A Paraconsistent Logic Programming Approach for Querying Inconsistent Knowledge Bases

Sandra de Amo
Universidade Federal de Uberlândia
Uberlândia-MG-Brazil
deamo@ufu.br

Mônica S. Pais
Centro Federal de Educação Tecnológica - CEFET
Urutaí-GO-Brazil
monica@lcc.ufu.br

Abstract

When integrating data coming from multiple different sources we are faced with the possibility of inconsistency in databases. A paraconsistent approach for knowledge base integration allows keeping inconsistent information and reasoning in its presence. In this paper, we use a paraconsistent logic (**LFI1**) as the underlying logic for the specification of P-Datalog, a deductive query language for databases containing inconsistent information. We present a declarative semantics which captures the desired meaning of a recursive query executed over a database containing inconsistent facts and whose rules allow infering information from inconsistent premises. We also present a bottom-up evaluation method for P-Datalog programs based on an alternating fixpoint operator.

Introduction

The treatment of inconsistencies arising from the integration of multiple sources has been a topic increasingly studied in the past years and has become an important field of research in databases. Two basic approaches have been followed in solving the inconsistency problem in knowledge bases : belief revision (Kifer and Lozinskii 1992; Subrahmanian 1994) and paraconsistent logic (Blair and Subrahmanian 1989). The goal of the first approach is to make an inconsistent theory consistent, either by revising it or by representing it by a consistent semantics. So, the main concern of this approach is to avoid contradictions. On the other hand, the paraconsistent approach allows reasoning in the presence of inconsistency, and contradictory information can be derived or introduced without trivialization.

In this paper, we introduce P-Datalog, a logic programming language for querying databases containing inconsistencies. Our approch is paraconsistent, so inconsistencies are not rejected. Our choice was motivated by the assumption that, in most situations inconsistent information can be useful, unavoidable and even desirable. Thus, discarding inconsistent information implies *losing* information.

P-Datalog is a language which allows infering facts from a knowledge base $\mathcal{K}$ obtained by integrating local consistent sources, which may be contradictory with respect to each other. The new facts inferred from $\mathcal{K}$ are related to the facts which would be inferred in each individual source integrating $\mathcal{K}$. If an inferred fact A is **true** in the global knowledge base (the integrated one), then it would be locally inferred as true in *all* individual sources. If it is globally **controversial** then it would be locally inferred as true in *some* individual sources and as false in others. If it is globally **false**, then it would be locally inferred as false in *all* individual sources.

The syntax of P-Datalog slightly differs from Datalog$^\neg$ syntax (Abiteboul, Vianu, and Hull 1995). As in Datalog$^\neg$, P-Datalog programs are set of rules where negation may appear in the body but not in the head of rules. P-Datalog programs may also include rules with the truth-value **i** in their bodies. In fact, the main difference between P-Datalog and Datalog$^\neg$ concerns their semantics. In the classical context of Datalog$^\neg$, the rules are first-order formulas (Horn clauses with (possibly) negated litterals in the body). The answers to a Datalog$^\neg$ query constitute a set of facts where each fact has an associated truth-value **t** (true), **f** (false) or **u** (unknown). In our approach, the rules in a P-Datalog program are formulas of a *paraconsistent logic* **LFI1**. This logic was originally introduced in (Carnielli, Marcos, and de Amo 2000; Carnielli and Marcos 2001; de Amo, Carnielli, and Marcos 2002) as a logical framework to model knowledge base integration. An answer to a P-Datalog query is a set of facts, where each fact has an associated truth-value which can be **t** (true), **f** (false), **u** (unknown) or **i** (inconsistent).

In order to define the 4-valued semantics of a P-Datalog query, we take advantage of the natural 3-valued semantics of the paraconsistent logic **LFI1** (where the truth-values are **t**, **f** and **i**). In doing so, we follow the idea used to defining the well-founded semantics of Datalog$^\neg$ programs. In this classical setting, 2-valued first-order logic models are "increased" with a third truth-value **u**. In our setting, 3-valued **LFI1** models are increased with the *unknown* **u** truth-value.

