SFLP. The proof is based on Hähnle’s more general result which applies to arbitrary signed formulas [Häh93]. The key to proving the theorem is in realizing that a sign appearing in front of an arbitrary Λ -formulas may be systematically “driven inwards” whereby Λ -connectives are replaced with classical connectives.

Theorem 12 Suppose P is an SFLP. Then there is a Λ -atomic SFLP P' such that $\text{Mod}(P) = \text{Mod}(P')$.

Typically, the Λ -atomic SFLP P' will contain many more clauses than P . However, implementations of procedures to process queries with respect to Λ -atomic SFLPs are much easier since we may adapt the simple stack-based method for classical logic programming in a relatively straightforward manner.

Example 13 Let us reconsider the MVL over $\Delta = \{0, 0.2, 0.5, 0.8, 1\}$. Let \wedge denote the function *min*.⁷ Suppose an SFLP contains the signed clause

$$\{0.5\} : A \leftarrow \{0.5\} : (B \wedge C).$$

Then, as \wedge corresponds to *min*, $B \wedge C$ evaluates to 0.5 iff one of B and C evaluates to 0.5 while the other evaluates to 0.5, 0.8 or 1. Hence one Λ -atomic equivalent of the above signed clause is an SFLP that contains the following five clauses.

$$\begin{aligned} &\{0.5\} : A \leftarrow \{0.5\} : B \ \& \ \{0.5\} : C \\ &\{0.5\} : A \leftarrow \{0.5\} : B \ \& \ \{0.8\} : C \\ &\{0.5\} : A \leftarrow \{0.5\} : B \ \& \ \{1\} : C \\ &\{0.5\} : A \leftarrow \{1\} : B \ \& \ \{0.5\} : C \\ &\{0.5\} : A \leftarrow \{0.8\} : B \ \& \ \{0.5\} : C \end{aligned}$$

Observe that the translation is not optimal – the clauses may be merged into a more succinct representation.

2.4 Processing Signed Query

We assume only Λ -atomic SFLPs in this section. Consider the following simple intuition. Given an SFLP P containing the signed clause $S_1 : A \leftarrow \text{Body}$. Suppose we pose the signed query $\leftarrow S_2 : A$ which asks whether the truth value of A is contained in the set S_2 . If we are able to show that *Body* of the given clause holds,

⁷This is the usual interpretation associated with conjunction in fuzzy logics.

then $S_1 : A$ holds in which case it remains to show that A has one of the values in $S_2 - S_1$. In a refutational setting, this translates to the following resolution inference.

Definition 14 (Signed resolution) Let C be the signed clause

$$S_0 : A_0 \leftarrow S_1 : A_1 \ \& \ \dots \ \& \ S_n : A_n$$

and Q be the signed query $\leftarrow D_1 \ \& \ S : A \ \& \ D_2$ where D_1, D_2 are conjunctions of signed atoms. Suppose A_0 and A are unifiable via mgu θ . Then the query

$$\leftarrow (D_1 \ \& \ \nu(S, S_0) : A \ \& \ S_1 : B_1 \ \& \ \dots \ \& \ S_n : B_n \ \& \ D_2)\theta$$

is called the *signed resolvent* of C and Q , where the binary function ν takes two arguments, both subsets of Δ , and it returns a subset of Δ defined by

$$\nu(T_1, T_2) = \Delta - ((\Delta - T_1) \cap T_2).$$

The idea of signed resolution is as described before. The reason for the two subtractions performed in ν is to “reverse” the sign.

It is fairly straightforward to see that if a signed query contains a signed atom $S : A$ where S evaluates to Δ , then the atom may be removed from the query without affecting its set of models. We assume that such a simplification step is taken whenever possible. In particular the signed atom $\nu(S, S_0) : A$ in the signed resolvent above may be removed if $\nu(S, S_0) = \Delta$. A simple way to test whether $\nu(T_1, T_2) = \Delta$ is by observing that $\nu(T_1, T_2) = \Delta$ iff $T_2 \subseteq T_1$.

Example 15 Suppose we have the Λ -atomic SFLP P shown below, and that we are interested in determining whether r can evaluate to one of $\{0.8, 1\}$, i.e. $\{0.8, 1\} : r$.

1. $\{1, 0.8, 0.5\} : r \leftarrow \{1\} : p$
2. $\{1, 0.8, 0.2\} : r \leftarrow \{0.8, 1\} : q \ \& \ \{0.8\} : s$
3. $\{1\} : p \leftarrow$
4. $\{0.8\} : q \leftarrow$
5. $\{0.8\} : s \leftarrow$

This question may be answered by the following signed deduction.

$$\begin{array}{ll}
Q_0 : \leftarrow \{0.8, 1\} : r & \text{(Initial Query)} \\
Q_1 : \leftarrow \{1, 0.8, 0.2, 0\} : r \ \& \ \{1\} : p & (Q_0, 1) \\
Q_2 : \leftarrow \{1\} : p \ \& \ \{0.8, 1\} : q \ \& \ \{0.8\} : s & (Q_1, 2) \\
Q_3 : \leftarrow \{0.8, 1\} : q \ \& \ \{0.8\} : s & (Q_2, 3) \\
Q_4 : \leftarrow \{0.8\} : s & (Q_3, 4) \\
Q_5 : \leftarrow \square & (Q_4, 5)
\end{array}$$

Theorem 16 Signed resolution is sound and complete for Λ -atomic formulas with respect to \mathcal{E}_P .

Proof (Sketch) Soundness may be proved by simple case analysis. Completeness on the other hand is more involved. A proof based on first principle may follow the traditional proof of completeness for classical logic programming where we first show by induction that $\leftarrow S : A$ has a proof iff $W_P^{\uparrow n}(A) = S$ for some $n \in \omega$. In this case completeness follows by Theorem 8.

In [Lu96], completeness of signed resolution was proved by resorting to the soundness and completeness of an inference procedure — annotated resolution — developed in [KS92] for annotated logic programming. The semantical relationship of SFLP to annotated logic programming was first explored. It was then shown that any annotated deduction may be “simulated” using signed resolution. \square

In the case that an SFLP contains signs in which variables occur, testing whether a signed deduction succeeds involves simultaneous testing of whether several signs denote Δ .

