PARACONSISTENT FOUNDATIONS FOR LOGIC PROGRAMMING *

by

Howard A. Blair, V. S. Subrahmanian

Abstract

Very large knowledge bases often contain inconsistent information. Paraconsistent logics provide a formal means for reasoning in inconsistent formal systems (cf. da Costa [CA74,77,81,87]). Hence, paraconsistent logics seem to provide the appropriate setting for reasoning about knowledge bases. In this paper, we present a class of logic programming languages - a program in one of these languages is a formula over a many-valued logic whose set of truth values forms a complete lattice. As clauses in such programs may contain negated atoms in the heads of clauses, programs may be inconsistent (in the intuition of two-valued logic). We give a paraconsistent model-theoretic semantics for programs written in these logics. We also characterize the models of such programs in terms of the pre-fixed-points of a monotone operator from interpretations to interpretations. As proof procedures must be computationally very efficient, we give an operational semantics based on AND/OR tree searching. We also give a faster proof procedure called SLDq-resolution for a subclass of programs. Soundness and completeness results are derived.

1 Introduction

1.1 Logic Programming

*This work was supported by U. S. Air Force Contract F30602-85-C-0008

The Journal of Non-Classical Logic
Volume 5, Number 2, November 1988
One of the main issues in the design of a programming language is that of developing a semantics for programs written in that language. However, in order to verify that a program indeed does what is intended by the programmer, one needs an abstract specification (usually in some other language) of the input-output relation computed by the program. Thus, program development may be seem to consist of three parts:

1. writing the abstract specification (in language $L_1$)
2. writing the program (in language $L_2$)
3. verifying that the program obtained in (2) satisfies the conditions in (1) above.

In the late sixties, R. Kowalski and A. Colmerauer realized that classical first order logic itself could be used as a programming language. In that case, the languages $L_1$ and $L_2$ are the same, the abstract specification of the program is exactly the program itself and the verification step (3) is unnecessary. Thus, program development is a substantially simpler process. This proposal was called logic programming. Since then logic programming has made great advances, and is currently the programming formalism being used in the Japanese Fifth Generation Computer project.

Perhaps in order to avoid computational problems, Kowalski restricted programs in his language to be finite sets of definite clauses, i.e. sentences of the form

$$B_1 \& \ldots \& B_n \Rightarrow A$$

where $A$ and $B_i$'s are the atomic formulas (all free variables in the above formula are implicitly universally quantified). It has been shown that any r.e. set [B82,SS82] can be represented as the set of logical consequences of a finite set of definite clauses - hence, this seems a reasonable restriction. In addition, definite clauses admit an efficient proof procedure. Queries to such a program are defined to be conjunctions of atoms where all occurrences of free variables are existentially quantified. Since Kowalski and Colmerauer's original proposal, several mechanisms have been studied for extending logic programming with negation, i.e. when a program consists of a
finite set of clauses of the form 

\[ L_0 \& \ldots \& L_n \Rightarrow A \]

where each \( L_i \) is a literal (i.e. atom or negated atom) and \( A \) is an atom. Programs consisting of a finite set of definite clauses have the property that their (Herbrand) models are the fixed points \(^1\) of a certain monotone operator from Herbrand interpretations to Herbrand interpretations.

Blair and Subrahmanian [BS87] were the first to develop a semantics for logic programming with clauses of the form 

\[ L_1 \& \ldots \& L_n \Rightarrow L_{n+1} \]

where each \( L_i \) is a literal - a very similar proposal was made in the case of many-valued logic programs in [S87].

1.2 Motivation

One of the areas being intensely studied in the foundations of logic programming is the possibility of many-valued logic programming. In earlier papers [S87, BS87a], we introduced the notion of an annotated literal. We argue in [BS87b] that logic programs may be taken as representing belief rather than truth. For example, a programmer writes certain clauses because he believes them to be true, not because they are indeed true. Thus, when he writes

\[ L_0 \Leftarrow L_1 \& \ldots \& L_n \]

(each \( L_i \) a literal), what is really intended is the statement:

If it is known that \( L_1 \) is true and ... and it is known that \( L_n \) is true, then it is known that \( L_0 \) is true.

---

\(^1\) If \( L \) is a partially ordered set and \( f : L \to L \) is a function, then \( z \in L \) is a (pre-fixed point, resp. fixed point) iff \( f(z) \leq z \), resp. \( f(z) = z \) where \( L \) is partially ordered by \( \leq \).
In this note, it is not our point to argue the philosophical implications of this. That is done in [BS87a,b]. It is also shown there that logic programs as defined in Lloyd [LL84] reduce to a special case of the above intuition. What we would indeed like to present in this paper is a very general framework for reasoning in arbitrary many-valued logics that satisfy certain conditions. These logics may be finite-valued or countably infinite-valued or uncountably infinite-valued. We present a declarative model-theoretic semantics for arbitrary sets of annotated formulas, relate models to fixed-points of operators associated with annotated logic programs and establish the connection between the class of logics introduced in this paper and the notion of paraconsistency [cf. da Costa CA74,77,81,87].

1.3 Syntax

Suppose \( I \) is a non-empty set of truth values. Constant symbols and variable symbols are terms. If \( f \) is an \( n \)-ary function symbol, and \( t_1, \ldots, t_n \) are terms, then \( f(t_1, \ldots, t_n) \) is a term. If \( p \) is an \( n \)-ary predicate symbol, and \( t_1, \ldots, t_n \) are terms, then \( p(t_1, \ldots, t_n) \) is an atom. If \( A \) is an atom, then \( A \) and \( \neg A \) are literals.

(1.1) DEFINITION. If \( A \) is an atom (literal) and \( \mu \in I \), then \( A : \mu \) is an annotated atom (literal) over \( I \).

If \( e_1, e_2 \) are syntactic first order expressions (terms or atoms), then a substitution \( \Theta \) of variable symbols for terms is called a unifier of \( e_1, e_2 \) iff the application of \( \Theta \) to \( e_1 \), (denoted by \( e_1 \Theta \)) yields the same expression as \( e_2 \Theta \). A most general unifier (mgu, for short) of any two syntactic expressions \( e_1, e_2 \) is a unifier \( \Theta \) such that for any unifier \( \theta \) of the expressions \( e_1, e_2 \), there is a substitution \( \gamma \) such that \( \Theta \gamma = \theta \). If \( e_1, e_2 \) are unifiable terms or atoms, then they possess a most general unifier (cf. [LL84]).