The following example gives an idea of the issues treated in this paper:

Example 1 (Motivation) Suppose we have the following rule in a dishonest public contest for hiring civil servants: "if there is *some evidence* that the candidate is supported by an influential person which is not a civil servant himself and if the candidate has no debts towards the income tax services then there is some evidence that this candidate will have the job." The intuitive meaning behind the expression

there is some evidence is that this information is supported by *at least* one source, even though some sources may affirm the contrary.

We can translate the above story in the following P-Datalog program P_{job}:

job(x) $\leftarrow \sim$*owe(x), supportedby(x,y),* $\sim$*job(y)*

In the paraconsistent logic **LFI1**, an atomic formula $R(\vec{x})$ is *verified* if its truth-value is **t** or **i** (in a paraconsistent approach, inconsistencies are not rejected). So, in the P_{job} program, literals *supportedby(x,y)* (in the clause body) and *job(x)* (in the clause head) represent information that is true or controvertial. On the other hand, the literals $\sim$ *owe(x)* and $\sim$ *job(y)* represent *sure* negative information: all sources of information affirm the fact that x has no records in the income tax services files concerning debts and that y is not a civil servant. Let us suppose that we have the following facts stored in the integrated knowledge base:

I = {$\circ$ *supportedby(charles,joseph),*
$\circ$ *supportedby(joseph,charles),*
$\circ$ *supportedby(paul,james),*
$\bullet$ *supportedby(john,kevin),*
$\circ$ *supportedby(james,kevin),*
$\bullet$ *owe(james)*}

The symbols $\circ$ and $\bullet$ attached to each fact in the knowledge base mean that the fact is *sure* and *controversial*, respectively. We notice that the facts stored in the knowledge base must be explicitly declared as sure or controversials (by attaching these symbols $\circ$ and $\bullet$). Following the closed-world assumption, facts that are not in the knowledge base are considered false.

We now show a 4-valued model **J** of P_{job} which includes the facts of the knowledge base **I**, that is, **J** agrees with **I** on the values of *owe* and *supportedby* atoms. This 4-valued model **J** contains the facts $job(x)$ which correspond to the answer to the query "For which people is there some evidence that they will get the job "? As we will show later (see Example 5), this model **J** is the well-founded semantics of P_{job} on input **I**. The values of the *job* atoms in the derived knowledge base **J** are the following:

surely true	*job(paul)*	**t**
controversial	*job(john)*	**i**
surely false	*job(kevin), job(james)*	**f**
unknown	*job(charles), job(joseph)*	**u**

This model asserts that James *surely does not get the job* because there is some evidence that he owes to the taxation office, and from this fact we can infer that Paul *surely gets the job*. Indeed, Paul does not owe any tax return and he is supported by James who is not a civil servant. It also can be deduced that Kevin *definitely does not succeed in getting the job*, because nobody supports him. In John's case, he does not owe the taxation office but it is controversial that he is supported by Kevin, who is not a public servant himself. Thus, it is *controversial that John gets the job*. Notice that in this case, a *controversial information was inferred*. On the other hand, it is unknown that Charles and Joseph succeed in the public contest. They fulfill almost all the requirements: they do not have debts, they have the support of an influential person but they depend on each other: Charles supports Joseph and Joseph supports Charles. The only chance for Charles getting the job is if Joseph (his only support) does not get it. And vice-versa, the only chance for Joseph getting the job is if Charles (his only support) does not get it. Therefore it is not possible to infer which one will get the job: either Charles or Joseph. This means that this information is *unknown*: we *cannot infer* the existence or nonexistence of any source supporting it.

So, the answer to our query : *"For which people is there some evidence that they will get the job "?* is Paul and John. Besides, we know that Paul surely gets the job, but in John's case, we only can affirm that it is controvertial that he will get the job. That means: (1) From the point of view of sources in $\mathcal{K}^+$ (those affirming that John *is* supported by Kevin), John gets the job, (2) From the point of view of sources in $\mathcal{K}^-$ (those affirming that John *is not* supported by Kevin), John does not get the job. So, in the integrated knowledge base $\mathcal{K}$, this derived information is controversial.

