October 12, 1995, 1–17 ()
Bucknell University Technical Report.

Signed Formula Logic Programming: Operational Semantics and Applications [*]

JACQUES CALMET calmet@ira.uka.de

Department of Computer Science, Institute for Algorithms and Cognitive Systems, University of Karlsruhe,76128 Karlsruhe, Am Fasanengarten 5

JAMES J. LU[**] jameslu@bucknell.edu

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

MARIA RODRIGUEZ[**]

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

JOACHIM SCHÜ schue@ira.uka.de

Department of Computer Science, Institute for Algorithms and Cognitive Systems, University of Karlsruhe,76128 Karlsruhe, Am Fasanengarten 5

Abstract. Signed formula can be used to reason about a wide variety of multiple-valued logics. The formal theoretical foundation of logic programming based on signed formulas is developed in [26]. In this paper, the operational semantics of signed formula logic programming is investigated through constraint logic programming. Applications to bilattice logic programming and truth-maintenance are considered.

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

[*] Please address all correspondence to:

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

E-mail: `jameslu@bucknell.edu`
Phone: +1 717 524 1394
Fax: +1 717 524 1822

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

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 [13], [14], [23], [24], [27], [28]. On the other hand, logic programming based on signed formulas — signed formula logic programming — is recently formalized in [26]. 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 [4], [19] are established. It is shown that signed formula logic programming and annotated logic programming together provide a basis for reasoning with "inconsistent" multiple-valued logic programs.

This paper extends the work in [26] 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 stand point, 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 [26] for processing queries with respect to signed formula logic programs. Secondly, in this paper we analyze two independent applications of signed formula logic programming: bilattice logic programming [9] and assumption based truth-maintenance [6]. 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 summerizes the theoretical foundation of signed formula logic programming. Section 3 describes 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).

2. Signed Formula Logic Program

2.1. Signed Formula

The basic building blocks of signed formulas are a multiple-valued logic Λ and its associated 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* — may be constructed recursively using signed atoms and classical connectives by: $\neg\mathcal{F}$, $\mathcal{F}_1 \mid \mathcal{F}_2$, $\mathcal{F}_1$ & $\mathcal{F}_2$, $\mathcal{F}_1 \leftarrow \mathcal{F}_2$, where $\mathcal{F}, \mathcal{F}_1, \mathcal{F}_2$ 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 Program

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

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

where $S_0 : A$ is a Λ-atomic signed atom, and each $S_1 : \mathcal{F}_1, ..., S_n : \mathcal{F}_n$ 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 assume 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.

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

PROPOSITION 1 *Mod extends to arbitrary signed formulas as follows.*

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

The classical notion of logical consequence applies. It is written with the symbol $\models$. The collection of all models of an SFLP P is denoted 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]

$$\{1\} : A \leftarrow$$
$$\{0\} : A \leftarrow$$

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 [25].

2.3. Semantics

An important property of classical logic programming is that a program P possesses an 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:

$$I_1(A) = 1$$
$$I_2(A) = 0$$

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 need not 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 in general 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 [22]. Fortunately we may obtain a good approximation to the models Mod(P) of an SFLP 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. **(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 $\mathcal{F}_1, ..., \mathcal{F}_n$ are variable free Λ-formulas. Then

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

Definition. **(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 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 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 1 *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 2 *Suppose $I_1 \sqsubseteq I_2$. Then for any signed atom $S : \mathcal{F}$. $I_1 \in$ EMod($S : \mathcal{F}$) implies $I_2 \in$ EMod($S : \mathcal{F}$).*

Various standard results of classical logic programming can now be proven for signed formula logic programs, including the existence of an 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 1 and 2. The interesting non-standard result is Theorem 3, where the connection between models of P (i.e. Mod(P)), and the e-models of P (i.e. EMod(P)) is established.