The intuitive reading of the atom \( A: \mu \) is it is believed that \( A \)'s truth value is at least \( \mu \). For example, the intuitive reading of \( A: \text{true} \) is it is believed that \( A \)'s truth value is at least true.
(1.2) DEFINITION. If \( L_0, \ldots, L_n \) are annotated literals over \( I \), then

\[
L_0 \leftarrow L_1 \& \ldots \& L_n
\]

is an annotated clause over \( I \). \( L_0 \) is called the head of the above annotated clause, while \( L_1 \& \ldots \& L_n \) is called the body. (We will often abuse terminology and refer to annotated clauses as just clauses).

(1.3) DEFINITION. An annotated logic program (ALP) over \( I \) is a finite set of annotated clauses over \( I \).

Thus, for example, the quantitative logic programs of [S87] are ALPs over \([0,1] \cup \{ \top \}\), while the evidential logic programs of [S87a] are ALPs over \([0,1] \times [0,1] \). General logic programs may be considered to be ALPs over \{true, false\}.

2 Semantics of ALPs

We assume that all interpretations have as their domain of interpretation the Herbrand base \( B_P \) (the set of all variable free atomic formulas of the language of \( P \)) of the ALP \( P \) under consideration. But first we explain the notion of satisfaction. We assume that \( I \) is a complete lattice under an (as yet unspecified) ordering \( \leq \). (In the case of general logic programs, \( \leq \) may be taken to be the ordering

\[
\text{true}
\]

\[
\text{false}
\]

while in the case of quantitative logic programs, it may be taken to be the \( \ll \) ordering of [S87]). Thus, an interpretation \( I \) of an ALP \( P \) over \( I \) may be considered to be a mapping \( I : B_P \rightarrow I \). The \( \leq \) ordering is extended to interpretation in the natural way, i.e.

\[
I_1 \leq I_2 \iff (\forall A \in B_P) I_1(A) \leq I_2(A)
\]

The orderings \( \geq, \succ, \prec \) are defined in the usual way. We also assume the existence of a function \( \rightarrow : I \rightarrow I \).
(2.1) DEFINITION. A formula is said to be closed iff it contains no free occurrences of a variable.

(2.2) DEFINITION. [Satisfaction] An interpretation \( I \) is said to satisfy

1. the formula \( F \) iff it satisfies every closed instance of \( F \).
2. the variable free annotated atom \( A : \mu \) iff \( I(A) \geq \mu \).
3. the variable free annotated literal \( \neg A : \mu \) iff \( I(A) \geq \neg(\mu) \) (iff \( I \models A : \neg \mu \)).
4. the variable free formula \( F_1 \& F_2 \) iff \( I \) satisfies \( F_1 \) and \( F_2 \).
5. the variable free formula \( F_1 \lor F_2 \) iff \( I \) satisfies \( F_1 \) or \( F_2 \).
6. the variable free formula \( F_1 \iff F_2 \) iff either \( I \) satisfies \( F_1 \) or \( I \) does not satisfy \( F_2 \).
7. the variable free formula \( F_1 \iff F_2 \) iff \( I \) satisfies \( F_1 \iff F_2 \) and \( I \) satisfies \( F_2 \iff F_1 \).
8. the closed formula \( (\exists x)F \) iff there is some variable free term \( t \) such that \( I \) satisfies \( F[t/x] \) where \( F[t/x] \) denotes the result of replacing all free occurrences of \( x \) in \( F \) by \( t \).
9. the closed formula \( (\forall x)F \) iff for every variable free term \( t \), \( I \) satisfies \( F[t/x] \).

Satisfaction is denoted by the symbol \( \models \). (We also use the symbol \( \models \) to denote logical consequence. The intended meaning of \( \models \) is usually evident from the context in which it is used). If \( F \) is a formula and then we use the notation \( (\exists)F \) and \( (\forall)F \) to denote, respectively, the existential and universal closure of \( F \).

(2.3) LEMMA. If \( I \) is an interpretation, then \( I \models (\exists)\neg A : \mu \) iff \( I \models (\exists)A : \neg(\mu) \).

PROOF: \( I \models (\exists)\neg A : \mu \) iff \( I \models \neg A(t^-/x^-) : \mu \) (where \( t^- \) is a vector of variable free terms and \( x^- \) is a vector of all variable symbols occurring in \( A \)).

iff \( I \models A(t^-/x^-) : \neg(\mu) \) (by definition).

iff \( I \models (\exists)A : \neg \mu \). \( \Box \)

The following theorem follows immediately from the above lemma.
(2.4) THEOREM. Suppose \( P \) is an ALP over \( I \). Let \( P' \) be the ALP obtained from \( P \) by replacing all annotated literals of the form \(-A : \mu\) by \( A : -\langle \mu \rangle\). Then \( I \) is a model of \( P \) iff \( I \) is a model of \( P' \). \( \Box \)

Throughout the rest of this paper, we will assume, without loss of generality, that ALPs contain no negated literals. Associated with every ALP \( P \) over \( I \) is a function \( T_P \) from Herbrand interpretations to Herbrand interpretations such that \( T_P (I)(A) = \sqcup \{ \mu | A : \mu \Leftarrow B_1 \& \ldots \& B_n \text{ is a ground instance of an annotated clause in } P \text{ and } I \models B_1 \& \ldots \& B_n \} \). (We use the notation \( \sqcup \) and \( \sqcap \) to denote least upper bound and greatest lower bound, respectively. As \( I \) is a complete lattice under \( \leq \), the least upper bound always exists. This definition guarantees that \( T_P \) is always monotonic, i.e. if \( I_1 \leq I_2 \), then \( T_P (I_1) \leq T_P (I_2) \).

In conventional logic programming, one explicitly specifies the \( \leq \) ordering on truth values, and this (usually) guarantees that the variously defined \( T_P \) are monotonic (see, for e.g. [VE86, AVE82, VK76, ABW86, LM85, FI85]). However, we do not wish to commit ourselves to a particular ordering.

(2.5) THEOREM. Suppose \( A \) is an ALP over \( I \) (where \( I \) is a complete lattice under the \( \leq \) ordering) and \( T_P \) is a function from Herbrand interpretations to Herbrand interpretations. Then \( I \) is a model of \( P \) iff \( T_P (I) \preceq I \).

