ParaLog_e: A Paraconsistent Evidential Logic Programming Language

Bráulio Coelho Ávila
Pontifical Catholic University of Paraná/UEPG*/IEA†
Department of Informatics
R. Imaculada Conceição, 1155
80215-901 Curitiba - PR, Brazil
avila@ccet.pucpr.br

Jair Minoro Abe
Paulista University/IEA†
Department of Informatics
R. Dr. Bacelar, 1212
04026-002 São Paulo - SP, Brazil

José Pacheco de Almeida Prado
Paulista University/IEA†
Department of Informatics
R. Chile, 845
14020-610 Ribeirão Preto - SP, Brazil

Abstract

Inconsistency is a natural phenomenon arising from the description of the real world. This phenomenon may be encountered in several situations. Nevertheless, human beings are capable of reasoning adequately. The automation of such reasoning requires the development of formal theories. Paraconsistent Logic was proposed by N.C.A. da Costa to provide tools to reason about inconsistencies.

This paper describes an extension of the ParaLog Logic Programming Language, called ParaLog_e that allows direct handling of inconsistency. Languages such as ParaLog_e, capable of merging Classical Logic Programming concepts with those of inconsistency, widen the scope of Logic Programming applications in environments presenting conflicting beliefs and contradictory information.

1. Introduction

The employment of logic systems allowing reasoning about inconsistent information is an area of growing importance in Computer Science, Data Base Theory and Artificial Intelligence. For instance, if a knowledge engineer is designing a knowledge base KB, related to a domain D, he may consult n experts in that domain. For each expert e_i, $1 \leq i \leq n$, of dominion D, he will obtain some information and will present it in some logic such as a set of sentences KB_i, for $1 \leq i \leq n$. A simple way of combining the knowledge amassed from all experts in a single knowledge base KB is:

$$KB = \bigcup_{i=1}^{n} KB_i$$

However, certain KB_i and KB_j bases may contain *conflicting* propositions — p and $\neg p$. In such case, p might be a logic consequence of KB_i, while $\neg p$ might be a logic consequence of KB_j. Therefore, KB is inconsistent and consequently meaningless, because of the lack of models. However, the knowledge base KB it not a useless set of information.

There are some arguments favoring this standpoint [12], as follows:

- certain subsets KB may be inconsistent and express significant information. Such information cannot be disregarded;
- the disagreement among specialists in a given domain may be significant. For instance, if physician M_1 concludes patient X suffers from a fatal cancer, while physician M_2 concludes that that same patient suffers from cancer, but a benign one, the patient will probably want to know the causes of

* State University of Ponta Grossa, Brazil
† Institute for Advanced Studies, University of São Paulo, Brazil

0-8186-8052-0/97 $10.00 © 1997 IEEE

such disagreement. This disagreement is significant because it may lead patient X to take appropriate decisions — for instance, to get the opinion of a third physician.

The reasoning for the last item is that it is not always advisable to find ways to exclude formulas identified as causing inconsistency(ies) in KB, because many times important information may be removed. In such cases, the very existence of inconsistency is important.

Though inconsistency is an increasingly common phenomenon in programming environments — especially in those possessing a certain degree of distribution — it cannot be handled, at least directly, by Classical Logic, on which most of the current logic programming languages are based.

Thus, one has to resort to alternatives to classical logic; it is therefore necessary to search for programming languages based on such alternatives.

The work by da Costa *et al.* [14] proposes a variation of the logic programming language Prolog [7] [8], based on Annotated Logic Q_τ and in [5] [19] [6] [11] [13] [1] [2], that allow inconsistency to be handled directly. The proposed logic programming language is called Paraconsistent Prolog — ParaLog.

This paper describes an extension of the ParaLog Logic Programming Language. This extended language — called ParaLog_e — employs evidential logic programming. The employment of *evidential reasoning* was proposed by Subrahmanian in [20] [6]. For that, Subrahmanian proposes an infinitely valued *paraconsistent logic* where the truth-values are members of $\{x \in \Re \mid 0 \leq x \leq 1\} \times \{x \in \Re \mid 0 \leq x \leq 1\}$. Subrahmanian's work is an extension of the logic concepts for quantitative reasoning in logic programming [21] [19].