Differently from some approaches treating paraconsistent query languages (Pereira and Alferes 1992; Sakama 1992; Blair and Subrahmanian 1989; Subrahmanian 1994), our well-founded semantics is a natural extension of the well-founded semantics for Datalog$^\neg$ programs (Przymusinski 1990). In this paper, we also present a bottom-up evaluation procedure for computing the well-founded semantics based on the alternating fixpoint computation introduced in (Van Gelder 1989).

The paper is organized as follows: Firstly, we briefly describe the basic notions of the logic **LFI1**, then we introduce P-Datalog programs and generalize the notion of database instance to allow the storage of inconsistent information in our knowledge bases. Next, we describe the well-founded semantics of a P-Datalog program. Finally, we present a bottom-up method for evaluating P-Datalog programs and briefly discuss its implementation. Due to lack of space, the proofs of the results in this paper are omitted.

LFI1 : A 3-valued Paraconsistent Logic

In this section we briefly describe the syntax and semantics of **LFI1** (Logic of Formal Inconsistency). A detailed presentation can be found in (Carnielli, Marcos, and de Amo 2000). The semantics of a P-Datalog program is based on the semantics of **LFI1**. Even if P-Datalog programs constitute a small fragment of the set of **LFI1** formulas (we only consider Prolog-like Horn clauses), inference in P-Datalog is based on the paraconsistent framework of **LFI1**.

Let **R** be a finite signature without functional symbols and **Var** a set of variables symbols. We assume that formulas of **LFI1** are defined in the usual way, as in the classical first-order logic setting, with the addition of a new symbol $\bullet$ (read "it is inconsistent"). So, a formula of **LFI1** is defined inductively by the following statements (and only by them) :

- If R is a predicate symbol of arity k and $x_1, ..., x_k$ are constants or variables, then $R(x_1, ..., x_k)$ and $x_1 = x_2$ are atomic formulas or atoms. The former is called a *relational* atom and the latter an *equality* atom.

• If F, G are formulas and x is a variable then $F \vee G$, $\neg F$, $\forall xF$, $\exists xF$ and $\bullet F$ are formulas.

A *sentence* is a formula without free variables. A *fact* is a relational atom without free variables. We denote by $\mathcal{F}$ the set of facts.

Definition 1 Let **R** be a finite signature. An *interpretation* over **R** is an application $\delta : \mathcal{F} \rightarrow \{$**f** (false), **t** (true), **i** (inconsistent)$\}$.

An interpretation of facts can be extended to the propositional sentences in a natural way by using the connective matrices described in the tables below. The connective $\wedge$ is derived from the connectives $\neg$, $\vee$: A $\wedge$ B = $\neg(\neg A \vee \neg B)$ and the connective $\rightarrow$ is derived from $\neg, \vee, \bullet$: $A \rightarrow B \equiv \neg(A \vee \bullet A) \vee B$.

$\vee$	**t**	**i**	**f**
t	**t**	**t**	**t**
i	**t**	**i**	**i**
f	**t**	**i**	**f**

	$\neg$	$\bullet$
t	**f**	**f**
i	**i**	**t**
f	**t**	**f**

$\wedge$	**t**	**i**	**f**
t	**t**	**i**	**f**
i	**i**	**i**	**f**
f	**f**	**f**	**f**

$\rightarrow$	**t**	**i**	**f**
t	**t**	**i**	**f**
i	**t**	**i**	**f**
f	**t**	**t**	**t**

The extension of δ to the quantified sentences is obtained by means of the concept of *distribution quantifiers*. Basically, this concept translates our basic intuition that an universal quantifier should work as a kind of unbounded conjunction and an existential quantifier as an unbounded disjunction. Due to lack of space, we do not present this extension here. For more details, see (Carnielli, Marcos, and de Amo 2000). In fact, the formulas we will deal with in the next sections are Horn clauses, which are interpreted over a finite Herbrand Universe. So, the universal quantifiers appearing in the clauses can be viewed as a bounded conjunction.