PROOF. (\( \Rightarrow \)) Suppose \( I \) is a model of \( P \). Let \( A \in B_P \). Let \( \Gamma \) be the set of all ground instances of annotated clauses \( C \) in \( P \) having \( A \) as the atom in their head and whose bodies are satisfied by \( I \). Let \( \Theta \) be the set of truth values occurring as annotations in the heads of clauses in \( \Gamma \). Then, by definition, \( T_P (I)(A) = \sqcup \Theta \). Since \( I \models P \), \( I(A) \geq \mu \) for each \( \mu \in \Theta \). Thus, \( T_P (I)(A) \leq I(A) \).

(\( \Leftarrow \)) Suppose \( I \) is a prefixed-point of \( T_P \), and suppose

\[
A_0 : \mu_0 \Leftarrow A_1 : \mu_1 \& \ldots \& A_k : \mu_k,
\]
is a ground instance of an annotated clause in $P$ whose body is satisfied by $I$. Then, by definition of $T_P$, we have $T_P(I)(A) \geq \mu_0$. As $I$ is a prefixed-point of $T_P$,

\[ I(A) \geq T_P(I)(A) \geq \mu_0 \]

Hence $I$ is a model of $P$. □

Note that monotonicity of $T_P$ is not needed for the above result to hold. Unfortunately, in the absence of monotonicity, $T_P$ need have no pre-fixed-points at all.

(2.6) THEOREM. $P$ has a least model that is identical to the least fixed-point of $T_P$.

PROOF. The theorem follows immediately from the fact that $T_P$ is a monotone function from a complete lattice to a complete lattice. □

As $I$ is, in all cases, a complete lattice, it has a least element which we shall denote by $\perp$. Then, associated with every $I$ there exists a specific interpretation $\triangle$ that assigns the truth value $\perp$ to every element $A \in B_P$.

(2.7) DEFINITION. If $P$ is an ALP over $I$, then the upward iteration of $T_P$ is defined as

\[ T_P \uparrow 0 = \triangle \]

\[ T_P \uparrow \lambda = \sqcup_{\alpha < \lambda} T_P(([T_P \uparrow \alpha]) \]

for all ordinals $\lambda$.

(2.8) THEOREM. $T_P \uparrow \omega$ is identical to the least fixed point of $T_P$.

PROOF. $[T_P \uparrow \omega \leq \text{Ifp}(T_P)]$ Direct consequence of the fact that $T_P$ is monotonic. $[\text{Ifp}(T_P) \leq T_P \uparrow \omega]$ First, observe that for any atom $A \in B_P$,

\[ (C) \ T_P \uparrow 0(A) \leq T_P \uparrow 1(A) \leq T_P \uparrow 2(A) \leq ... \]
Second, observe that each $T_P \uparrow n(A)$ is either the bottom element of $I$ or the least upper bound of some set of truth values annotating the heads of some clause in $P$. Thus, the ascending chain $(C)$ contains finitely many distinct truth values and therefore contains its own least upper bound. It follows that if

$$A_1 : \mu_1 \& \ldots \& A_n : \mu_n$$

is a conjunction of ground atoms such that

$$T_P \uparrow \omega \models A_1 : \mu_1 \& \ldots \& A_n : \mu_n$$

then there is some integer $m$ such that

$$T_P \uparrow m \models A_1 : \mu_1 \& \ldots \& A_n : \mu_n$$

From this, and the definition of $T_P$, it follows that

$$T_P (T_P \uparrow \omega) \preceq T_P \uparrow \omega$$

and hence $T_P \uparrow \omega$ is a fixed-point of $T_P$. □

**Note.** In contrast with an ordinary definite Horn clause program $P$ whose corresponding operator $T_P$ is continuous even when $P$ is infinite, the preceding theorem depends on $P$ having only finitely many clauses. Moreover, Theorem 2.8 suggests that $T_P$ might be continuous. Unfortunately, we have the following example.

(2.9) **EXAMPLE.** We present an example where $T_P$ is not continuous. Let $I$ be the set of truth values shown in the Hasse diagram below:
Let $P$ be the ALP containing the single annotated clause

$$q : \infty \Leftarrow p : e$$

and let $D$ be the directed set of interpretations $\{I_0, I_1, \ldots\}$ such that $I_n(p) = n$, and $I_n(q) = 0$. Then $T_P(I_n)(q) = \sqcup \forall = 0$ for all $n$. Hence,

$$\bigcup_n T_P(I_n)(q) = 0$$

But

$$\bigcup_n I_n(p) = \infty \geq e$$

Hence,

$$T_P\left(\bigcup_n I_n\right)(q) = \infty$$

Therefore, $T_P$ is not continuous. $\square$

(2.10) DEFINITION. A model $I$ of the ALP $P$ over $I$ is supported iff $I(A) = \sqcup\{\mu \mid A : \mu \Leftarrow B_1 : \mu_1 \& \ldots \& B_n : \mu_n\}$ is a ground instance of an annotated clause in $P$ and $I \models B_1 : \mu_1 \& \ldots \& B_n : \mu_n$.

(2.11) THEOREM. $I$ is a fixed-point of $T_P$ iff $I$ is a supported model of $P$. 
PROOF. Immediate from definition. □

Analogous to the interpretation $\Delta$ that assigns $\bot$ to every element of the Herbrand Base of $P$ is the interpretation $\nabla$ that assigns the $\top$ element of $I$ (which must exist as $I$ is a complete lattice) to every element of the Herbrand base of $P$.

(2.12) DEFINITION. The downward iteration of $T_P$ is defined as follows:

$$T_P \downarrow 0 = \nabla$$

$$T_P \downarrow \lambda = \cap_{\alpha < \lambda} (T_P \downarrow \alpha)$$

where $\lambda$ is any ordinal.

(2.13) DEFINITION. The ALP $P$ is canonical iff $T_P \downarrow \omega$ is a fixed-point of $T_P$.

(2.14) THEOREM. $P$ is canonical iff $T_P \downarrow \omega$ is the greatest fixed-point of $T_P$. □

(2.15) THEOREM. If $P$ is canonical, then $T_P \downarrow \omega$ is the greatest supported model of $P$. □

3 Operational Semantics

Once the model-theoretic semantics of a logic has been given, the next step is to develop a proof theory, and show that the proof procedure is sound and complete. Typically, one would give a tableau style proof procedure, where it is usually a straight-forward task to verify completeness. However, in computer science (and in logic programming), motivations are a bit different. One would like the proof procedure to be as efficient as possible. Tableau-style proof procedures are usually highly intractable (from a computational point of view). The prevailing view in the computer science community is to give a computationally fast proof method that is complete for a large a fragment of the logic as possible.
(3.1) DEFINITION. If $P$ is an ALP, and $A_1 : \mu_1 \& \ldots \& A_k : \mu_k$ a conjunction of annotated atoms, then the existential closure of this conjunction is called a query.