2. The ParaLog_e Language

In several events in the real world, *evidences* [18] play a fundamental role in decision-making. Most human decision-making is based on *previous experiences.* Therefore, an individual, when faced with decision-making and imprecise information, considers all possibilities — investigates all evidences — and finally decides on the course of action to be taken.

For instance, if the following evidences exist: less than 10% of animals can fly; more than 90% of birds can fly; generally all birds are animals; Tweety is a bird; it is intuitive to represent the proposition *less than 10% of animals can fly* as:

$$fly(X) : 0.1 \leftarrow animal(X) : 1.0$$

But how to represent that *over 90% of animals cannot fly?*

In this example, the answer to the following query: *Can Tweety fly?*, may not be so simple and may lead to erroneous conclusions.

The ParaLog language, using the annotation concept, may relate just one evidence to proposition p. However, it has been shown that the use of two evidences related to the same proposition p may increase its expression power. According to Subrahmanian [20], such two-evidence annotation may be thought of as one *evidence favoring* p and one *evidence opposed* to p. No restriction is applied to these evidences excepting that they be within the interval $\{x \in \Re \mid 0 \leq x \leq 1\}$.

In the previous example, the same proposition could be represented in ParaLog_e as follows:

$$fly(X) : [0.1, 0.9] \leftarrow animal(X) : [1.0, 0.0]$$

thereby allowing an increase in the expression power of that proposition, leading to proper conclusions. That is, the answer to the previous query must state that there is an 80% inconsistency for the fact that Tweety can fly.

The ParaLog_e language is based on a non-classical — annotated (evidential) paraconsistent — logic that renders the use of non-logic extensions unnecessary. The semantics of a ParaLog_e program is based on the semantics on Herbrand's minimal model. The new resources introduced by ParaLog_e are therefore application-independent and based on 1st order annotated logic, complete and correct in relation to the semantics employed.

3. Syntax of ParaLog_e

The implementation of ParaLog_e is based in Edimburgh's syntax, plus new syntax elements related to Evidential Logic Programming [20].

Definition 3.1 (Alphabet) The basic ParaLog_e *alphabet* possesses the same set of symbols of the standard Prolog, plus the symbol ":" and the evidential symbol. ParaLog_e symbols are:

1. letters: $a, b, \ldots, z, A, B, \ldots, Z$
2. digits: $0, 1, \ldots, 9$
3. special symbols: $\_, +, -, /, *$ and the space
4. punctuation marks: (,), . ,",",
5. connective symbols: & (conjunction), <-- (implication), not (negation)
6. annotation symbol: ":"

7. annotation symbol[1]: $[\rho_1, \rho_2]$

8. evidential constant: $\rho_1, \rho_2 \qquad (0 \leq \rho_1, \rho_2 \leq 1)$

Definition 3.2 (Expression) A ParaLog_e *expression* is any finite sequence of its alphabet symbols.

Definition 3.3 (Atom) A ParaLog_e *atom* is:

1. every expression made up of letters and digits, starting with a minuscule;

2. an expression made up of digits, having at most an occurrence of the symbol ".";

3. every expression — including a space — delimited by quotation marks.

Definition 3.4 (Constant) A ParaLog_e *constant* is defined as:

1. an atom; or

2. an element of the lattice $\mathcal{T} = \{x \in \Re \mid 0 \leq x \leq 1\} \times \{x \in \Re \mid 0 \leq x \leq 1\}$, called evidential constant.

Definition 3.5 (Variable) A ParaLog_e *variable* is defined as:

1. an expression of letters and digits the first element of which is a capital letter; or

2. the symbol "_", called anonymous variable.

Definition 3.6 (Term) A ParaLog_e *term* is inductively defined as:

1. a variable is a term;

2. a constant is a term;

3. if f is an atom and f has the role of a functional n-ary symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term; and

4. a ParaLog_e expression is a term if and only if it is obtained by applying one of the foregoing — 1 to 3 — conditions.