We denote by **Dom** the Herbrand Universe of **R** (the constant symbols of **R**). In fact, we are supposing that the universe domain of any interpretation δ is **Dom** (thus, δ interprets the constant symbols by themselves). A *valuation* is an application v : **Var** $\rightarrow$ **Dom**.

Definition 2 Let $F(x_1, ..., x_n)$ be a formula of **LFI1** with free variables x_1,...,x_n, v a valuation and δ an interpretation. We say that (δ, v) *satisfies* $F(x_1, ..., x_n)$ (denoted by $(\delta, v) \models F(x_1, ..., x_n)$) iff $\delta(F[v(x_1), ..., v(x_n)/x_1, ..., x_n])$ is **t** or **i**. If (δ,v) $\models F$ for each valuation v, we say that δ is a *model* of F (denoted $\delta \models F$). We also say that F is *verified* or *satisfied* by δ.

The Query Language P-Datalog

In this section we use the logical formalism **LFI1** to generalize the notion of database instance to allow the storage of inconsistent information in our databases. We also introduce the query language P-Datalog which is designed to query databases containing inconsistent information. We assume that the reader is familiar with traditional database terminology (Abiteboul, Vianu, and Hull 1995). In what follows, we denote by $R(\vec{u})$ the formula $R(u_1, ..., u_k)$, where $u_1, ..., u_k$ are variables and we denote by $R(\vec{a})$ the fact $R(a_1, ..., a_k)$, where $a_1, ..., a_k$ are constants (k = arity of R).

Definition 3 (Paraconsistent Databases) Let **R** be a database schema, i.e., a set of relation names (or predicate names). A *3-valued instance* over **R** (or a *paraconsistent database*) is an interpretation **I** such that for each R $\in$ **R** the set $\mathbf{I}_{\mathrm{R}}$ = $\{\vec{a}$: **I**(R($\vec{a}$)) = **t** or **I**(R($\vec{a}$)) = **i**$\}$ is finite. So, an instance over **R** can be viewed as a finite set of facts over **R**, having truth-values **t** or **i**. The facts which are not in the instance **I** have truth-value **f**. A fact R($\vec{a}$) such that **I**(R($\vec{a}$)) = **i** is intended to be *controversial*, i.e. there may be evidence in favor of R($\vec{a}$) and also evidence against R($\vec{a}$). On the other hand, if **I**(R($\vec{a}$)) = **t**, R($\vec{a}$) is intended to be a safe information.

P-Datalog is an extension of Datalog$^{\neg}$ (Abiteboul, Vianu, and Hull 1995). This well-known deductive query language uses the classical first order logic as its underlying logic, and a Datalog$^{\neg}$ query applies over a classical database instance, i.e. a finite first-order interpretation. Rather than classical first-order logic, P-Datalog uses the paraconsistent logic **LFI1** as its underlying logic, and P-Datalog queries apply over paraconsistent databases. P-Datalog programs are first-order Horn clauses as in Datalog$^{\neg}$ programs, i.e. first-order clauses with positive and negative literals in their bodies. Negation in P-Datalog (as well as in Datalog$^{\neg}$) is understood as *negation by default*. We will denote this negation by the symbol $\sim$. The negation $\neg$ used in **LFI1** is the *weak negation*. The relationship between these two negations is given by: $\sim A = \neg A \wedge \neg \bullet A$ and $\neg A = \sim A \vee \bullet A$. The intuitive meaning of default and weak negations is the following: (1) the ground formula $\sim R(\vec{a})$ is verified by a paraconsistent database **I** if the fact $R(\vec{a})$ is not in **I** (i.e., $R(\vec{a})$ is surely false); (2) the ground formula $\neg R(\vec{a})$ is verified by **I** if the fact $R(\vec{a})$ is in **I** as controversial or if it is not in **I** (i.e., the only thing we can affirm is that $R(\vec{a})$ is not surely true).

Definition 4 (P-Datalog Programs) A P-Datalog *program* is a finite set of rules $L \leftarrow L_1, ..., L_n$, where L is a literal of the form $R(\vec{u})$, and L_i are literals of the form: $R(\vec{u})$ or $\sim R(\vec{u})$. R is a relation name and $\vec{u}$ is a free tuple of appropriate arity. The literal L is called the *head* of the rule. The literals $L_1, ..., L_n$ are called the *body*. One requires also that each variable occurring in the head of the rule must occur in at least one of the free tuples in the body.