THEOREM 1 *Suppose P is an SFLP. Then there is an unique minimal e-model $\mathcal{E}_P$ of P under the ordering $\sqsubseteq$, given by*

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

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:

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

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

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

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

THEOREM 2 $W_P^{\Uparrow \omega} = \mathcal{E}_P$.

THEOREM 3 $\{I(A) \mid I \in Mod(P)\}$ *is an e-model of P.*

COROLLARY 1 $\mathcal{E}_P(A) \sqsubseteq \{I(A) \mid I \in Mod(P)\}$.

In general, an SFLP P may be translated into an equivalent Λ-atomic SFLP.

THEOREM 4 *Suppose P is an SFLP. Then there is a Λ-atomic SFLP P' such that $Mod(P) = Mod(P')$.*

Typically, the Λ-atomic SFLP P' will contain many more clauses than P. However, queries with respect to Λ-atomic SFLPs are much easier to process with as we may adapt query answering procedures for classical logic programming in a relatively straightforward manner.

Example: Let's 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.

$\{0.5\} : A \leftarrow \{0.5\} : B \ \& \ \{0.5\} : C$
$\{0.5\} : A \leftarrow \{0.5\} : B \ \& \ \{0.8\} : C$
$\{0.5\} : A \leftarrow \{0.5\} : B \ \& \ \{1\} : C$
$\{0.5\} : A \leftarrow \{1\} : B \ \& \ \{0.5\} : C$
$\{0.5\} : A \leftarrow \{0.8\} : B \ \& \ \{0.5\} : C$ □

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 $S_1 : A \leftarrow \mathit{Body}$. 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, 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.***(Signed resolution)** Let C be the signed clause $S_0 : A_0 \leftarrow S_1 : A_1 \mathbin{\&} \ldots \mathbin{\&} S_n : A_n$ and Q be the signed query $\leftarrow D_1 \mathbin{\&} S : A \mathbin{\&} D_2$ where D_1, D_2 are conjunctions of signed atoms. Suppose A_0 and A are unifiable via mgu θ. Then the query

$$\leftarrow (D_1 \mathbin{\&} \nu(S, S_0) : A \mathbin{\&} S_1 : B_1 \mathbin{\&} \ldots \mathbin{\&} S_n : B_n \mathbin{\&} D_2)\theta$$

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 $\nu(S, S_0) = \Delta$.

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

This question may be answered by the following signed deduction.

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

$\square$

LEMMA 3 $\nu(T_1, T_2) = \Delta$ *iff* $T_2 \subseteq T_1$.

THEOREM 5 *Signed resolution is sound and complete for Λ-atomic formulas with respect to $\mathcal{E}_P$.*

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

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

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

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

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. [2]), to SFLPs.

The work on constraint logic programming was pioneered by Jaffar and Lassez [16]. 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

$$A \leftarrow \Xi \mid\mid B_1 \ \& \ ... \ \& \ B_n$$

where Ξ is a constraint, and $A, B_1, ..., B_n$ 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.

An 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 $A \leftarrow \Xi || B_1 \ \& \ ... \ \& \ B_n$ in P, if θ is a Σ-valuation that is a solution of Ξ and $B_i\theta \in I$ for $i = 1, ..., n$, then $A\theta \in I$. As Jaffar and Lassez showed [16], a CLP program is assured of a least Σ-model, which 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 that follows extends 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. **(Constraint Form)** Given a Λ-atomic SFLP P, the *constraint form* of P is the CLP, denoted $CF(P)$, made up of the following two collections of non-ground CLP clauses.

1. $A(V) \leftarrow S \subseteq V \mid\mid B_1(S_1) \ \& \ ... \ \& \ B_n(S_n)$

   where the signed clause $S : A \leftarrow S_1 : B_1 \ \& \ ... \ \& \ S_n : B_n$ is in P.