Definition 3.7 (Atomic Formula) An *evidential atomic formula* is the expression of form $p(t_1, \ldots, t_n)$: $[\mu_1, \mu_2]$, for $n \geq 0$ and $[\mu_1, \mu_2] \in \mathcal{T}$, where $t_1, \ldots, t_n$ are terms and p is an atom in the role of a predicate n-ary symbol. For simplification purposes, when $n = 0$, $p()$: $[\mu_1, \mu_2]$ may be written as p: $[\mu_1, \mu_2]$.

[1]The lattice used is defined in [20]

Thus, an atom may simultaneously play the role of one or more functional symbols or predicates of different arities. This multiple use of the same atom does not result in ambiguity since the context of a program always points out the role it represents.

Definition 3.8 (Evidential Clause) The set of ParaLog_e *evidential clauses* if inductively defined as:

1. if p is an evidential atomic formula, then p is an evidential clause, called unitary evidential clause;

2. if p: $[\mu, \nu]$ and q_1: $[\mu_1, \nu_1], \ldots, q_n$: $[\mu_n, \nu_n]$ are evidential atomic formulas, then the expression

$$p\text{: } [\mu, \nu] \text{ <-- } q_1\text{: } [\mu_1, \nu_1] \& \ldots \& q_n\text{: } [\mu_n, \nu_n]$$

is an evidential clause, called non-unitary evidential clause, where p: $[\mu, \nu]$ is the head and q_1: $[\mu_1, \nu_1] \& \ldots \& q_n$: $[\mu_n, \nu_n]$ is the body of the clause;

3. if p_1: $[\mu_1, \nu_1], \ldots, p_n$: $[\mu_n, \nu_n]$ are evidential atomic formulas, then

$$\text{<-- } p_1\text{: } [\mu_1, \nu_1] \& \ldots \& p_n\text{: } [\mu_n, \nu_n]$$

is an evidential clause, called *objective clause*;

4. a ParaLog_e expression is a clause if and only if it is obtained by applying one of the foregoing — 1 to 3 — conditions.

Definition 3.9 (ParaLog_e Program) A *ParaLog_e program* is a finite non-empty set of unitary and non-unitary evidential clauses.

A ParaLog_e program is presented below showing the example of Tweety bird proposed on section 2. This example has been adapted from Ng & Subrahmanian's work [16].

Example 3.1 A ParaLog_e program on Tweety bird

```
flies(X):[0.1,0.9] <--
            animal(X):[1.0,0.0].
flies(X):[0.9,0.1] <--
            bird(X):[1.0,0.0].
animal(X):[1.0,0.0] <--
            bird(X):[1.0,0.0].
bird(tweety):[1.0,0.0].
```

In this example, the fact that *Tweety is a bird* is represented as:

```
bird(tweety):[1.0,0.0].
```

This clause may be read as: "It is known, with *absolute favourable evidence* and with *no contrary evidence* that Tweety is a bird".

4. Semantics of ParaLog_e

4.1. Declarative and Procedural Semantics

The results and concepts of Evidential Logic Programs [20] — ELPs — can be adapted to the ParaLog_e programs.

Similarly to a standard Prolog program, a P_0 program and a Q_0 query must undergo a regularization process, where:

- every anonymous variable is replaced by a new distinct variable;
- all atoms occurring in more than one role will be renamed, so that at the end of that process there are no atoms with the same name in different roles.

The P_1 program and the Q_1 query resulting from this regularization process are equivalent to P_0 and Q_0 respectively, as shown in [7], for instance. Differently from the standard Prolog, a ParaLog_e program must also undergo two more syntactic transformations:

1. elimination of the negation; and
2. closure.

These two syntactic transformations are necessary to eliminate the facilities introduced by the ParaLog_e language that are not part of the ELPs syntax.