Intuitively, the above query asks the ALP $P$ if ($\exists$)($A_1 : \mu_1 \& \ldots \& A_k : \mu_k$) is a logical consequence of the ALP $P$. In future, we will assume that a query consists of just one annotated atom (there is no loss of generality in doing so). Given an ALP $P$, and a query $G$, we construct an AND/OR tree $\Upsilon(P, G)$ as follows:

(3.2) DEFINITION. Let $P$ be an ALP, and $A : \mu$ be a query. Then $\Upsilon(P, A : \mu)$ is an and/or tree with the following properties:

1. each node is called either an or-node or an and-node but not both

2. The root of $\Upsilon(P, A : \mu)$ is an or-node labelled $A : \mu$.

3. If $N$ is an or-node, then it is labelled by a single annotated atom.

4. Each and-node is labelled by a clause from $P$ and a substitution.

5. Descendants of an and-node are all or-nodes, and vice-versa.

6. If $N$ is an or-node labelled $A : \mu(\mu \neq \perp)$, and if $C\theta$ (cf. the discussion of substitution following Definition 1.1) is

$$A : \beta \leftarrow B_1 \& \ldots \& B_n$$

is an instance of a clause $C$ in $P$ ($\theta$ being the most general unifier of $A$ and the head of $C$), then there is a descendant (AND) node of $N$ labelled with $C$ and $\theta$. An or-node with no descendants is called an uninformative node.

7. If $N$ is an and-node labelled by the clause $C$ and the substitution $\theta$, then for every annotated literal $B : \gamma$ in the body of $C$, there is a descendent or-node labelled with $B\theta : \gamma$. An and-node with no descendants (i.e. the body of $C$ is empty) is called a success node.

We now define a partial function that, given any node $N$ in the tree $\Upsilon(P, A : \mu)$, associates a truth value with $N$.

---

$^2$For a sentence $F$ to be a logical consequence of $A$, we only require that $F$ be true in all Herbrand models of $A$. Note that this is weaker than the usual notion of logical consequence, where non Herbrand models are taken into consideration also.
(3.3) DEFINITION. We define a partial function $\nu$ that assigns truth values to the nodes of an and-or tree associated with a program and a query. $\nu$ assigns a truth value only to nodes which do not occur in infinite branches.

1. If $N$ is a success node labelled $B : \mu$, then $\nu(N) = \mu$.
2. If $N$ is an uninformative node, then $\nu(N) = \bot$.
3. If $N$ is an or-node not occurring in an infinite branch that is not uninformative, then $\nu(N) = \cup\{\nu(N_i) \mid N_i$ is an immediate descendant node of $N, 1 \leq i \leq k\}$.
4. If $N$ is an AND-node not occurring on an infinite path that is not a success node labelled with the clause $B_0 : \rho \leftarrow B_1 : \rho_1 \& \ldots \& B_k : \rho_k$, and if the truth-value $v_i$ of each of its descendant nodes $N_i$ labelled with $B_i : \rho_i$ is such that $v_i \geq \rho_i$, then $\nu(N) = \rho$, else, $\nu(N) = \bot$.

Before proceeding to derive further theoretical results, we present an example of an and-or tree associated with an ALP and a query.

(3.4) EXAMPLE. Let $I = \{\bot, t, f, T\}$ be the set of truth values ordered as follows:

```
   T
  /\  \\
 /   \\
 t   f
 \   /\  \\
  \ /   \\
    \bot
```

Consider the ALP $P$ over $I$ defined below:

$C1 : p(X) : t \leftarrow q(X) : t \& r(X) : t$

$C2 : q(a) : t \leftarrow$

$C3 : q(b) : f \leftarrow$
\[ C4 : r(a) : t \Leftarrow \]
\[ C5 : r(b) : f \Leftarrow \]

Then the and-or tree \( T(P, p(a) : t) \) is shown in Figure 1 below, together with the values of the nodes of the tree. (\( \varepsilon \) denotes the identity substitution.)

![Figure 1](image)

(3.5) DEFINITION. If \( T(P, A : \mu) \) is an and-or tree, then the height of this tree is the maximum number of or-nodes occurring along any path of the tree. The height of an and/or tree may be (countably) infinite.
(3.6) DEFINITION. An ALP P is said to be covered iff for every clause C in P, if X is a variable in the body of C, then X occurs in the head of C.

(3.7) STRONG SOUNDNESS THEOREM. If P is a covered ALP over I and T(P, A : μ) is finite (where A ∈ B_P), then T_P ↑ω(A) = ν(∃(T(P, A : μ))). (We use the notation ∃(Γ) to denote the root of the tree Γ).

PROOF. By induction on the height of the tree, we easily prove that

$$T_P ↑ω(A) ≥ ν(∃(T(P, A : μ)))$$

We now prove by induction on n that for all integers n,

$$ν(∃(Γ)) ≥ T_P ↑n(A).$$

As T_P ↑ω(A) = μ implies that T_P ↑n(A) = μ for some n < ω, the result follows.

**Base Case:** [n = 0] Trivially true as T_P ↑0(A) = ⊥ ≤ ν(∃(T(P, A : μ))).

**Inductive Case:** [n + 1] Suppose T_P ↑(n + 1)(A) = ρ. Then there is a finite set {C_1, ..., C_k} of clauses in P having instances C_iθ_i of the form

$$A : ρ_i ⇐ B_1^i : ψ_1^i & ... & B_{r_i}^i : ψ_{r_i}^i,$$

such that T_P ↑n |= B_1^i : ψ_1^i & ... & B_{r_i}^i : ψ_{r_i}^i for all i and \{ρ_1, ..., ρ_n\} ≥ ρ.