2. $A(V) \leftarrow (V_1 \cap V_2) \subseteq V \mid\mid A(V_1) \ \& \ A(V_2)$

   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 $Q = \leftarrow S_1 : B_1 \ \& \ ... \ \& \ S_n : B_n$ is obtained as a special case of the first step above. That is, $CF(Q) = \leftarrow B_1(S_1) \ \& \ ... \ \& \ B_n(S_n)$.

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

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

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

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

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

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

Before demonstrating that ψ is "meaning preserving", we prove a useful property of the CLP models of $CF(P)$. First, a model I is said to be *supported* if for each atom $A \in I$, there is a ground instance of a clause $A \leftarrow \Xi \| B_1 \ \& \ ... \ \& \ B_n$ such that each B_i is in I, and that the constraint Ξ is true. This is a simple generalization of the notion of supportedness for classical logic programming [1].

LEMMA 4 *Suppose I is a supported CLP model of $CF(P)$ and $A(S)$ is a ground atom in I. Then for any $T \subseteq \Delta$ such that $S \subseteq T$, $A(T) \in I$.*

Proof: I is supported implies that there is a clause C in $CF(P)$

$$A(V) \leftarrow S_0 \subseteq V \parallel B_1(S_1) \ \& \ ... \ \& \ B_n(S_n)$$

and a substitution θ such that $A(V)\theta = A(S)$, $B_i(S_i)\theta \in I$, and θ is a solution to $S_0 \subseteq V$. Clearly, any substitution γ that is identical to θ for any variable other than V, and that $V\theta \subseteq V\gamma$, is a solution to $S_0 \subseteq V$. Moreover, $B_i(S_i)\gamma \in I$ since $B_i(S_i)\gamma = B_i(S_i)\theta$ (note that V does not occur in any $B_i(S_i)$ by the construction of $CF(P)$). It follows that as I is a model of C, $A(V)\gamma \in I$. ■

THEOREM 6 *I is an e-model of P iff $\psi(I)$ is a CLP model of $CF(P)$.*

Proof:

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

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

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

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

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

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

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 where C has the form

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

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

only if: Suppose

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

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

$$A(V_0) \leftarrow S_0 \subseteq V_0 \ || \ B_1(S_1) \ \& \ ... \ \& \ B_n(S_n)$$

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 $I(A) \subseteq S_0$.

■

COROLLARY 2 *$\psi(\mathcal{E}_P)$ 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

$$\leftarrow \Xi \ || \ A_1 \ \& \ ... \ \& \ A_n$$

where Ξ is a constraint, and $A_1, ..., A_n$ 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, consider the following example.

Example: 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.

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

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

□

THEOREM 7 *(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 $CF(Q)$.*

3.3. Implementation Issues

Two simple prototype interpreters have been implemented in the language C for experimentation[30]. One is based on signed resolution, and the other is based on CLP. We are especially interested in comparing the structure of the search space induced by each of the query processing methods.

Annotated logic, as studied in [4], [17], [18], [19], has been shown to relate to signed formulas [23]. In [20], 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 [21] no longer holds. Hence a function that fairly chooses signed atoms from queries appearing in a deduction has to be provided. Viewing an SFLP as a CLP further sheds light on this issue.

Let's reconsider 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.

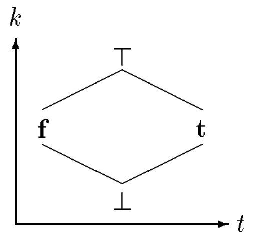

Figure 1. The Bilattice FOUR.

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] This observation can, in fact, be generalized.

4. Applications

4.1. Bilattice Logic Programming

This section applies SFLP to analyzing finitely valued bilattice logic programs [9]. 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 a atom A, can A evaluate to some value in S, under the intended meaning $[\![P]\!]$ of P?

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

$$SFB(P) \models S : \mathcal{F} \text{ iff } [\![P]\!](\mathcal{F}) \in S.$$

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: $\bot$, $\top$, $\mathbf{f}$, and $\mathbf{t}$, which denote respectively the least and the greatest elements with respect to $\preceq_k$, and $\preceq_t$. FOUR shown in Figure 1 is thus 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 $\preceq_t$ [9].