In the first transformation, program P_1 and Q_1 query undergo the negation elimination process. In this process, all occurrences of *not* $p{:}\,[\mu,\nu]$ evidential atomic formulas are replaced by $p{:}\,[\nu,\mu]$ equivalent atomic formulas. The elimination of P_1 and Q_1 negation results in program P_2 and Q_2 query.

The last syntactic transformation is the closure, as described in [5], proving that program $CL(P_2)$ resulting from this transformation is logically equivalent to P_2; that is I is a P_2 model if and only if I is a $CL(P_2)$ model.

Program $CL(P_2)$ and query $CL(Q_2)$ resulting from this syntactic transformation process are equivalent to P_0 and Q_0 in ParaLog_e, respectively. Program $CL(P_2)$ is equivalent to a ELP E and query $CL(Q_2)$ is equivalent to a C query, if and only if:

1. a clause $p{:}\,[\mu,\nu]$ occurs in $CL(P_2)$ if and only if there is a clause $p{:}\,[\mu,\nu]\leftarrow\!-$ in E;
2. a non-unitary ParaLog_e clause $p{:}\,[\mu,\nu] \leftarrow\!- q_1{:}\,[\mu_1,\nu_1]\&\ldots\&q_n{:}\,[\mu_n,\nu_n]$ occurs in $CL(P_2)$ if and only if there is a clause $p{:}\,[\mu,\nu] \leftarrow\!- q_1{:}\,[\mu_1,\nu_1]\&\ldots\&q_n{:}\,[\mu_n,\nu_n]$ in E;
3. $CL(Q_2)$ is in form $\leftarrow\!- C_1{:}\,[\mu_1,\nu_1]\&\ldots\&C_n{:}\,[\mu_n,\nu_n]$ if and only if C is in form $C_1{:}\,[\mu_1,\nu_1]\&\ldots\&C_n{:}\,[\mu_n,\nu_n]$.

At the end of these syntactic transformations, the resulting program and query can be handled as *well-behaved* ELPs, where resolution-SLDe can be applied.

4.2. Operational Semantics

ParaLog_e inference engine offers an operational semantics for the implemented language; its execution is based on the resolution-SLDe method.

In this inference engine the selection function f maps target $C_1{:}\,[\mu_1,\nu_1]\&\ldots\&\ C_n{:}\,[\mu_n,\nu_n]$ in literal $C_i{:}\,[\mu_i,\nu_i]$, where $1 \leq i \leq n$ and $[\mu_i,\nu_i] = \mathbf{sup}\{[\mu_1,\nu_1],\ldots,[\mu_n,\nu_n]\}$. However, the selection function f employed by the ParaLog_e inference engine is not the same standard selection function described in [15].

Also, the procedure to select the program clauses does not follow standard strategy. In this inference engine program clauses are selected so that the selected clause $C{:}\,[\mu,\nu]$ has evidence $[\mu,\nu]$ equal to the supreme of the set formed by the candidate clauses evidences.

These two selection strategies cause the inference engine refutation procedure to simulate a search similar to the best-first search in the program clauses $CL(P_2)$ refutation tree.

Thus, given a program P and a query Q, the ParaLog_e inference engine provides as an answer an evidence $[\rho,\psi]$, as previously described in this section, so that $[\rho,\psi] \in \mathcal{T}$. In the cases in which $[\rho,\psi] \neq [0,0]$, the answer also includes a replacement θ for the variables of Q.

5. Programming in ParaLog_e

A small knowledge base in the domain of medicine is presented as a ParaLog_e program. The development of this small knowledge base was subsidized by the information provided by three experts in Medicine. The first two specialists — clinicians — provided six[2] diagnosis rules for two diseases: disease1 and disease2. The last specialist — a pathologist — provided information on four symptoms: symptom1, symptom2, symptom3 and symptom4. This example was adapted from da Costa and Subrahmanian's work [11].