Example 17 Let P be the SFLP defined over $\Delta = \{0, 0.5, 1\}$ as shown below.

$$\begin{aligned} (V \cap W) : A \leftarrow V : B \ \& \ W : C \\ \{1, 0\} : B \leftarrow \\ \{1, 0.5\} : C \leftarrow \end{aligned}$$

The query $\leftarrow \{1\} : A$ can be answered with the following signed refutation.

$$\begin{aligned} \leftarrow \nu(\{1\}, (V \cap W)) : A \ \& \ V : B \ \& \ W : C \\ \leftarrow \nu(\{1\}, (V \cap W)) : A \ \& \ \nu(V, \{1, 0\}) : B \ \& \ W : C \\ \leftarrow \nu(\{1\}, (V \cap W)) : A \ \& \ \nu(V, \{1, 0\}) : B \ \& \ \nu(W, \{1, 0.5\}) : C \end{aligned}$$

In each of the queries, if the variables that occur in any of the signs can be consistently replaced by subsets of Δ so that each sign evaluates to Δ , then the deduction terminates. A careful inspection reveals that only the last of the above queries can be so substituted with $V = \{1, 0\}$ and $W = \{1, 0.5\}$.

3 Constraint Logic Programming

Query processing based on signed resolution, as presented in Section 2.4, is theoretically straightforward. However, complex implementation issues arise due to the possibility that a signed atom resolved upon may remain in the resolvent (see Section 3.3). In light of this, we seek to find a connection between SFLP and an existing logic programming formalism with a simpler operational semantics. If such a connection can be established, then we benefit first of all by having available existing implementation techniques, and secondly the operational simplicity of the existing logic programming formalism may help to clarify issues that are specific to SFLPs. This is the motivation behind the work described in this section. Specifically, we show a transformation of SFLP to constraint logic programming (CLP) over the domain of $\mathcal{P}(\Delta)$; the “upside down” powerset lattice over Δ . This translation will enable the application of CLP query processing techniques, together with set constraint solving methods (e.g. [AW92]), to SFLPs.

The work on constraint logic programming was pioneered by Jaffar and Lassez [JL87] and earlier by Colmerauer in a more restricted form [COL82]. The integration of constraint solving into the semantics of logic programming significantly extends the applicability of logic programming to domains once thought unsuited for logic programming. We give a very brief summary of the theory of CLP.

A CLP consists of definite horn clauses augmented with constraints over some specified domain. We assume a first order language \mathcal{L} . A structure over \mathcal{L} , Σ , is a collection of objects D (i.e. the *carrier*), and an assignment of the symbols of \mathcal{L} to the functions and the relations on D .

The predicate symbols in \mathcal{L} are divided into two disjoint sets, Π_c and Π_p . An *atomic constraint* is an atom formed in the usual way from the symbols of \mathcal{L} , but whose predicate symbol belongs to the set Π_c . A *constraint* is a well-formed formula built from atomic constraints, logical connectives, and quantifiers. A *constraint clause* is an expression of the form

$$A \leftarrow \Xi \parallel B_1 \& \dots \& B_n$$

where Ξ is a constraint, and A, B_1, \dots, B_n are atoms whose head symbols belong to Π_p . A *constraint logic program* is then a finite collection of constraint clauses.

A Σ -valuation is a mapping from variables in \mathcal{L} to elements in D , extended straightforwardly to arbitrary expressions in \mathcal{L} . A constraint Ξ is *solvable* if there is a Σ -valuation that when applied to Ξ , yields a relation $\Xi\theta$ over D that is true. The Σ -valuation is said to be a *solution* of Ξ .

A Σ -base of a constraint logic program P is the set

$$\{p(\vec{x})\theta \mid p \in \Pi_p \text{ and } \theta \text{ is a } \Sigma\text{-valuation}\}.$$

A Σ -interpretation is a subset of the Σ -base, and a Σ -*model* I of a constraint logic program P is a Σ -interpretation such that for every constraint clause $A \leftarrow \Xi \parallel B_1 \& \dots \& B_n$ in P , if θ is a Σ -valuation that is a solution of Ξ and $B_i\theta \in I$ for $i = 1, \dots, n$, then $A\theta \in I$. As Jaffar and Lassez showed [JL87], a CLP program is assured of a least Σ -model with respect to set containment. This least model may be approximated through a monotone operator that is analogous to the T_P operator of classical logic programming.

To simplify the presentation, we consider in the remainder of the section, only those SFLPs in which all Λ -formulas are propositional; signs that appear in signed clauses may still contain variables. This assumption is made only for the sake of brevity. All of the discussion lifts to non-ground SFLPs easily.

3.1 SFLP to CLP

As mentioned above, there is a natural representation of a Λ -atomic SFLP as a CLP program over the domain $\mathcal{P}(\Delta)$.

Definition 18 (Constraint Form) Given a Λ -atomic SFLP P , the *constraint form* of P is the CLP, denoted $CF(P)$, made up of the following three collections of non-ground CLP clauses.

1. $A(V) \leftarrow S \subseteq V \parallel B_1(S_1) \& \dots \& B_n(S_n)$
where the signed clause $S : A \leftarrow S_1 : B_1 \& \dots \& S_n : B_n$ is in P .
2. $A(V) \leftarrow (V_1 \cap V_2) \subseteq V \parallel A(V_1) \& A(V_2)$
where A is any atom that occurs in P .
3. $A(V) \leftarrow V = \Delta$
where A is any atom that occurs in P .

The variables V, V_1 and V_2 range over non-empty subsets of Δ . The constraint form of a signed query $Q = \leftarrow S_1 : B_1 \& \dots \& S_n : B_n$ is obtained as a special case of the first step above. That is, $CF(Q) = \leftarrow B_1(S_1) \& \dots \& B_n(S_n)$.

Example 19 Consider the SFLP P from the last example in Section 2.4.

$$\begin{aligned} (V \cap W) : A \leftarrow V : B \& W : C \\ \{1, 0\} : B \leftarrow \\ \{1, 0.5\} : C \leftarrow \end{aligned}$$

The constraint form of P is the $CF(P)$ below.

$$\begin{aligned} A(U) \leftarrow (V \cap W) \subseteq U \parallel B(V) \& C(W) \\ A(U) \leftarrow (U_1 \cap U_2) \subseteq U \parallel A(U_1) \& A(U_2) \\ A(U) \leftarrow U = \Delta \\ B(U) \leftarrow \{1, 0\} \subseteq U \\ B(U) \leftarrow (U_1 \cap U_2) \subseteq U \parallel B(U_1) \& B(U_2) \\ B(U) \leftarrow U = \Delta \\ C(U) \leftarrow \{1, 0.5\} \subseteq U \\ C(U) \leftarrow (U_1 \cap U_2) \subseteq U \parallel C(U_1) \& C(U_2) \\ C(U) \leftarrow U = \Delta \end{aligned}$$

Note the clauses $B(U) \leftarrow U = \Delta$ and $C(U) \leftarrow U = \Delta$ are subsumed by $B(U) \leftarrow \{1, 0\} \subseteq U$ and $C(U) \leftarrow \{1, 0.5\} \subseteq U$ respectively, and hence they may be removed.