[Note that since P is covered and A is variable-free, each B_j^i is variable-free (1 ≤ j ≤ r_i)]. By the induction hypothesis, each B_j^i : ψ_j^i is the label of the root of an and-or tree T(P, B_j^i : ψ_j^i) and such that ν(∃(P, B_j^i)) ≥ T_P ↑ω(B_j^i) ≥ ψ_j^i. Thus, T_P ↑ω(A) ≤ ν(∃(T(P, A : μ))) as indicated in Figure 2 below.
At the beginning of this section, we briefly discussed the issue of efficient proof procedures for logics. We emphasize, once again, that this is vitally necessary because computers have a limited amount of memory, and it is normally not feasible to search very large spaces (and the space of all derivations in a logic forms a very large space). Thus, in 1965, a technique called resolution was introduced by J. A. Robinson [R65], and this method was proved to be a sound and complete proof procedure for first-order logic. Since then, resolution has been refined a great deal. One may take advantage of the fact that logic programs constitute the definite clause fragment of first-order logic, and try to improve the proof procedure. This was done, and a mechanism called SLD-resolution is currently in wide use (for more details, see [LL84]). However, in the presence of negation, SLD-resolution is known to be incomplete - a procedure called SLDNF-resolution has been proposed, but it is complete only for a small class of programs (cf. [K86,87,B86,CL87]). We now propose a technique called SLDa-resolution that is similar to SLD-resolution in spirit, but which can be used as a fast operational mechanism for a class of ALPs.

4 Efficient Operational Semantics for ALPs

We first define the class of programs that we are interested in.

(4.1) DEFINITION. An ALP $P$ (over $I$) is said to be well-defined iff, for every $A \in B_P$, if $C_1, C_2$ are clauses in $P$ whose heads are of the form $A_1 : \rho_1, A_2 : \rho_2$ respectively and $A_1$ and $A_2$ are unifiable, then either $\rho_1 \geq \rho_2$ or $\rho_2 \geq \rho_1$.

(4.2) COMMENT. As in general logic programs, only atoms are allowed to appear in the heads of clauses, every general logic program is well-defined (as an atom $A$ may be considered to be the annotated atom $A : t$ - or $A$ is known to be true, cf [BS87]). That the class of well defined programs strictly extends the class of general logic programs may be observed from the fact that the following ALP is
well defined, but is not a general logic program:

\[ p(a) : f = \]

(4.3) EXAMPLE. Take \( I \) to be the same as in Example (3.4). Then the following ALP is not well-defined:

\[
p(a) : t =
\]

\[
p(a) : f =
\]

(4.4) DEFINITION. If \( G \) is the query

\[ A_1 : \mu_1 & \ldots & A_k : \mu_k \]

and \( C \) is the annotated clause

\[ D : \beta = B_1 : \beta_1 & \ldots & B_r : \beta_r \]

such that for some \( 1 \leq i \leq k, \beta \geq \mu_i \) and \( D\theta = A_i\theta \) (where \( \theta \) is the most general unifier of \( A_i, D \) respectively), then the \textit{a-resolvent} of \( G \) and \( C \) w. r. t. \( A_i : \mu_i \) is the query

\[
(A_1 : \mu_1 & \ldots & A_{i-1} : \mu_{i-1} & B_1 : \beta_1 & \ldots & B_r : \beta_r & A_{i+1} : \mu_{i+1} & \ldots & A_k : \mu_k)\theta
\]

We may assume that \( C \) and \( G \) share no common variables, since the variables in \( C \) are bound by universal quantifiers.

(4.5) EXAMPLE. The \textit{a-resolvent} of the query \( G \equiv p(a) : t & p(b) : f \) with the clause

\[
p(X) : t = q(X) : t & r(X) : f
\]

w. r. t. \( p(a) : t \) is the query

\[
q(a) : t & r(a) : f & p(b) : f
\]

There is no resolvent of \( G \) and \( C \) w. r. t. \( p(b) : f \) because \( t \not\equiv f \).

(4.6) DEFINITION. An \textit{SLLD} \textit{deduction} from the initial goal \( G_0 \) and the ALP \( P \) (over \( I \)) is a sequence

\[ < G_0, C_1, \theta_1 >, \ldots, < G_i, C_{i+1}, \theta_i+1 >, \ldots \]
where each \( G_{j+1} \) is obtained by resolving \( G_j \) and a copy \( C_{j+1} \) of a clause in \( P \) via the most general unifier \( \theta_{j+1} \). (By a copy of \( C_{j+1} \), we mean the clause obtained by renaming all variables in \( C_{j+1} \) so that the renamed version of the clause and \( G_0, ..., G_j, C_0, ..., C_j \) share no common variables). The SLDa-tree associated with the above query is shown in Figure 3 below.

\[
\begin{align*}
& G_0 \quad \quad \quad \quad \quad \quad \quad \quad \quad C_1 \\
& \quad \quad \quad \quad \quad G_1 \quad \quad \quad C_2 \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad G_2
\end{align*}
\]

\textit{Figure 3}

(4.7) DEFINITION. An \textit{SLDa-refutation} (of length \( n \)) from the initial goal \( G_0 \) is an SLDa-deduction

\[
< G_0, C_1, \theta_1 >, ..., < G_n, C_{n+1}, \theta_{n+1} >
\]

with \( G_{n+1} \) being the empty goal (i. e. the empty conjunction of annotated atoms)

(4.8) SOUNDNESS THEOREM. If the query \( G_0 \) is \( A_1 : \mu_1 \land ... \land A_m : \mu_m \) and \( G_0 \) has an SLDa-refutation from the well defined ALP \( P \), then \( T_P \upharpoonright \omega \models (\exists)G_0 \).
PROOF. Suppose

\[ < G_0, C_1, \theta >, \ldots, < G_n, C_{n+1}, \theta_{n+1} >, \ldots \]

is an SLDa-refutation from \( P \). We will prove by induction on \( n \) that \( T_P \uparrow \omega \models (\exists)G_0 \).