Example 5.1 A small knowledge base in Medicine implemented in ParaLog_e

[2] The first four diagnosis rules were supplied by the first expert clinician and the two remaining diagnosis rules were provided by the second expert clinician.

```
disease1(X):[1.0,0.0] <--
            symptom1(X):[1.0,0.0] &
            symptom2(X):[1.0,0.0].
disease2(X):[1.0,0.0] <--
            symptom1(X):[1.0,0.0] &
            symptom3(X):[1.0,0.0].
disease1(X):[0.0,1.0] <--
            disease2(X):[1.0,0.0].
disease2(X):[0.0,1.0] <--
            disease1(X):[1.0,0.0].
disease1(X):[1.0,0.0] <--
            symptom1(X):[1.0,0.0] &
            symptom4(X):[1.0,0.0].
disease2(X):[1.0,0.0] <--
            symptom1(X):[0.0,1.0] &
            symptom3(X):[1.0,0.0].
symptom1(john):[1.0,0.0].
symptom1(bill):[0.0,1.0].
symptom2(john):[0.0,1.0].
symptom2(bill):[0.0,1.0].
symptom3(john):[1.0,0.0].
symptom3(bill):[1.0,0.0].
symptom4(john):[1.0,0.0].
symptom4(bill):[0.0,1.0].
```

In this example, several types of queries can be performed. Table 1 shows some query types, the evidences provided as answers by the ParaLog_e inference engine and their respective meaning.

The knowledge base implemented in Example 5.1 may alse be implemented in standard Prolog, as shown in Example 5.2.

Example 5.2 Knowledge base of Example 5.1 implemented in standard Prolog

```
disease1(X):-
      symptom1(X),
      symptom2(X).
disease2(X):-
      symptom1(X),
      symptom3(X).
disease1(X):-
      not disease2(X).
disease2(X):-
      not disease1(X).
disease1(X):-
      symptom1(X),
      symptom4(X).
disease2(X):-
      not symptom1(X),
      symptom3(X).
symptom1(john).
symptom3(john).
```

Query and Answer Form		Meaning
query	disease1(bill):[1.0,0.0]	Does Bill have disease1?
evidence	[0.0,1.0]	Bill does not have disease1.
query	disease2(bill):[1.0,0.0]	Does Bill have disease2?
evidence	[1.0,0.0]	Bill has disease2.
query	disease1(john):[1.0,0.0]	Does John have disease1?
evidence	[1.0,1.0]	The information on John's disease1 is inconsistent.
query	disease2(john):[1.0,0.0]	Does John have disease2?
evidence	[1.0,1.0]	The information on John's disease2 is inconsistent.
query	disease1(bob):[1.0,0.0]	Does Bob have disease1?
evidence	[0.0,0.0]	The information on Bob's disease1 is unknown.

Table 1. Query and Answer Forms in ParaLog_e

```
symptom3(bill).
symptom4(john).
```

In this example, several types of queries can be performed as well. Table 2 shows some query types provided as answers by the standard Prolog and their respective meaning.

Starting from Examples 5.1 and 5.2 it can be seen that there are different characteristics between implementing and consulting in ParaLog_e and standard Prolog. Among these characteristics, the most important are:

1. the semantic characteristic; and
2. the execution control characteristic.

The first characteristic may be intuitively observed when the program codes in Example 5.1 and 5.2 are placed side by side. That is, when compared to ParaLog_e, the standard Prolog representation causes loss of semantic information on facts and rules. This is due to the fact that standard Prolog cannot directly represent the negation of facts and rules.

In Example 5.1, ParaLog_e program presents a four-valued evidence representation. However, the information loss may be greater for a standard Prolog program, if the facts and rules of ParaLog_e use the intermediate evidence of lattice $\mathcal{T} = \{x \in \Re \mid 0 \leq x \leq 1\} \times \{x \in \Re \mid 0 \leq x \leq 1\}$.

Query and Answer Form		Meaning
query	disease1(bill)	Does Bill have disease1?
answer	loop	System enters into an infinite loop.
query	disease2(bill)	Does Bill have disease2?
answer	loop	System enters into an infinite loop.
query	disease1(john)	Does John have disease1?
answer	yes	John has disease1.
query	disease2(john)	Does John have disease2?
answer	yes	John has disease2.
query	disease1(bob)	Does Bob have disease1?
answer	no	Bob does not have disease1.