The extended interpretations for P and the CLP-interpretations of $CF(P)$ naturally correspond, in the sense of satisfiability, via the following mapping ψ . For any ground atom A and variable free sign S :

$$A(S) \in \psi(I) \text{ iff } I(A) \subseteq S.$$

Hence if we have an SFLP written over the truth values $\Delta = \{0, 0.5, 1\}$, and I is the interpretation that maps A to $\{0.5\}$, then

$$\psi(I) = \{A(\{0.5\}), A(\{0, 0.5\}), A(\{1, 0.5\}), A(\{0, 0.5, 1\})\}$$

Theorem 20 I is an e-model of P iff $\psi(I)$ is a CLP model of $CF(P)$.

Proof only if: Consider a ground instance C of a clause in $CF(P)$ whose body is satisfied by $\psi(I)$. There are three cases to consider, each corresponding to a case in the construction of $CF(P)$ (see the definition of constraint form). In the first case, C has the form

$$A(S_0) \leftarrow S \subseteq S_0 \parallel B_1(S_1) \& \dots \& B_n(S_n)$$

where $S \subseteq S_0$ holds. There is a corresponding instance C_0 in P of the form

$$S : A \leftarrow S_1 : B_1 \& \dots \& S_n : B_n$$

As $B_i(S_i) \in \psi(I)$, $I(B_i) \subseteq S_i$. Then $I(A) \subseteq S$ since I is an e-model of P . It follows by transitivity that $I(A) \subseteq S_0$. By the definition of ψ , $A(S_0) \in \psi(I)$.

In the second case, C has the form

$$A(S_0) \leftarrow (S_1 \cap S_2) \subseteq S_0 \parallel A(S_1) \& A(S_2)$$

where $A(S_1)$ and $A(S_2)$ are both contained in $\psi(I)$, and $(S_1 \cap S_2) \subseteq S_0$ holds. For $i = 1, 2$, $I(A) \subseteq S_i$. Consequently, $I(A) \subseteq (S_1 \cap S_2)$. By the transitivity of set inclusion, $I(A) \subseteq S_0$ and $A(S_0) \in \psi(I)$.

In the last case, C has the form

$$A(S_0) \leftarrow S_0 = \Delta$$

where $S_0 = \Delta$ holds. As $I(A) \subseteq \Delta$ holds trivially, $A(\Delta) \in \psi(I)$.

if: Suppose

$$S_0 : A \leftarrow S_1 : B_1 \& \dots \& S_n : B_n$$

is a ground instance of a clause in P where $I \in \text{EMod}(S_i : B_i)$ for each $1 \leq i \leq n$. The corresponding constraint form of the clause can be represented via the schema

$$A(V_0) \leftarrow S_0 \subseteq V_0 \parallel B_1(S_1) \& \dots \& B_n(S_n)$$

where V_0 stands for any subset of Δ that contains S_0 . Clearly, one particular instance of V_0 is S_0 . Hence as $\psi(I)$ is a CLP model of $CF(P)$, $A(S_0) \in \psi(I)$. It follows that $I(A) \subseteq S_0$. \square

Thus there is an isomorphism between the collection of CLP models of $CF(P)$, and the collection of e-models of an SFLP P . We have the following corollary.

Corollary 21 $\psi(\mathcal{E}_P)$ coincides with the least CLP model of $CF(P)$.

3.2 Query Processing Revisited

A query in a CLP language is an expression of the form

$$\leftarrow \Xi \parallel A_1 \& \dots \& A_n$$

where Ξ is a constraint, and A_1, \dots, A_n are atoms. Query processing in CLP combines classical logic programming backtracking with constraint solving. We call such a procedure CLP-resolution. At each step of a CLP-deduction, the solvability of the constraint part of the current goal is required. Considerations such as incremental computation is useful in practice. As a starting point, we examine the following example.

Example 22 Recall the example in Section 3.1. A signed refutation of the query $\leftarrow \{1\} : A$ was given earlier in the example in Section 2.4. The corresponding CLP query is the expression $\leftarrow A(\{1\})$ and may be refuted as follows.

$$\begin{aligned} &\leftarrow (V \cap W) \subseteq \{1\} \parallel B(V) \ \& \ C(W) \\ &\leftarrow (V \cap W) \subseteq \{1\} \ \& \ \{1, 0\} \subseteq V \parallel C(W) \\ &\leftarrow (V \cap W) \subseteq \{1\} \ \& \ \{1, 0\} \subseteq V \ \& \ \{1, 0.5\} \subseteq W. \end{aligned}$$

The constraint appearing in each step of the deduction is solvable.

Consider another example in which the extra clauses of the constraint form, introduced via the second step of the definition of the constraint form, are used.

Example 23 Let $\Delta = \{0, 0.5, 1\}$ and let P be the SFLP below to the left. $CF(P)$ is the CLP shown on the right.

SFLP P	Corresponding CLP $CF(P)$
$\{1, 0\} : A \leftarrow$	$A(V) \leftarrow \{1, 0\} \subseteq V$
$\{1, 0.5\} : A \leftarrow$	$A(V) \leftarrow \{1, 0.5\} \subseteq V$
	$A(V) \leftarrow (U_1 \cap U_2) \subseteq V \parallel A(U_1) \ \& \ A(U_2)$

The query $Q = \leftarrow \{1\} : A$ may be refuted using both signed resolution and CLP resolution shown below.

Signed Deduction	CLP Deduction
$\leftarrow \{1\} : A$	$\leftarrow A(\{1\})$
$\leftarrow \nu(\{1\}, \{1, 0\}) : A$	$\leftarrow (U_1 \cap U_2) \subseteq \{1\} \parallel A(U_1) \ \& \ A(U_2)$
$\leftarrow \nu(\nu(\{1\}, \{1, 0\}), \{1, 0.5\}) : A$	$\leftarrow (U_1 \cap U_2) \subseteq \{1\} \ \& \ \{1, 0\} \subseteq U_1 \parallel A(U_2)$
	$\leftarrow (U_1 \cap U_2) \subseteq \{1\} \ \& \ \{1, 0\} \subseteq U_1 \ \& \ \{1, 0.5\} \subseteq U_2$

The next theorem follows from Theorem 20 and the fact that CLP-deduction is sound and complete.

Theorem 24 (Soundness and Completeness) Suppose P is an SFLP and $\leftarrow Q$ is a signed query. Then $P \models Q$ iff there is a CLP-deduction of the empty clause from the program $CF(P)$ beginning with the query $CF(Q)$.