*Base Case: \( |n = 0| \) Then \( G_1 \) is the empty query, i.e. \( G_0 \) is just \( A_1 : \mu_1 \), and there is a clause in \( P \) of the form

\[ A : \beta \leftarrow \]

where \( \beta \geq \mu_1 \) and such that \( A, A_1 \) are unifiable via mgu \( \theta_1 \). Since \( T_P \uparrow \omega \) is a model of \( P \), it is a model of the above clause; hence, \( T_P \uparrow \omega \models (\forall)A : \beta \), and

\[ T_P \uparrow \omega \models (\forall)A \theta_1 : \beta, \text{ i.e.} \]

\[ T_P \uparrow \omega \models (\forall)A \theta_1 : \mu_1 \text{ because } \beta \geq \mu_1. \]

*Inductive Case: \( |n + 1| \) Suppose

\[ < G_0, C_1, \theta_1 >, \ldots, < G_{n+1}, C_{n+2}, \theta_{n+2} > \]

is an SLDa-refutation of \( G_0 \) of length \( (n + 1) \). Then

\[ < G_1, C_2, \theta_2 >, \ldots, < G_{n+1}, C_{n+2}, \theta_{n+2} > \]

is an SLDa-refutation of \( G_1 \) of length \( n \). Therefore, by the induction hypothesis, \( T_P \uparrow \omega \models (\exists)G_1 \). But \( G_1 \) is of the form

\[ (A_1 : \mu_1 \& \ldots \& A_{i-1} : \mu_{i-1} \& (E_1 : \alpha_1 \& \ldots \& E_r : \alpha_r) \& A_{i+1} : \mu_{i+1} \& \ldots \& A_m : \mu_m) \theta_1 \]

where \( C_1 \) is the clause

\[ A' : \delta \leftarrow E_1 : \alpha_1 \& \ldots \& E_r : \alpha_r \]

such that \( A' \theta_1 = A_i \theta_1 \) and \( \delta \geq \mu_i \). \( T_P \uparrow \omega \models (\exists)G_1 \). But \( T_P \uparrow \omega \models (\forall)C_1 \theta_1 \). It follows that \( T_P \uparrow \omega \models (\exists)G_1. \square \)

(4.9) **Completeness Theorem for Ground Queries.** If \( P \) is a well-defined covered ALP, and \( G_0 \) is the goal \( A_1 : \mu_1 \& \ldots \& A_m : \mu_m \) where each \( A_i \in B_P \) and \( \mu_i \neq \bot \) for all \( 1 \leq i \leq m \), and if \( T_P \uparrow \omega \models (A_1 : \mu_1 \& \ldots \& A_m : \mu_m) \theta_1 \)
\( \mu_1 \& \ldots \& A_m : \mu_m \), then there is an SLD\( a \)-refutation from \( G_0 \) and \( P \).

**Proof.** Suppose \( P \) is an ALP, and \( T_P \uparrow \omega \models G_0 \). Therefore, for some \( n < \omega \), it is the case that \( T_P \uparrow n \models G_0 \), by an argument similar to that of Theorem 2.8. We proceed by induction on \( n \).

**Base Case:** \( |n| = 1 \) Suppose \( T_P \uparrow 1 \models G_0 \). Then, since \( P \) is well-defined, for every \( 1 \leq i \leq k \), there is a clause \( C_i \) in \( P \) having the following ground instance

\[
A_i : \beta_i \leftarrow
\]

where \( \beta_i \geq \mu_i \). Then \( < A_1 : \mu_1 \& \ldots \& A_m : \mu_m, C_1, \theta_1 >, < (A_2 : \mu_2 \& \ldots \& A_m : \mu_m)\theta_1, C_2, \theta_2 >, \ldots < (A_k : \mu_k)\theta_1 \ldots \theta_{k-1}, C_k, \theta_k > \) is an SLD\( a \)-refutation of \( G_0 \) from \( P \).

**Inductive Case:** \( |n| + 1 \) Suppose \( T_P \uparrow (n + 1) \models G_0 \). For each \( A_i : \mu_i \), there is a clause \( C \) in \( P \) of the form

\[
A_i' : \beta_i \leftarrow B_1 : \rho_1 \& \ldots \& B_r : \rho_r
\]

such that \( A_i' \theta = A_i \theta = A_i \) and such that \( T_P \uparrow n \models B_1 : \rho_1 \& \ldots \& B_r : \rho_r \). (We have used here the assumption that \( P \) is well-defined.) (As \( A_i \) is variable free, and as \( P \) is covered, each \( B_j \) is variable-free).

Thus, by the induction hypothesis, let \( S_i \) be an SLD\( a \)-refutation of \( (B_1 : \rho_1 \& \ldots \& B_r : \rho_r) \). The following is then an SLD\( a \)-refutation of \( A_i : \mu_i \):

\[
< A_i : \mu_i, C, \theta >, S_i
\]

This gives an SLD\( a \)-refutation \( \Gamma_i \) for each \( A_i : \mu_i (1 \leq i \leq m) \) and so \( \Gamma_1, \ldots, \Gamma_k \) is an SLD\( a \)-refutation of \( G_0 \). \( \Box \)

(4.10) **Example.** Let \( I \) be as described in Example (3.4). Consider the ALP \( P \) over \( I \) given below:

\[
\begin{align*}
C1 : P(X, a) : & \quad t \leftarrow q(X) : f \& r(X) : t \\
C2 : p(X, b) : & \quad t \leftarrow q(X) : t \& r(X) : f \\
C3 : q(a) : & \quad t \leftarrow \\
C4 : q(b) : & \quad t \leftarrow \\
C5 : r(a) : & \quad f \leftarrow \\
C6 : r(b) : & \quad f \leftarrow
\end{align*}
\]
and the query \( p(b, b) : t \). The following is an SLDa-refutation of this query.

\[
\begin{align*}
\langle p(b, b) : t, C_2, X/b \rangle, \\
\langle q(b) : t \& r(b) : f, C_4, \varepsilon \rangle, \\
\langle r(b) : f, C_6, \varepsilon \rangle.
\end{align*}
\]

The above Theorem (4.9) assures us that any variable free query that is satisfied by \( T_P \uparrow \omega \) is guaranteed to have an SLDa-refutation. We now briefly show how to remove the restriction to well-defined programs. Without any loss of generality, we assume that no two distinct clauses in any ALP contain any common variables.