Table 2. Query and Answer Forms in standard Prolog

This last characteristic may be observed in Tables 1 and 2. These two tables show five queries and answers, presented and obtained both in ParaLog_e and standard Prolog program.

The answers obtained from the two approaches present major differences. That is, to the first query: "Does Bill have disease1?", ParaLog_e answers "Bill has disease1", while the standard Prolog enters into a loop. This happens because the standard Prolog inference engine depends on the ordination of facts and rules to reach deductions. This, for standard Prolog to be able to deduct an answer similar to ParaLog_e, the facts and rules in Example 5.2 should be reordered. On the other hand, as the ParaLog_e inference engine does not depend on reordering facts and rules, such reordering becomes unnecessary.

In the second query: "Does Bill have disease2?", ParaLog_e answers that "Bill does not have disease2", while the standard Prolog enters into a loop. This happens for the same reasons explained in the foregoing item.

In the third query: "Does John have disease1?", ParaLog_e answers that the information on John's disease1 is inconsistent, while the standard Prolog answers that "John has disease1". This happens because the standard Prolog inference engine, after reaching the conclusion that "John has disease1" does not check whether there are other conclusions leading to a contraction. On the other hand, ParaLog_e performs such check, leading to more appropriate conclusions.

In the fourth query: "Does John have disease2?", ParaLog_e answers that the information on John's disease2 is inconsistent, while the standard Prolog answers that "John has disease2". This happens for the same reasons explained in the foregoing item.

In the last query: "Does Bob have disease1", ParaLog_e answers that the information on Bob's disease1 is unknown, while the standard Prolog answers that "Bob does not have disease1". This happens because the standard Prolog inference engine does not distinguish the two possible interpretations for the answer no[3]. On the other hand, the ParaLog_e inference engine, being based on an infinitely valued paraconsistent evidential logic, allows the distinction to be made.

In view of the above, it is demonstrated that the use of the ParaLog_e language may handle several Artificial Intelligence questions more naturally.

6. Conclusions

Inconsistency is a natural phenomenon arising from the description of the real world. This phenomenon may be encountered in several situations. Nevertheless, human beings are capable of reasoning adequately. The automation of such reasoning requires the development of formal theories. Paraconsistent Logic was proposed by N.C.A. da Costa [9] [10] [11] [12] [13], to provide tools to reason about inconsistencies.

Paraconsistent Logic, despite having been initially developed from the purely theoretical standpoint, found in recent years extremely fertile applications in Computer Science [4] [3] [17], thus solving the problem of justifying such logic systems from the practical standpoint.

This paper described an extension of the ParaLog Logic Programming Language [14], called ParaLog_e. Languages such as ParaLog_e, capable of merging Classical Logic Programming concepts with those of inconsistency, widen the scope of Logic Programming applications in environments presenting conflicting beliefs and contradictory information.

References

[1] Abe, J.M., *Fundamentos da Lógica Anotada*, (Foundations of Annotated Logics), Ph.D. Thesis, Faculdade de Filosofia, Letras e Ciências Humanas, Universidade de São Paulo, São Paulo, Brazil, 1992. (in portuguese)

[2] Abe, J.M., *On Annotated Model Theory*, Coleção Documentos, Série: Lógica e Teoria da Ciência Nº11, Instituto de Estudos Avançados, Universidade de São Paulo, São Paulo, Brazil, June, 1993.

[3] Abe, J.M.; Prado, J.P.A.; Ávila, B.C., *On a Class of Paraconsistent Multimodal Systems*, First World

[3] In a possible interpretation, the standard Prolog inference engine could answer "unknown". In the other possible interpretation, the inference engine could answer "no".