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 $\mathbf{t}$, and $false$ is a constant that evaluates to $\mathbf{f}$ 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 [9], 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.

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

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 the Knaster-Tarski theorem on monotone operators over lattices. We denote $lfp_t(\Phi_P)$ the least fixed point of Φ_P under the $\preceq_t$ ordering.

Example: Consider the bilattice logic program P over FOUR

$r \leftarrow p \otimes (q \vee t)$
$s \leftarrow t \oplus p$
$u \leftarrow p \otimes t$
$p \leftarrow true$
$q \leftarrow true$

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

The fixed point $lfp_t(\Phi_P)$ establishes $[\![P]\!]$. 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 $\uparrow_t \mu = \{\beta \in \Delta | \mu \preceq_t \beta\}$.

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

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

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

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

$\uparrow_t V : r \leftarrow V : (p \otimes (q \vee t))$
$\uparrow_t V : s \leftarrow V : (t \oplus p)$
$\uparrow_t V : u \leftarrow V : (p \otimes t)$
$\{\mathbf{t}\} : p \leftarrow$
$\{\mathbf{t}\} : q \leftarrow$
$\{\mathbf{t}, \mathbf{f}, \bot, \top\} : \mathit{false} \leftarrow$
$\{\mathbf{t}\} : \mathit{true} \leftarrow$

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

$true$	$false$	u	t	s	r	q	p
$\{\mathbf{t}\}$	$\{\mathbf{t}, \mathbf{f}, \bot, \top\}$	$\{\bot, \mathbf{t}\}$	$\{\mathbf{t}, \mathbf{f}, \bot, \top\}$	$\{\top, \mathbf{t}\}$	$\{\mathbf{t}\}$	$\{\mathbf{t}\}$	$\{\mathbf{t}\}$

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 bilattice logic program, as the next theorem indicates.
□

THEOREM 8 *Suppose P is a positive bilattice logic program. Then $lfp_t(\Phi_P)(A) = \mu$ iff $lfp(W_{SFB(P)})$ e-satisfies the signed atom $\uparrow_t \mu : A$.*

$SFB(P)$ can now be used to answer questions about P under the meaning $[\![P]\!]$. 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 [6]. 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 [12], 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 [31] 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 maintenance[9] 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 [29], 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 following example we show how the fixpoint operator W_P computes the label of ground atoms. We define a function f_n as follows: $f_n : \mathcal{P}(\mathcal{A})^n \mapsto \mathcal{P}(\mathcal{A})$ is defined as

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

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: 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 $W_P^{\Uparrow\omega}$.
□

5. Related Work

Ideas described in this paper evolved from a number of recent work. Signed resolution was developed by Baaz and Fermüller [3] for signed formulas whose signs were restricted to singleton sets. A more general version of signed resolution was studied independently by Hähnle [15], and Murray and Rosenthal [27]. 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 [10]. 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 [8] who elaborated independently on a closely related idea. His work focuses on Gabbay's labeled deductive system [11] which is a much more general framework than signed formulas which 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 proposed and implemented in [5]. Some benchmarking results can also be found there. However, the work again applies only to annotated logic programming, which is a restricted form of SFLP.

Acknowledgements

Reiner Hähnle's provided numerous useful comments on a preliminary draft of this paper. The latex style file was created by Amy Hendrickson.

Notes

1. To 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.
2. Abstractly, formulas in Λ are constructed from atomic formulas and connectives of various arity. Suppose Θ is an n-ary connective, and $\mathcal{F}_1, ..., \mathcal{F}_n$ are Λ-formulas. Then the the expression $\Theta(\mathcal{F}_1, ..., \mathcal{F}_n)$ is also a Λ-formula.
3. We use $|$ 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.
4. Again for emphasis, I is a witness to the question "Can $\mathcal{F}$ evaluate to a value in S?"
5. The truth value set $\{0, 0.2, 0.5, 0.8, 1\}$ has been applied in fuzzy reasoning [32].
6. We use $\Uparrow$, 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 [7].
7. This is the usual interpretation associated with conjunction in fuzzy logics.
8. This amounts to, also, the strategy adopted by Frühwirth in his implementation of annotated logic programming [10].
9. For historical reasons the term ATMS (assumption based truth maintenance) is sometimes used in this paper.