We denote by $sch(P)$ the set of relations (predicates) appearing in P, by $adom(P)$ the set of constants appearing in P and by **B**(P) all facts of the form $R(\vec{a})$ where $R \in sch(P)$ and $\vec{a}$ is a tuple of constants in $adom(P)$ (the Herbrand Base of P). The set of relations which appear in the head of rules are called the *intensional relations* and is denoted by $idb(P)$. The set of those appearing only in the body of rules are called *extensional relations* and is denoted by $edb(P)$. In fact, the set $edb(P)$ contains only the relations R where $R(\vec{a})$ is a fact in **I**.

Definition 5 (P-Datalog Query) A P-Datalog *query* is a pair (P,$Q(u_1, ..., u_n)$) where P is a P-Datalog program, $Q \in idb(P)$ and $u_1, ..., u_n$ are variables or constants in $adom(P)$ (n is the *arity* of the relation Q).

Example 2 (Running Example) Let us consider the same situation presented in Example 1. The rule P_{job} and the 3-valued instance **I** described in that example constitutes a P-Datalog program P where $sch(P)$ ={*supportedby, job, owe*}, $adom(P)$ = {*charles, john, james, joseph, paul, kevin*} and **B**(P) = {*owe(charles), owe(joseph), supportedby(charles,joseph), supportedby(joseph,charles), ...*}. The intensional and extensional schemas are $edb(P)$ = {*supportedby, owe*}, $idb(P)$ = {*job*}. The pair (P_{job},$job(x)$) is a P-Datalog query ("For which people there is some evidence that they will get the job "?). The pair (P_{job},$job(Kevin)$) corresponds to the boolean query "Is there some evidence that Kevin may get the job" ?

Answering P-Datalog Queries

In this section we introduce the well-founded semantics for P-Datalog programs. The well-founded semantics of a P-Datalog program P is designed to capture the natural semantics of queries $(P, Q(u_1, ..., u_n))$ where $Q \in idb(P)$, that is, what we expect to be their answers. Our approach is a natural extension of the well-founded semantics for Datalog$^\neg$ (Przymusinski 1990). Our definition of a P-Datalog query makes use of *4-valued instances*, in which facts may assume one of the four truth-values in the set ***Val*** = {*true*(**t**), *false*(**f**), *inconsistent*(**i**) *unknown*(**u**)}. We assume that the reader is familiar with the notions of lattices, lattice operators, monotonicity and continuity, fixpoints, etc. For details, see (Lloyd 1993).

4-valued Models

Let us consider the complete lattice (***Val***, $\leqslant$), where **f** $\leqslant$ **u** $\leqslant$ **i** $\leqslant$ **t**.

Definition 6 Let P be a P-Datalog program. A ***4-valued instance*** **I** over $sch(P)$ is an application $\mathbf{I} : \mathbf{B}(P) \longrightarrow$ {**t, f, u, i**}.

The answer of a program P is a special 4-valued instance which corresponds to the well-founded semantics of P. The main goal of this section is to define this particular instance.

There is a natural ordering $\preccurlyeq$ among 4-valued instances over $sch(P)$, defined by: **I** $\preccurlyeq$ **J** iff for each $A \in$ **B**(P), **I**(A) $\leqslant$ **J**(A).

The set of 4-valued instances of a P-Datalog program P is denoted by 4-$Inst_P$. It is easy to verify that (4-$Inst_P$, $\preccurlyeq$) constitutes a complete lattice. We denote by $\top$ the *maximal* 4-valued instance (where all facts have truth-value **t**) and by $\bot$ the minimal 4-valued instance (where all facts have truth-value **f**).

We also represent a 4-valued instance by listing the positive, inconsistent and negative facts, and omitting the unknown ones.