Proof CLP-deduction is sound and complete with respect to the least Σ -model of a given constraint logic program. By Corollary 21, it is sound and complete with respect to the least model \mathcal{E}_P . \square

The completeness result is perhaps more appropriately called weak completeness since we have based the result on the completeness of CLP-deduction. Thus the result tells us that whenever there is a proof of $\leftarrow Q$ from P , then there *exists* a CLP-proof of $CF(Q)$ from $CF(P)$. A strong completeness result appears possible

where given a proof R of $\leftarrow Q$ from P , we can construct a CLP-proof $CF(R)$ of $CF(Q)$ from $CF(P)$ which “simulates” R in a step-by-step manner. This is an interesting (but perhaps not crucial) topic that we will examine in the future.

3.3 Search Space Considerations

3.3.1 Literal and Clause Selections

Annotated logic, as studied in [BS89, KL92, KS92], has been shown to relate to signed formulas in a natural way [LMR97]. In [LL94], a query processing procedure for annotated logic programming was introduced that shares certain characteristics with the signed resolution procedure developed in this paper. In particular, since the signed atom resolved upon in signed resolution is not necessarily removed in the resolvent, and since signed atoms may share variables in their signs, the independence of literal selection in classical logic programming [Llo88] no longer holds. Hence a function that fairly chooses signed atoms from queries appearing in a deduction is required to ensure completeness. Viewing an SFLP as a CLP further sheds light on this issue.

Consider the program P from the last example in Section 2.4. Suppose we adopt the strategy of selecting the leftmost signed atom in each deduction step, no proof of the query $\leftarrow \{1\} : A$ can be obtained. In the proof exhibited in Section 2.4, signed resolution was applied to the first signed atom in the first query, the second signed atom in the second query, and the third signed atom in the last query.

Now viewing the program in its constraint form (see example in Section 3.1), it can be seen that the problem of atom selection is transformed into the (traditional) problem of fair clause selection. The query in question has the constraint form $\leftarrow A(\{1\})$, and can be resolved easily by CLP-resolution using the usual Prolog left-most atom selection (see the first example in Section 3.2). Indeed, any other atom selection strategy will work. Hence by considering SFLP as CLP, we have traded off selection strategies on the signed atoms of queries for selection strategies on the clauses in the transformed program.

A closer examination reveals that in this example, the structure of the search space induced by CLP-resolution can be obtained by, in each step, shuffling the signed atom resolved upon to the rightmost part of the resolvent prior to the next deduction.⁸

3.3.2 Partial Solvability for Regular Signed Logics

Most applications of the logic of signed formulas — annotated and fuzzy logic being special instances — adopt underlying truth value sets equipped with at least a partial ordering. Moreover, the most frequently studied classes of signs are the so called *regular logics*, where the signs are assumed to either be the upset of a single truth value, the downset of a single truth value, or the complements of such sets.⁹

⁸This amounts to the strategy adopted by Frühwirth in his implementation of annotated logic programming [Fr94].

⁹The upset of an element μ in a partially order set $\langle \Delta, \preceq \rangle$ is the set $\{\beta \in \Delta \mid \mu \preceq \beta\}$, and is denoted $\uparrow \mu$.

Certainly for both annotated and fuzzy logics, the restriction to regular signs are always assumed. With the assumption, a sign is completely characterized by an element of Δ , and we may write the signed atom $(\uparrow \mu) : p$ as simply $\mu : p$.

Now in lattice-based formalisms (e.g. [Fit91], [KS92], [LNS96]), regular logics have the property that if $\mu_1 : p$ and $\mu_2 : p$ are entailed with respect to the least e-model \mathcal{E}_P , then $\mu : p$, where μ is the join of μ_1 and μ_2 , is also true in \mathcal{E}_P . This is a special case of the Reduction Lemma of the logic of signed formulas which states that $S_1 : p \ \& \ S_2 : p$ is e-equivalent to $(S_1 \cap S_2) : p$. As it turns out, this simple property is at the heart of most research into deduction techniques for multiple-valued logic programming. To answer the atom $\mu : p$ in a query, methods for gathering two facts $\mu_1 : p$ and $\mu_2 : p$ to form a sufficiently large truth value is required. That is, the join of μ_1 and μ_2 must be above μ . This is the essential idea behind the reduction inference rule of [KS92]. Conversely, it is possible to replace the atom $\mu : p$ by an equivalent set of atoms, each one answerable with existing program clauses. Such a decomposition technique has been investigated in [Fr94, Mes96]. None of these techniques, however, take into account the effect of making an inference. In many cases, an application of any one of the inference rules results in a query which is subsumed by some previous query. Preventing such irrelevant subqueries from being generated will greatly enhance the effectiveness of each of the inference rules. Here, we briefly examine a restriction technique through the notion of *partial solvability*. The concept naturally fits into the use of constraint solving in CLP, and appears applicable to a wide variety of lattice-based truth value sets.

Definition 25 Given a lattice $\langle \Delta, \preceq \rangle$ and two elements $\mu, \beta \in \Delta$, we denote $\mathcal{U}(\mu, \beta)$ the set of minimal elements (with respect to \preceq) in the following set.

$$\{\gamma \in \Delta \mid \mu \preceq \sqcup(\beta, \gamma)\}$$

Then, suppose $\mu \not\preceq \beta$, we say that β *partially solves* μ if for each $\gamma \in \mathcal{U}(\mu, \beta)$, $\mu \not\preceq \gamma$.

The intuition is that in the case where a signed atom $\mu : p$ is to be answered, then attention should be restricted to those signed clauses whose heads have atoms that unify with p , and have signs that at least partially solve μ . We provide some examples to illustrate this idea.

Let Δ denote the set FOUR (see Figure 1) where the ordering of the elements is given by the reflexive and transitive closure of the base ordering $\mathbf{t} \prec \top$, $\mathbf{f} \prec \top$, $\perp \prec \mathbf{f}$, and $\perp \prec \mathbf{t}$. Given the SFLP $\{\mathbf{t} : p \leftarrow, \mathbf{f} : p \leftarrow\}$, only the first signed clause should be considered when answering the query $\leftarrow \mathbf{t} : p$ since \mathbf{t} , but not \mathbf{f} , partially solves \mathbf{t} . In the case of \mathbf{f} , $\mathcal{U}(\mathbf{t}, \mathbf{f}) = \{\mathbf{t}\}$. Since the only element in the set, \mathbf{t} , does not satisfy $\mathbf{t} \not\preceq \mathbf{t}$, the condition for partial solvability does not hold. Without the restriction to partial solvability, the reduction inference of [KS92] will potentially generate the new fact $\top : p \leftarrow$ from the two program clauses which can be used to answer the query, but it is unnecessary since the query is already answerable from existing clauses. Similarly, the decomposition inference of [Fr94] will potentially generate the subquery $\leftarrow \mathbf{f} : p$ by decomposing the original query which results in

an extra inference step. In both cases, a careful test for partial solvability would avoid the unnecessary inference.