Congress on Paraconsistency, Universiteit Gent, Ghent, Belgium, 1997. (to appear)

[4] Ávila, B.C., *Uma Abordagem Paraconsistente Baseada em Lógica Evidencial para Tratar Exceções em Sistemas de Frames com Múltipla Herança*, (A Evidential Logic-Based Paraconsistent Approach to Handle Exceptions in Multiple Inheritance Frame Systems), Ph.D. Thesis, University of São Paulo, São Paulo, Brazil, 1996, 133 pg., (in portuguese)

[5] Blair, H.A.; Subrahmanian, V.S., *Paraconsistent Logic Programming*, Proc. 7^{th} Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Springer-Verlag, Vol. 287, pp. 340-360, 1987.

[6] Blair, H.A.; Subrahmanian, V.S., *Paraconsistent Foundations for Logic Programming*, Journal of Non-Classical Logic, 5, 2, pp. 45-73, 1988.

[7] Casanova, M.A.; Giorno, F.A.C.; Furtado, A.L., *Programação em Lógica e a Linguagem Prolog*, Editora Edgard Blücher Ltda., São Paulo, Brazil, 1987. (in portuguese)

[8] Clocksin, W.F.; Mellish, C.S., *Programming in Prolog* (3^{a} Ed.), Springer-Verlag, 1987.

[9] da Costa, N.C.A., *Calculs Propositionnels pour les Systèmes Formels Inconsistants*, Compte Rendu Acad. des Sciences (Paris), 257, pp. 3790-3792, 1963.

[10] da Costa, N.C.A., *On the Theory of Inconsistent Formal Systems*, Notre Dame J. of Formal Logic 15 (1974), pp. 497-510.

[11] da Costa, N.C.A.; Subrahmanian, V.S., *Paraconsistent Logics as a Formalism for Reasoning About Inconsistent Knowledge Bases*, Artificial Intelligence in Medicine 1, pp. 167-174, Burgverlag Tecklenburg, West Germany, 1989.

[12] da Costa, N.C.A.; Henschen, L.J.; Lu, J.J.; Subrahmanian, V.S., *Automatic Theorem Proving in Paraconsistent Logics: Theory and Implementation*, Proc. 10^{th} International Conference on Automated Deduction, Lecture Notes in Computer Science, Vol. 449, pp. 72-86, 1990.

[13] da Costa, N.C.A.; Abe, J.M.; Subrahmanian, V.S., *Remarks on Annotated Logic*, Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, Vol. 37, pp. 561-570, 1991.

[14] da Costa, N.C.A.; Prado, J.P.A.; Abe, J.M.; Ávila, B.C.; Rillo, M., *ParaLog: Um Prolog Paraconsistente Baseado em Lógica Anotada*, (Paralog: A Annotated Logic-Based Paraconsistent Prolog), Coleção Documentos, Série: Lógica e Teoria da Ciência $N^{o}18$, Institute for Advanced Studies, University of São Paulo, São Paulo, Brazil, April, 1995, (in portuguese)

[15] Lloyd, J.W., *Foundations of Logic Programming*, Springer-Verlag, 1984.

[16] Ng, R.T.; Subrahmanian, V.S., *Relating Dempster-Shafer Theory to Stable Semantics*, CS-TR-2647, University of Maryland, 1991, 40 pg.

[17] Prado, J.P.A., Abe, J.M.; Ávila, B.C., *ParaNet: A Paraconsistent Multi-Agent System*, First World Congress on Paraconsistency, Universiteit Gent, Ghent, Belgium, 1997. (to appear)

[18] Shafer, G., *A Mathematical Theory of Evidence*, Princeton Univerty Press, 1976.

[19] Subrahmanian, V.S., *On the Semantics of Quantitative Logic Programs*, Proc. 4^{th} IEEE Symp. on Logic Programming, pp. 173-182, Computer Society Press, San Francisco, Sep., 1987.

[20] Subrahmanian, V.S., *Towards a Theory of Evidential Reasoning in Logic Programming*, Logic Colloquium '87, The European Summer Meeting of the Association for Symbolic Logic, Granada, Spain, July, 1987.

[21] van Emden, M.H., *Quantitative Deduction and its Fixpoint Theory*, J. of Logic Programming, 4, 1, pp. 37-53, 1986.