Example 3 (4-valued instance) Let **J** be a 4-valued instance, where **J**(p)=**t**, **J**(q)=**t**, **J**(r)=**u** and **J**(s)=**f**. **J** can be written as **J** = $\{\circ p, \circ q, \sim s\}$. Let **J**$'$ = $\{\circ p, \circ q, \bullet s\}$. Then **J** $\preccurlyeq$ **J**$'$.

We extend the 3-valued connective matrices of **LFI1**, to the following 4-valued matrices:

	$\sim$
t	**f**
i	**f**
u	**u**
f	**t**

$\vee$	**t**	**i**	**u**	**f**
t	**t**	**t**	**t**	**t**
i	**t**	**i**	**i**	**i**
u	**t**	**i**	**u**	**u**
f	**t**	**i**	**u**	**f**

$\wedge$	**t**	**i**	**u**	**f**
t	**t**	**i**	**u**	**f**
i	**i**	**i**	**u**	**f**
u	**u**	**u**	**u**	**f**
f	**f**	**f**	**f**	**f**

$\rightarrow$	**t**	**i**	**u**	**f**
t	**t**	**f**	**f**	**f**
i	**t**	**i**	**f**	**f**
u	**t**	**t**	**t**	**f**
f	**t**	**t**	**t**	**t**

It is important to note the difference between the first line in the matrix of the $\rightarrow$ connective above and the same line of its counterpart in **LFI1**: In P-Datalog, the truth-value of **t** $\rightarrow$ **i** is **f** and not **i** as in **LFI1**. Indeed, in P-Datalog we cannot derive a controversial fact from a surely true one.

If F is the body of a P-Datalog rule and **J** is a 4-valued instance, we denote by **J**(F) the truth-value associated to F according to the matrices for the connectives given above.

Definition 7 Let P be a P-Datalog program. An *instantiated rule* of P is a rule where all variables are replaced by constants in $adom(P)$. We denote by $ground(P)$ the set of instantiated rules of P. A 4-valued instance **J** over $sch(P)$ *satisfies* a boolean combination α of atoms in $\mathbf{B}(P)$ iff **J**(α) $\in$ {**t**, **i**}. A *4-valued model* of P is a 4-valued instance **J** over $sch(P)$ satisfying each rule in $ground(P)$, i.e., the truth-value of each rule in $ground(P)$ is **t** or **i**. So, **J** is a model of the **LFI1** formulas corresponding to the rules of P (see definition 2). A 4-valued model M is *minimal* iff for all $M' \subset M$, M' is not a model.

Extended P-Datalog programs

The well-founded semantics is based on the notion of stable models. Stable models are usually defined as fixpoint of an immediate consequence operator. Following the same idea underlying the definition of 3-stable models in (Przymusinski 1990), we introduce the notion of *extended P-Datalog programs*. We will see that for such programs we can define an immediate consequence operator which is monotonic and so, has a unique least fixpoint.

Definition 8 An *extended P-Datalog program* is a P-Datalog program where (1) negative facts $\sim A$ do not appear in the body of rules and (2) truth-values **t, f, u** and **i** may occur as literals in the body of rules.

Next we define the *immediate consequence operator 4-T_P associated to an extended program P*.

Definition 9 Let P be an extended P-Datalog program. The *immediate consequence operator 4-T_P associated to P* is a mapping 4-T_P : 4-$Inst_P$ $\rightarrow$ 4-$Inst_P$ defined as follows. Let **J** be a 4-valued instance and $A \in \mathbf{B}(P)$, then

$$\text{4-}T_P(\mathbf{J})(A) = \begin{cases} \max\{\mathbf{J}(F_k)\} & \text{if there are rules } A \leftarrow F_k \text{ in } \textit{ground(P)}, 0 \leqslant k \leqslant n. \\ \mathbf{f} & \text{otherwise} \end{cases}$$

The following lemma says that the immediate consequence operator for extended programs have a least fixpoint.

Lemma 1 Let P be an extended P-Datalog program. Then 4-T_P is monotonic and the sequence $\{4\text{-}T_P^i(\bot)\}_{i\geqslant 0}$ is increasing and converges to the least fixpoint of 4-T_P. Besides, P has a unique 4-valued minimal model that equals the least fixpoint of 4-Tp (denoted by $P(\bot)$).