(4.11) DEFINITION. An ALP \( P \) over \( I \) is said to be closed iff for any two distinct annotated clauses \( C_1, C_2 \) in \( P \) of the form

\[
A_1 : \rho_1 \leftarrow B_1 : \mu_1 \& \ldots \& B_k : \mu_k
\]

\[
A_2 : \rho_2 \leftarrow D_1 : \psi_1 \& \ldots \& D_m : \psi_m
\]

such that \( A_1 \) and \( A_2 \) are unifiable (via \( \text{mgu} \ \theta \)), and \( \rho_1 \) and \( \rho_2 \) are incomparable (i.e. \( \rho_1 \nleq \rho_2 \) and \( \rho_2 \nleq \rho_1 \)) it is the case that there is a (renamed version) of the annotated clause

\[
A_1 \theta : \sqcup \{ \rho_1, \rho_2 \} \leftarrow B_1 : \mu_1 \& \ldots \& B_k : \mu_k \& D_1 : \psi_1 \& \ldots \& D_m : \psi_m
\]

denoted \( \lambda(C_1, C_2) \)

(4.12) DEFINITION. Let \( P \) be an ALP over \( I \). Then we define:

\[
\begin{align*}
A_1(P) &= P \\
A_{n+1}(P) &= \{ \lambda(C_1, C_2) | C_1, C_2 \in A_n(P) \} \cup A_n(P)
\end{align*}
\]

For any ALP \( P \) there is some integer \( n \) such that \( A_n(P) = A_{n+1}(P) \). We call \( A_n(P) \) the closure of \( P \) and denote it by \( CL(P) \).

For example, the annotated program \( P \) of Example (4.3) is not closed. The closure of \( P \) is obtained by adding the unit clause \( p : \top \) to \( P \). Here, \( A_2(P) = A_3(P) \).

The proof of the following theorem is almost immediate:
(4.13) THEOREM. Suppose $P$ is a closed ALP. Then:
1. $I$ is a model of $P$ iff $I$ is a model of $CL(P)$
2. $T_P = T_{CL(P)}$

PROOF. (2) can be proved by showing that for all $n$, $T_{A_n(P)} = T_{A_{n+1}(P)}$. This is achieved by an easy induction on $n$. (1) is a consequence of (2) and Theorem 2.5. □

Thus, we know that given any finitely closed ALP $P$, there exists a closed ALP $CL(P)$ such that $P$ and $CL(P)$ are logically equivalent. However, $CL(P)$ possesses the following property which $P$ may lack.

(4.14) THEOREM. Suppose $P$ is a closed ALP and $A \in B_P$. If $T_P \uparrow \omega \models A : \mu$ (where $\mu \neq \bot$), then there is an annotated clause in $P$ having a ground instance of the form:

$$A : \mu' \leftarrow B_1 : \psi_1 \& \ldots \& B_k : \psi_k$$

where $\mu' \supseteq \mu$ and $T_P \uparrow \omega \models B_1 : \psi_1 \& \ldots \& B_k : \psi_k$.

PROOF. Suppose $T_P \uparrow \omega \models A : \mu$. As $T_P \uparrow \omega$ is a supported model of $P$, it follows that there is a finite set of ground instances of annotated clauses in $P$ of the form:

$$A : \rho_i \leftarrow D^i_1 : \phi^i_1 \& \ldots \& D^i_m : \phi^i_m,$$

$1 \leq i \leq r$ for some $r$ such that

1. $\cup\{\rho_1, ..., \rho_r\} = \delta \supseteq \mu$
2. $T_P \uparrow \omega \models D^i_1 : \phi^i_1 \& \ldots \& D^i_m : \phi^i_m$, for all $1 \leq i \leq r$.
3. $\rho_i$ and $\rho_j$ are incomparable for all $i \neq j$.

As $P$ is closed, the clause:

$$A : \delta \leftarrow D^1_1 : \phi^1_1 \& \ldots \& D^r_m : \phi^r_m,$$

is in $P T_P \uparrow \omega \models D^1_1 : \phi^1_1 \& \ldots \& D^r_m : \phi^r_m$, and as $\delta \supseteq \mu$, the proof is complete. □
(4.15) **SOUNDNESS & COMPLETENESS THEOREMS.** Let $P$ be an ALP over $I$ and $CL(P)$ be its closure and let $A \in B_P$. Then,

1. If $P$ is covered and there is an SLDa-refutation of $A : \mu$ from $CL(P)$, then $T_P \uparrow \omega \models A : \mu$.

2. If $T_P \uparrow \omega \models A : \mu$ then there is an SLDa-refutation of $A : \mu$ from $CL(P)$.

PROOF. Similar to the proof of theorems 4.8. and 4.9 respectively.

\[\square\]

5 **Paraconsistent ALPs**

Having investigated the declarative and operational semantics of the class of ALPs, we investigate three useful instances of the semantics developed in the preceding sections.

(5.1) **DEFINITION.** An ALP over $I$ is **negation-inconsistent** iff there is some variable free annotated atom $A : \mu, (A \in B_P)$ such that $T_P \uparrow \omega \models A : \mu$ and $T_P \uparrow \omega \not\models A : \neg(\mu)$.

The notion of negation inconsistency is very similar to that of $\neg$-inconsistency of Arruda [AR79]. We now define a notion similar to that of non-triviality.

(5.2) **DEFINITION.** An ALP $P$ over $I$ is **non-trivial** iff there is some variable free annotated atom $A : \mu$ such that $T_P \uparrow \omega \not\models A : \mu$.

(5.3) **DEFINITION.** An ALP $P$ over $I$ is **paraconsistent** iff $P$ is both negation-inconsistent and non-trivial.

(5.4) **GENERALLY HORN PROGRAMS:** In this case, $I = \{ \bot, t, f, T \}$, and the $\preceq$ ordering on $I$ is as defined in Example (3.4). $\neg$ is defined as: $\neg(\bot) = \top, \neg(\top) = \bot, \neg(t) = f, \neg(f) = t$. Thus, consider the following program:

\[
\begin{align*}
p(a) : t & \leftarrow \\
p(a) : f & \leftarrow 
\end{align*}
\]
\( q(a) : t \Leftarrow \)

The least model of this program assigns \( \top \) to \( p(a) \), and \( t \) to \( q(a) \). That this program is negation-inconsistent is seen from the fact that \( T_P \uparrow \omega \models p(a) : t \) and \( T_P \uparrow \omega \nvdash p(a) : f \). That it is non-trivial follows from the fact that \( T_P \uparrow \omega \nvdash q(a) : f \). Thus, this generally Horn program is paraconsistent. All the results of Sections 2, 3 and 4 apply to generally Horn programs.

(5.5) QUANTITATIVE LOGIC PROGRAMS: it is common for expert systems to associate some degree of uncertainty with the information contained in it. Recently, a logical framework for reasoning with uncertain information was proposed by Subrahmanian [S87]. Here \( I = [0, 1] \cup \{ \top \} \), and the ordering \( \preceq \) is defined as shown below:

\[
\begin{array}{c}
\top \\
1 \\
0.5 \\
0 \\
\end{array}
\]

Negation is defined over \( I \) as:
\[
\neg(\top) = \top \\
\neg(X) = 1 - X \text{ if } X \in [0, 1].
\]

The results of Sections 2, 3 and 4 can be easily seen to hold.