The above example also raises the interesting question of what the relationship is between solvability and partial solvability. The following lemmas shows that the name *partially solvable* is well-chosen.

Lemma 26 Suppose $\mu \neq \perp$ and $\mu \preceq \beta$ is solvable. Then β partially solves μ .

Proof Without loss of generality, assume that both μ and β are variable free. Clearly, for any element $\gamma \in \Delta$, $\mu \preceq \sqcup\{\gamma, \beta\}$. This holds in particular for \perp . By the definition of \mathcal{U} , $\mathcal{U}(\mu, \beta) = \{\perp\}$. Thus, the condition for partial solvability is trivially satisfied since $\mu \not\preceq \perp$. \square

Consider another example specified over a slightly larger space of truth values. Let Δ denote the set $\{\mathbf{t}, \mathbf{dt}, \mathbf{f}, \mathbf{df}, \top, \perp\}$ where the ordering is given by the reflexive and transitive closure of the base ordering $\mathbf{t} \prec \top$, $\mathbf{f} \prec \top$, $\mathbf{dt} \prec \mathbf{t}$, $\mathbf{df} \prec \mathbf{f}$, $\perp \prec \mathbf{df}$, and $\perp \prec \mathbf{dt}$.¹⁰ Let P denote the SFLP $\{\mathbf{dt} : p \leftarrow, \mathbf{f} : p \leftarrow\}$.

The fact $\top : p$ is entailed by the program, and hence the query $\leftarrow \mathbf{t} : p$ should yield a proof from the program. Interestingly, unlike the previous example, both signed clauses in the program will be admitted under the restriction based on partial solvability. In particular, \mathbf{f} now partially solves \mathbf{t} since $\mathcal{U}(\mathbf{t}, \mathbf{f}) = \{\mathbf{dt}\}$ which does satisfy the condition that for every element in the set, i.e. \mathbf{dt} , $\mathbf{t} \not\preceq \mathbf{dt}$.

This example illustrates the subtleties involved in the notion of partial solvability and its dependence on the underlying lattice. The concept probably bears a connection to the Birkhoff Representation Theorem in the case of distributive lattices [DP90], but a closer examination of this relationship will be necessary and is slated for future research.

4 Applications

4.1 Bilattice Logic Programming

This section uses SFLP as a tool for analyzing finitely valued bilattice logic programs [Fit91]. For simplicity, we again focus on variable-free formulas only.

We are interested in finding, for each bilattice logic program P , an SFLP $SFB(P)$ that can be used to answer questions of the form:

Given bilattice logic program P , a sign S and an atom A , can A evaluate to some value in S , under the intended meaning $\llbracket P \rrbracket$ of P ?

In bilattice logic programming, $\llbracket P \rrbracket$ is typically associated with a single interpretation — though several acceptable choices exist. Hence formally, the relationship desired is

$$SFB(P) \models S : \mathcal{F} \text{ iff } \llbracket P \rrbracket(\mathcal{F}) \in S.$$

¹⁰The values \mathbf{dt} and \mathbf{df} can be read intuitively as default true and default false.

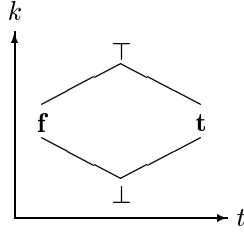


Figure 1: The Bilattice FOUR.

A logic of bilattice Λ_B is a multiple-valued logic whose set of truth values Δ is a bilattice — a set equipped with two orderings, \preceq_k and \preceq_t , each inducing a complete lattice on the elements in Δ . Δ contains four distinguished elements: \perp , \top , **f**, and **t**, which denote respectively the least and the greatest elements with respect to \preceq_k , and \preceq_t . FOUR shown in Figure 1 is therefore the smallest non-trivial bilattice. The least upper bound and greatest lower bound operations with respect to the ordering \preceq_k are denoted \otimes and \oplus respectively, while with respect to the ordering \preceq_t , they are denoted \vee and \wedge respectively. The symbol \neg denotes negation, and satisfies the properties $a \preceq_k b \Rightarrow \neg a \preceq_k \neg b$ and $a \preceq_t b \Rightarrow \neg b \preceq_t \neg a$. Furthermore, Δ satisfies the *interlacing* condition, which says that each of the operations \vee, \wedge is monotone with respect to the ordering \preceq_k , and similarly, each of the operations \oplus, \otimes is monotone with respect to the ordering \preceq_t [Fit91].

There are a number of constants in the language of Λ_B . A *body formula* is built out of atomic formulas, constants, and the connectives $\neg, \vee, \wedge, \otimes$, and \oplus . A Λ_B -clause is an expression of the form $A \leftarrow \mathcal{F}$ where A is an atomic formula, and \mathcal{F} is a body formula. A finite set of Λ_B -clauses is called a bilattice logic program.

A Λ_B -interpretation I assigns a value in Δ to each constant, each ground atom, and are extended to each body formula according to the functions represented by the operators $\neg, \vee, \wedge, \otimes$, and \oplus . It is assumed that all interpretations evaluate the constants in the same way, in particular *true* is a constant that evaluates to **t**, and *false* is a constant that evaluates to **f** under any Λ_B -interpretation.

As mentioned, several reasonable possibilities exist for the intended meaning of a bilattice logic program P . We focus on the one provided by the operator Φ_P , given by Fitting in [Fit91], which maps from and to Λ_B -interpretations.

Given a Λ_B -interpretation I , $\Phi_P(I)$ is the Λ_B -interpretation that assigns to each atomic formula A , a truth value determined by the following.

$$\Phi_P(I)(A) = \bigvee \{I(\mathcal{F}) \mid A \leftarrow \mathcal{F} \text{ a ground instance in } P\}$$

Φ_P is monotone with respect to \preceq_k , and it is monotone with respect to \preceq_t provided that the symbol \neg does not appear in P . In each case, the existence of the least fixed point of Φ_P is guaranteed by Tarski's theorem on monotone operators over lattices [TAR55]. We denote $lfp_t(\Phi_P)$ the least fixed point of Φ_P under the \preceq_t ordering.

Example 27 Consider the bilattice logic program P over FOUR

$$r \leftarrow p \otimes (q \vee t)$$

$$\begin{aligned}
s &\leftarrow t \oplus p \\
u &\leftarrow p \otimes t \\
p &\leftarrow true \\
q &\leftarrow true
\end{aligned}$$

$lfp_t(\Phi_P)$ assigns \mathbf{t} to each of r , p , and q . It assigns \top to s , \perp to u , and \mathbf{f} to t .