4-stable Models

According to (Przymusinski 1990), the semantics of a Datalog$^\neg$ program P is an *appropriate* 3-valued model **I** of P. We extend this idea to P-Datalog programs and introduce the 4-stable Models, a class of *special* models. The semantics of a P-Datalog query will be the intersection of all 4-stable models.

Let P be a P-Datalog program and **I** a paraconsistent database (3-valued) instance. We denote by $P_{\mathbf{I}}$ the program obtained from P by adding to P unit clauses $A \leftarrow$ for each A such that **I**(A) = **t**, and a clause $A \leftarrow$ **i** for each A such that **I**(A) = **i**. From now on, we suppose that our programs include these clauses corresponding to the input facts of **I**. Thus, as mentioned in the introduction, P-Datalog programs may include rules with the truth-value **i** in their bodies.

Let **I** be a 4-valued instance over $sch(P)$. The *positivized ground version* of P according to **I** (denoted $pg(P,\mathbf{I})$), is the P-Datalog program obtained from $ground(P)$ by replacing each negative literal $\sim A$ by **I**($\sim A$) (i.e, by its respective truth value: **t, f, u, i**). So, $pg(P,\mathbf{I})$ is an extended P-Datalog program, i.e., a program without negation. By lemma 1, the least fixpoint $pg(P,\mathbf{I})(\bot)$ of its immediate consequence operator exists. It contains all facts that are inferred from P and **I**, by assuming the values for the negative premises as given by **I**.

We denote $pg(P,\mathbf{I})(\bot)$ by $conseq_P(\mathbf{I})$, i.e. $conseq_P(\mathbf{I})$ is the least fixpoint of the extended P-Datalog program $pg(P,\mathbf{I})$.

Definition 10 Let P be a P-Datalog program. A 4-valued instance **I** over $sch(P)$ is a *4-stable model* of P iff $conseq_P(\mathbf{I}) = \mathbf{I}$.

The following example illustrates the notion of 4-stable model:

Example 4 (4-stable model) Consider the P-Datalog program P_{job} given in the example 1 and the input instance **J**:

surely true	**t**	*supportedby(charles,joseph), supportedby(joseph,charles), supportedby(paul,james), supportedby(james,kevin),job(paul)*
surely false	**f**	*job(james), job(kevin)*
controversial	**i**	*supportedby(john,kevin), job(john), owe(james)*
unknown	**u**	*job(charles), job(joseph)*

Let us check that **J** is a 4-stable model of P_{job}. For this, we have to compute $conseq(\mathbf{J})$ and show that $conseq(\mathbf{J})$ = **J**. The program $P' = pg(P,\mathbf{J})$ is

job(charles) ← **t**,*supportedby(charles,joseph)*, **u**
job(joseph) ← **t**,*supportedby(joseph,charles)*, **u**
...
supportedby(paul,james) ←
supportedby(charles,joseph) ←
supportedby(john,kevin)← **i**
...

The minimal 4-valued model of P' is obtained by iterating 4-$T_P(\bot)$ up to a fixpoint. The first execution of 4-T_P yields 4-$T^1_{P'}(\bot) = \{\sim$ *job(charles)*, $\sim$ *job(joseph)*, $\sim$ *job(paul)*, $\sim$ *job(john)*, $\sim$ *job(james)*, $\sim$ *job(kevin)*$\}$. We can verify that 4-$T^2_{P'}(\bot)$= 4-$T^3_{P'}(\bot) = \{\circ$ *job(paul)*, $\bullet$ *job(john)*, $\sim$ *job(james)*, $\sim$ *job(kevin)* $\}$. Thus $conseq_P(\mathbf{J}) = \mathbf{J}$ and so, **J** is a 4-*stable model* of P. We notice that the instance **J** coincides with **I** for the atoms $supportedby$ and owe.

Well-founded Semantics

P-Datalog programs generally may have several 4-stable models, and each P-Datalog program has at least one 4-stable model (see theorem 4). Then it is reasonable to say that the desired answer to a P-Datalog query consists of the positive, *inconsistent* and negative facts belonging to *all* 4-stable models of the program.