References

1. K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages

89–142. Morgan Kaufmann, 1987.
2. 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.
3. 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.
4. H.A. Blair and V.S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.
5. D. Debertin. Parallel inference algorithms for distributed knowledge bases (in german). Master's thesis, Institute for Algorithms and Cognitve Systems, University of Karlsruhe, 1994.
6. J. DeKleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.
7. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
8. D. Fehrer. A Unifying Framework for Reason Maintenance. In Michael Clarke, Rudolf Kruse, and Serafín Moral, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertaint y, Proceedings of ECSQARU '93, Granada, Spain, Nov. 1993*, volume 747 of *Lecture Notes in Computer Science*, pages 113–120, Berlin, Heidelberg, 1993. Springer.
9. M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
10. T. Frühwirth. Annotated constraint logic programming applied to temporal reasoning. In *Proceedings of the Symposium on Programming Language Implementation and Logic Programming*, pages 230–243. Springer-Verlag, 1994.
11. Dov M. Gabbay. LDS- labelled deductive systems. Preprint, Dept. of Computing, Imperial College, London, September 1989.
12. M.L. Ginsberg. Multivalued logics: A uniform approach to inference in artificial intelligence. *Computational Intelligence*, 4(3):265–316, 1988.
13. 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.
14. R. Hähnle. *Automated Theorem Proving in Multiple-Valued Logics*. Oxford University Press, 1993.
15. 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.
16. 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.
17. M. Kifer and E. Lozinskii. Ri: A logic for reasoning with inconsistency. In *IEEE Symposium on Logic in Computer Science*, pages 253–262, 1989.
18. M. Kifer and E. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 9:179–215, 1992.
19. M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
20. S. Leach and J.J. Lu. Computing annotated logic programs. In *Proceedings of the International Conference on Logic Programming*, pages 257–271. MIT Press, 1994.
21. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2 edition, 1988.
22. J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.
23. J.J. Lu, N.V. Murray, and E. Rosenthal. Signed formulas and annotated logics. In *Proceedings of the 23rd International Symposium on Multiple-Valued Logics*, pages 48–53. Computer Society Press, 1993.
24. J.J. Lu, N.V. Murray, and E. Rosenthal. Signed formulas and fuzzy operator logics. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*. Springer-Verlag, 1994.
25. J.J. Lu and E. Rosenthal. Annotations, signs, and generally paraconsistent logics. *Intelligent Systems*, pages 143–157, 1995.
26. J. Lu. Logic programming with signs and annotations. To appear in Journal of Logic and Computation, 1995.
27. N.V. Murray and E. Rosenthal. Signed formulas: A liftable meta-logic for multiple-valued logics. In *Proceedings of International Symposium on Methodologies for Intelligent Systems*, pages 275–284. Springer-Verlag, 1993.

28. N.V. Murray and E. Rosenthal. Signed formulas: A liftable meta-logic for multiple-valued logics. *Fundamenta Informatica*, 1995.
29. J.P. Martins and S.C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35:25–79, 1988.
30. M. Rodriguez. Solving set constraints in signed formulas and logic programming, May 1995. Honors Thesis for the degree of BS, Bucknell University.
31. V. Saraswat. *Concurrent Constraint Programming*. PhD thesis, Carnegie-Mellon, 1991.
32. T.J. Weigert, J-P. Tsai, and X. Liu. Fuzzy operator logic and fuzzy resolution. *Journal of Automated Reasoning*, 10:59–78, 1993.