The fixed point $lfp_t(\Phi_P)$ establishes $\llbracket P \rrbracket$. It tells us that for each ground atom A , the truth value of A is *at least* $lfp_t(\Phi_P)(A)$, with respect to the ordering \preceq_t . To mimic this semantic using an SFLP, the signs that we choose must allow the iteration of the operator W to reflect $lfp_t(\Phi_P)$. It turns out that the signs of interest are of the form $\uparrow_t \mu = \{\beta \in \Delta \mid \mu \preceq_t \beta\}$.

Definition 28 (SFB) Let P be a bilattice logic program P . $SFB(P)$ is the SFLP consisting of the following set of signed clauses.

$$\begin{aligned}
&\{\uparrow_t \mathbf{t} : A \leftarrow \mid A \leftarrow true \in P\} \cup \\
&\{\uparrow_t \mathbf{f} : A \leftarrow \mid A \leftarrow false \in P\} \cup \\
&\{\uparrow_t V : A \leftarrow V : \mathcal{F} \mid A \leftarrow \mathcal{F} \in P \text{ where } \mathcal{F} \text{ is a complex body formula, and} \\
&\quad V \text{ is a variable that does not occur in the clause}\} \cup \\
&\{\uparrow_t \mathbf{t} : true \leftarrow, \uparrow_t \mathbf{f} : false \leftarrow\}
\end{aligned}$$

The last set in the above union ensures that the constants *true* and *false* are interpreted faithfully in $SFB(P)$.

Example 29 Continuing with the previous example, $SFB(P)$ contains the following signed clauses.

$$\begin{aligned}
&\uparrow_t V : r \leftarrow V : (p \otimes (q \vee t)) \\
&\uparrow_t V : s \leftarrow V : (t \oplus p) \\
&\uparrow_t V : u \leftarrow V : (p \otimes t) \\
&\{\mathbf{t}\} : p \leftarrow \\
&\{\mathbf{t}\} : q \leftarrow \\
&\{\mathbf{t}, \mathbf{f}, \perp, \top\} : false \leftarrow \\
&\{\mathbf{t}\} : true \leftarrow
\end{aligned}$$

The function $lfp(W_{SFB(P)})$ is shown below.

<i>true</i>	<i>false</i>	<i>u</i>	<i>t</i>	<i>s</i>	<i>r</i>	<i>q</i>	<i>p</i>
$\{\mathbf{t}\}$	$\{\mathbf{t}, \mathbf{f}, \perp, \top\}$	$\{\perp, \mathbf{t}\}$	$\{\mathbf{t}, \mathbf{f}, \perp, \top\}$	$\{\top, \mathbf{t}\}$	$\{\mathbf{t}\}$	$\{\mathbf{t}\}$	$\{\mathbf{t}\}$

For each proposition A , $lfp(W_{SFB(P)})(A) = \uparrow_t \mu$ iff $lfp_t(\Phi_P)(A) = \mu$. Indeed, this relationship holds for any negation-free bilattice logic program, as the next theorem indicates. First, given S and T subsets of Δ and $*$ $\in \{\oplus, \otimes, \vee, \wedge\}$, we denote

$$S * T = \{\mu_0 * \beta_0 \mid \mu_0 \in S, \beta_0 \in T\}.$$

Lemma 30 Suppose $S = \uparrow_t \mu$ and $T = \uparrow_t \beta$ and $*$ $\in \{\oplus, \otimes, \vee, \wedge\}$. Then $S * T \subseteq \uparrow_t (\mu * \beta)$.

Proof In the case where $*$ \in $\{\wedge, \vee\}$, the lemma follows easily. We thus consider the case where $*$ $=$ \oplus . The case where $*$ $=$ \otimes follows by duality.

Suppose $\gamma \in S \oplus T$, it suffices to show that $\oplus\{\mu, \beta\} \preceq_t \gamma$. We have $\gamma = \mu_0 \oplus \beta_0$ for some $\mu \preceq_t \mu_0$ and $\beta \preceq_t \beta_0$. That is, $\oplus\{\mu_0, \beta_0\}$ is a \preceq_k -upper bound of both μ and β , and hence of $\oplus\{\mu, \beta\}$. By the interlacing condition, it is a \preceq_t -upper bound of $\oplus\{\mu, \beta\}$ as well. \square

This lemma does not apply in the case of the negation operator (i.e. \neg). Hence, the theorem below holds only if we assume that P is negation-free.

Theorem 31 Suppose P is a negation-free bilattice logic program. Then $lfp_t(\Phi_P)(A) = \mu$ iff $lfp(W_{SFB(P)})$ e-satisfies the signed atom $\uparrow_t \mu : A$.

Proof Recall $lfp_t(\Phi_P) = \Phi_P^{\uparrow\omega}$ and $lfp(W_{SFB(P)}) = W_{SFB(P)}^{\uparrow\omega}$.

Suppose $\Phi_P^{\uparrow\omega}(A) = \mu$. Then $\Phi_P^{\uparrow n}(A) = \mu$ for some $n \in \omega$. The proof is by induction on n that $\uparrow_t \mu : A$ is e-satisfied by $W_{SFB(P)}^{\uparrow\omega}$. The base case holds trivially. In the inductive case, there is a set of clauses

$$\begin{aligned} A &\leftarrow \mathcal{F}_1 \\ A &\leftarrow \mathcal{F}_2 \\ &\dots \\ A &\leftarrow \mathcal{F}_n \end{aligned}$$

such that $\bigvee\{\Phi_P^{\uparrow n}(\mathcal{F})\} = \mu$. We may simplify the proof by assuming that the above set of clauses is written in the form of a single clause

$$A \leftarrow \bigvee_{i=1}^n \mathcal{F}_i.$$

By the induction hypothesis, for each atom B in $\bigvee_{i=1}^n \mathcal{F}_i$, $\Phi_P^{\uparrow n}(B) = \beta$ iff $W_{SFB(P)}^{\uparrow\omega}$ e-satisfies $\uparrow_t \beta : B$. By Lemma 30, the sign assigned to $\bigvee_{i=1}^n (\mathcal{F}_i)$ by $W_{SFB(P)}^{\uparrow\omega}$ is a subset S of $\uparrow_t \mu$ such that $\mu \in S$. It follows that, as the signed clause

$$\uparrow_t V : A \leftarrow V : \bigvee_{i=1}^n \mathcal{F}_i$$

is contained in $SFB(P)$, $W_{SFB(P)}^{\uparrow\omega}$ assigns to A $\uparrow_t S = \uparrow_t \mu$.