(5.6) EVIDENTIAL LOGIC PROGRAMMING: Subsequent to the work of (5.5) above, it was proposed that a user of an expert system could be told the degree of inconsistency by changing \( I \) to the set
\([0, 1] \times [0, 1]\). Intuitively, if a proposition is assigned the truth value \([\mu_1, \mu_2]\), then the degree of belief in that proposition is \(\mu_1\), while the degree of disbelief in that proposition is \(\mu_2\). Note that \(\mu_1, \mu_2\) are independently assigned, and do not depend on each other. Thus, it is quite possible that \(\mu_1 = \mu_2 = 1\). The case of absolute inconsistency occurs when \([1, 1]\) is assigned to a proposition, while when a proposition is assigned \([0, 0]\), nothing is known about it.

The \(\leq\) ordering on \(I\) is: \([\mu_1, \mu_2] \leq [\rho_1, \rho_2]\) iff \(\mu_1 \leq \rho_1\) and \(\mu_2 \leq \rho_2\) where \(\leq\) is the usual less than or equals ordering on the reals. It can easily be seen that \(I\) is a complete lattice under \(\leq\). The bottom element (resp. top element) of this lattice is \([0, 0]\) (resp. \([1, 1]\)). Negation is defined as:

\[-([\mu_1, \mu_2]) = [\mu_2, \mu_1]\]

The results of Sections 2, 3, 4 all follow. Consider the ALP over \(I\) below:

\[
A : [0, 1] \leftarrow \\
A : [1, 0] \leftarrow \\
B : [0.2, 0.3] \leftarrow
\]

The least model of this program assigns \([1, 1]\) to \(A\), and \([0.2, 0.3]\) to \(B\). The program is negation-inconsistent as \(T_P \uparrow \omega \models A : [0, 1]\) and \(T_P \uparrow A : [1, 0]\) (as, \(\neg([0, 1]) = [1, 0]\)). It is non-trivial because \(T_P \uparrow \omega \nvDash B : [1, 1]\). Thus, this ALP is paraconsistent, and the degree of uncertainty (under-definedness) of \(B\) is the distance of the point \([0.2, 0.3]\) from the line \(x + y = 1\) (in the sense of analytic geometry, see [S87a] for further details.)

We have thus seen from the above examples that generally Horn programs, quantitative logic programs, and evidential logic programs all display some traits of paraconsistent logics. In each of the above cases, \(\neg\) is injective and strict (i.e. \(\neg(\bot) = \bot\)) where \(\bot\) is a generic symbol used to define the smallest element in the lattice under consideration. Strict maps are well known in lattice theory (cf. Dana Scott [S82], Gierz et al [G80]).

(5.7) DEFINITION. A mapping \(f\) from a complete lattice \(L\) to \(L\)
is strict iff \( f(\bot) = \bot \) where \( \bot \) is the least element of \( L \). (as \( L \) is a complete lattice, such an element is guaranteed to exist).

(5.8) REMARK. Suppose \( P \) is an ALP over the complete lattice (under the as-yet unspecified ordering \( \leq \)) \( I \) of truth values, and suppose \( \neg \) is an injective and strict function from \( I \) to \( I \). Also suppose that \( I \) contains at least two elements. Then there exists an ALP \( P' \) over \( I \) such that

1. If \( A \) is in the language of \( P \), then \( T_P \uparrow \omega \models A : \mu \) iff \( T_{P'} \uparrow \omega \models A : \mu \) and

2. \( P' \) is paraconsistent.

That \( P \) is itself negation-inconsistent follows because for all \( A \in B_P \), it is the case that \( T_P \uparrow \omega \models A : \bot \) and \( T_P \uparrow \omega \models A : \neg(\bot) \) [as \( \neg \) is strict, \( \neg(\bot) = \bot \)]. If \( P \) is nontrivial, we may take \( P \) to be \( P \) itself, and we would be done. If \( P \) is trivial, let \( P' \) be the program \( P \cup \{ q : \bot \iff \} \) where \( q \) is a 0-ary predicate symbol not in \( P \). Then \( T_{P'} \uparrow \omega \not\models q : \mu_1 \) where \( \mu_1 \neq \bot \) is an element of \( I \). (Such an element must exist as \( I \) contains at least two elements). Thus \( P' \) is non-trivial, and hence, paraconsistent.

The above theorem shows that given any ALP \( P \), one can find a paraconsistent ALP whose meaning is the same as \( P \).

6 Conclusions

In this paper, we have introduced a class of paraconsistent logics, and shown how logic programming languages based on such logics can be designed. We have demonstrated three specific examples of useful instances of the logic programming language scheme proposed in this paper – viz. the generally Horn programs, the quantitative logic programs, and the evidential logic programs. We have been able to characterize the semantics of ALPs in terms of the fixed points of a monotone operator from interpretations to interpretations, and have shown that this precisely corresponds to the model theoretic semantics of these programs. Lastly, we have remarked that every logic program may be considered to be a paraconsistent one [Remark (5.8)].
Acknowledgements. We are indebted to Aida Batarekh for several interesting conversations on paraconsistency. VSS thanks Professor Newton da Costa for making many thought-provoking suggestions during our correspondence.

References


of Logic Programming, 2, 4, pps 295-312.

[G80] Gierz, G. Hoffman, K. H., Keimel, K., Lawson, J. D., Mislove, M., Scott,

Reinhold.

4, 4, pps 289-308.

manuscript.

Verlag.

Theoretical Computer Science, 39, pps 115-125.

Colloquium on Automata, Languages and Programming, Springer Verlag.

[S87] Subrahmanian, V. S. (1987) On the semantics of quantitative logic pro-

in logic programming, Logic Colloquium '87 (European Summer Meeting of the

Functions, in Logic Programming, (eds. Clark, K., L, and Tarnlund, S. A.),

[VE86] Van Emden, M. H. (1986) Quantitative deduction and its fixpoint the-
ory, J. of Logic Programming, 4, 1, pps 37-53.

logic as a programming language, JACM, 23, 4, pps 733-742.

Received March 02, 1988.

School of Computer & Information Science
313 Link Hall
Syracuse University
Syracuse, N. Y. 13244-1240, USA