In the reverse direction, suppose $W_{SFB(P)}^{\uparrow\omega}$ e-satisfies $\uparrow_t \mu : A$. Then $W_{SFB(P)}^{\uparrow n}$ e-satisfies $\uparrow_t \mu : A$ for some $n \in \omega$. The proof also is by induction on n that $\Phi_P^{\uparrow\omega}(A) = \mu$. The key is in realizing that in the inductive case, for each clause

$$\uparrow_t V : A \leftarrow V : \mathcal{F}$$

in $SFB(P)$, each atom in \mathcal{F} is assigned a \preceq_t -upset in $W_{SFB(P)}^{\uparrow n}$. By a simple induction on the structure of \mathcal{F} , we may conclude that the least \preceq_t -element in the sign of \mathcal{F} is exactly the truth value computed according to values of the atoms and the operators in \mathcal{F} . \square

$SFB(P)$ can now be used to answer questions about P under the meaning $\llbracket P \rrbracket$. Given an atom A and a subset S of Δ , the question of whether A evaluates to S under the intended meaning of P can be expressed as a signed query $\leftarrow S : A$, and an answer may be obtained through procedures such as signed resolution.

4.2 Assumption-Based Truth Maintenance

A lattice of truth values similar to $\mathcal{P}(\Delta)$ of Section 3 using the reverse subset ordering appears in *assumption-based* truth maintenance systems [DeK86]. The powerset $\mathcal{P}(\mathcal{A})$ of a propositional language \mathcal{A} forms a complete lattice when ordered under the reverse subset ordering, denoted \preceq . The basic idea of coding the assumptions under which a proposition holds into its truth values was originally proposed by Ginsberg [Gin88], but his work was carried out in the context of multiple-valued logic theorem proving.

Here, we provide a *semantical characterization* of assumption-based reason maintenance by means of signed formulas. In addition to gaining theoretical insights, since we have revealed that signed formula logic programs can be operationalized by means of constraint logic programs, a possible parallel implementation of an assumption-based reason maintenance system by means of concurrent constraint logic programming languages [Sar91] will therefore be possible.

Informally *assumptions* are primitive data from which all other data can be derived through the use of *justifications*. A justification in the original ATMS is just a propositional Horn-clause without negation. A node consists of a datum, label and justifications. To illustrate the difference between a justification in the ATMS and a clause in the problem solver, consider the following example from DeKleer: the deduction of $Q(a)$ from $P(a)$ and $Q(X) \leftarrow P(X)$ is recorded as a *justification* $\gamma_{P(a)}, \gamma_{Q(X) \leftarrow P(X)} \Rightarrow \gamma_{Q(a)}$ where γ_{datum} refers to a datum in the truth maintenance system. An ATMS determines beliefs based on the justifications so far encountered *not with respect to the logic of the problem solver*. Therefore, the propositional symbols occurring within labels are uninterpreted symbols and justifications are material implications.

In our approach, the underlying logic of the problem solver does the bookkeeping performed by the reason maintenance system. Since the problem solver is a signed logic program, the inferences and data to be recorded by the reason maintenance are restricted. The problem solver datum is either derived, or it is a program clause. An *environment* is a set of given assumptions and a *label* is a set of environments. Formally, a label is a propositional formula in disjunctive normal form, and a datum holds in a given environment if it can be derived from the justifications and the environment. A *Nogood* is a minimal assumption set such that the assumptions contained within cannot be true together with respect to the set of justifications. An ATMS *context* is the set formed by the assumptions of a consistent environment combined with all nodes derivable from those assumptions.

One particular difference between our formulation and the original ATMS is that our semantics does not capture the removal of environments subsumed by Nogoods (labels of atoms with inconsistent truth values). In other words, the semantics of an SFLP is monotonic in contrast to the ATMS where just discovered

Nogoods are to be removed. In our case, a Nogood is simply an empty clause with a nonempty sign.

The key idea in redefining assumption-based reason maintenance¹¹ as signed logic program is to write labels in the form of signs, i.e. define a suitable set of truth values Δ . In this sense our reason maintenance system departs from most other systems as it amalgamates the inference machine of the problem solver and the reason maintenance component. Following the argument of [MS88], a reason maintenance system itself should be able to detect inconsistencies and to compute automatically the dependencies of new beliefs from older ones instead of just recording them passively. Besides, the amount of time spent for communication between the problem-solver and the reason maintenance system is reduced since the dependency computation takes place without any extra costs during the inference process. As pointed out earlier, we may define Δ as $\mathcal{P}(\mathcal{A})$. Then an appropriate lattice function computing the *minimal label* from the sign of the body literals may be written in the heads of signed clauses. In the next example we show how the fixpoint operator W_P computes the label of ground atoms. In this example, the function $f_n : \mathcal{P}(\mathcal{A})^n \rightarrow \mathcal{P}(\mathcal{A})$ is defined as

$$f_n(E_1, \dots, E_n) = \bigcup_{L \in (E_1 \times \dots \times E_n)} \bigcup_{i=1}^n L \downarrow i$$

where $E_i \in \mathcal{P}(\mathcal{A})$ for each $1 \leq i \leq n$, and $L \downarrow i$ denotes the i -th component of L .

Example 32 Let us consider a MVL Λ over $\Delta = \mathcal{P}(\{A, B, C, D, E\})$ and the following SFLP.

$$\begin{aligned} f_2(V, W) : p &\leftarrow V : q, W : r \\ \{\{A, B\}, \{B, C, D\}\} : q &\leftarrow \\ \{\{A, C\}, \{D, E\}\} : r &\leftarrow \end{aligned}$$

Then, the label of p is computed as follows. The cartesian product $V \times W$ is the following set.

$$\begin{aligned} V \times W = & \{(\{A, B\}, \{A, C\}), (\{A, B\}, \{D, E\}), \\ & (\{B, C, D\}, \{A, C\}), (\{B, C, D\}, \{D, E\})\} \end{aligned}$$

Then the collection of $l_1 \cup l_2$ for each pair (l_1, l_2) in $V \times W$ is the set

$$\{\{A, B, C\}, \{A, B, D, E\}, \{A, B, C, D\}, \{B, C, D, E\}\}$$

This set is the result of $f_2(V, W)$. It is also the truth value assigned to p by $W_P^{\uparrow\omega}$.

¹¹For historical reasons the term ATMS (assumption based truth maintenance) is sometimes used in this paper.

5 Related Work

Ideas described in this paper evolved from a number of recent work. Signed resolution was developed by Baaz and Fermüller [BF92] for signed formulas whose signs were restricted to singleton sets. A more general version of signed resolution was studied independently by Hähnle [Häh93], and Murray and Rosenthal [MR94]. Each of these developments was set in the context of theorem proving. The application of CLP to SFLP was based in part on the work of Frühwirth [Fr94]. His method generated CLP-queries directly from the original program; the program is not first transformed into a CLP. In addition, only applications to annotated logic programming was considered. In order to characterize different reason maintenance systems semantically, a similar line of research has been pursued by Fehrer [Feh93]. His work focuses on Gabbay's labeled deductive system [Gab89] which is a much broader framework than signed formulas that can be used for general theorem proving in different kinds of logics.

The basic idea of transforming a multiple-valued logic program into a constraint logic program was implemented in [Deb94]. Some benchmarking results can be found there. However, the work again applies only to annotated logic programming, which is a restricted form of SFLP.

6 Conclusion

The aim of the current paper is to examine the operational semantics of SFLP more closely based on the declarative semantics set forth in [Lu96]. Section 2 summarizes the basics of SFLP. Section 3 presents the main result which shows that any SFLP P may be suitably translated to CLP. Consequently, SFLP may be computed through CLP-deduction. This enables the direct application of various CLP implementation techniques. Moreover, the translation sheds insights on the computational behavior of signed resolution. In Section 4, two applications of SFLP are considered. The first shows the implementation of bilattice logic programming using SFLP, and the second examines the connection of SFLP to ATMS.

Future research will extend existing results in several directions, including:

1. the clarification of the relationship between the semantics of base-level multiple-valued logic programs and their corresponding meta-level signed formula logic programs. Currently, the use of EMod as the declarative semantics of SFLP represents only an "approximation" to the meaning of underlying multiple-valued logic program, given by Mod. A more precise understanding of this approximation will be useful in understanding the information conveyed by computed answer.
2. the further development of partial solvability as a means for restricting the search space of inference procedures.
3. the extension of signed logic programming with nonmonotonic operators through a combination of tabling techniques [CW93] and constraint solving.

Nonmonotonicity makes possible the expression of meta-level information. The addition of nonmonotonicity to signed logic programs will enhance the possible implementation of hybrid knowledge bases [LNS96] through signed logic programs.

Acknowledgement: Reiner Hähnle's provided numerous useful comments on a preliminary draft of this paper. The reviewers detailed suggestions greatly improved the presentation of the material.

References

- [AW92] A. Aiken and E. Wimmers. Solving systems of set constraints. In *Proceedings of the 7th Symposium Logic in Computer Science*, pages 329–340. Computer Society Press, 1992.
- [BF92] M. Baaz and C. G. Fermüller. Resolution for many-valued logics. In A. Voronkov, editor, *Proceedings of Conference Logic Programming and Automated Reasoning*, pages 107–118. Springer-Verlag, 1992.
- [BS89] H.A. Blair and V.S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.
- [CW93] W. Chen and D.S. Warren. Query Evaluation Under the Well-Founded Semantics. In *Proceedings of the 12th ACM Symposium on Principles of Database Systems*, pages 168–179, ACM Press, 1993.
- [COL82] A. Colmerauer. Prolog and Infinite Trees. In K.L. Clark and S.A. Tarnlund editors, *Logic Programming*, pages 231–251, New York, 1982. Academic Press.
- [Deb94] D. Debertin. Parallel inference algorithms for distributed knowledge bases (in german). Master's thesis, Institute for Algorithms and Cognitive Systems, University of Karlsruhe, 1994.
- [DeK86] J. DeKleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.
- [DP90] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
- [Feh93] D. Fehrer. A Unifying Framework for Reason Maintenance. In Michael Clarke, Rudolf Kruse, and Serafin Moral, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty, Proceedings of EC-SQARU '93, Granada, Spain, Nov. 1993*, volume 747 of *Lecture Notes in Computer Science*, pages 113–120, Berlin, Heidelberg, 1993. Springer.
- [Fit91] M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.

- [Fr94] T. Frühwirth. Annotated constraint logic programming applied to temporal reasoning. In M. Hermengildo and P. Penjam, editors, *Proceedings of the Symposium on Programming Language Implementation and Logic Programming*, volume 844 of *Lecture Notes in Computer Science*, pages 230–243, Berlin, Heidelberg, 1994. Springer.
- [Gab89] Dov M. Gabbay. *Labelled Deductive Systems: Volume 1*. Oxford University Press, 1996.
- [Gin88] M.L. Ginsberg. Multivalued logics: A uniform approach to inference in artificial intelligence. *Computational Intelligence*, 4(3):265–316, 1988.
- [Häh91] R. Hähnle. Uniform notation of tableau rules for multiple-valued logics. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 26–29. Computer Society Press, 1991.
- [Häh93] R. Hähnle. Short normal forms for arbitrary finitely-valued logics. In *Proceedings of International Symposium on Methodologies for Intelligent Systems*, pages 49–58. Springer-Verlag, 1993.
- [Häh94a] R. Hähnle. *Automated Deduction in Multiple-Valued Logics*. International Series of Monographs on Computer Science, vol. 10. Oxford University Press, Oxford, UK. 1994.
- [Häh94b] R. Hähnle. Short conjunctive normal forms in finitely-valued logics. *Journal of Logic and Computation*, 4(6):905–927, 1994.
- [JL87] J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119. ACM Press, 1987.
- [KL92] M. Kifer and E. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 9:179–215, 1992.
- [KS92] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
- [LL94] S. Leach and J.J. Lu. Computing annotated logic programs. In P. van Hentenryck, editor, *Proceedings of the International Conference on Logic Programming*, pages 257–271. MIT Press, 1994.
- [Llo88] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2 edition, 1988.
- [LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.
- [Lu96] J.J. Lu. Logic Programming with Signs and Annotations. *Journal of Logic and Computation*, 6(6):755–778, 1996.

- [LMR97] J.J. Lu, N.V. Murray, and E. Rosenthal. A Framework for Reasoning in Multiple-Valued Logics. *Journal of Automated Reasoning*. To appear.
- [LNS96] J.J. Lu, A. Nerode, and V.S. Subrahmanian. Hybrid Knowledge Bases. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):773–785, 1996.
- [LR95] J.J. Lu and E. Rosenthal. Annotations, signs, and generally paraconsistent logics. *Intelligent Systems*, pages 143–157, 1995.
- [Mes96] B. Messing. Aggregation of Vague and Uncertain Knowledge with Many-valued Logics. *Data and Knowledge Engineering Journal*. To appear.
- [MR94] N.V. Murray and E. Rosenthal. Adapting classical inference techniques to multiple-valued logics using signed formulas. *Fundamenta Informaticae*, 21:237–253, 1994.
- [MS88] J.P. Martins and S.C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35:25–79, 1988.
- [Sar91] V. Saraswat. *Concurrent Constraint Programming*. PhD thesis, Carnegie-Mellon, 1991.
- [TAR55] A. Tarski. A Lattice-theoretical Fixpoint Theorem and its Applications. *Pacific J. Math.*, 5:285–309, 1955.
- [WTL93] T.J. Weigert and J-P. Tsai and X. Liu. Fuzzy operator logic and fuzzy reasoning. *Journal of Automated Reasoning*, 10:59–78, 1993.