Definition 11 Let P be a P-Datalog program. The *well-founded semantics* of P is a 4-valued instance consisting of the positive, inconsistent and negative facts belonging to all 4-stable models of P. This semantics is denoted by P^{4wf}.

Bottom-up Evaluation of P-Datalog Queries

The previous description of the well-founded semantics, although effective, is inefficient. It involves checking all possible 4-valued instances of a program, determining which are 4-stable models, and then taking their intersection.

A much simpler method is based on an *alternating fixpoint* computation (Van Gelder 1989), that converges to the well-founded semantics. The idea of the method is as follows. We define an alternating sequence $\{\mathbf{I}_i\}_{i\geqslant 0}$ of 4-valued instances that are underestimates and overestimates of the facts known in every 4-stable model of P. The sequence is defined as follows: $\mathbf{I}_0 = \bot$ and $\mathbf{I}_{i+1} = conseq_P(\mathbf{I}_i)$, for $i \geqslant 0$.

Theorem 1 The operator $conseq_P$ is antimonotonic. That is, if $\mathbf{I} \preccurlyeq \mathbf{J}$ then $conseq_P(\mathbf{J}) \preccurlyeq conseq_P(\mathbf{I})$.

>From this theorem, we can easily see that:

(*) $\mathbf{I}_0 \preccurlyeq \mathbf{I}_2 \preccurlyeq \mathbf{I}_4 \preccurlyeq \cdots \preccurlyeq \mathbf{I}_{2i} \preccurlyeq \mathbf{I}_{2i+2} \preccurlyeq \ldots$
$\cdots \preccurlyeq \mathbf{I}_{2i+1} \preccurlyeq \mathbf{I}_{2i-1} \preccurlyeq \cdots \preccurlyeq \mathbf{I}_5 \preccurlyeq \mathbf{I}_3 \preccurlyeq \mathbf{I}_1$

Thus the even subsequence is increasing and the odd one is decreasing. Because there are finitely many 4-valued instances relatively to a given program P, each of these sequences becomes constant at some point: $\mathbf{I}_{2k_0} = \mathbf{I}_{2k_0+2}$ = ... $\mathbf{I}_{2k_0+4}$ = and $\mathbf{I}_{2j_0+1} = \mathbf{I}_{2j_0+3} = \mathbf{I}_{2j_0+5}$ = ..., for some $k_0 \geqslant 0$ and some $j_0 \geqslant 0$.

Let $\mathbf{I}_*$ be the *least upper bound* of the increasing sequence: $\mathbf{I}_* = lub\{\mathbf{I}_{2i}\}_{i\geqslant 0}$, and let $\mathbf{I}^*$ be the *greatest lower bound* of the decreasing sequence: $\mathbf{I}^* = glb\{\mathbf{I}_{2i+1}\}_{i\geqslant 0}$. From **(*)**, it follows that $\mathbf{I}_* \preccurlyeq \mathbf{I}^*$.

Theorem 2 Let $\mathbf{I}$ be a 4-valued instance of a P-Datalog program. Then $conseq_P(\mathbf{I}_*) = \mathbf{I}^*$ and $conseq_P(\mathbf{I}^*) = \mathbf{I}_*$.

>From the 4-valued instances $\mathbf{I}_*$ and $\mathbf{I}^*$ we can define the 4-valued instance $\mathbf{I}^*_*$ which coincides with the well-founded semantics of a P-Datalog program, as we will see in Theorem 4.

Definition 12 Let $\mathbf{I}^*_*$ be a 4-valued instance of a P-Datalog program P, consisting of the facts known in both $\mathbf{I}_*$ and $\mathbf{I}^*$, that is:

$$\mathbf{I}^*_*(A) = \begin{cases} \mathbf{t} & \mathbf{I}_*(A) = \mathbf{I}^*(A)\text{=}\mathbf{t} \\ \mathbf{i} & \mathbf{I}_*(A) = \mathbf{I}^*(A)\text{=}\mathbf{i} \\ \mathbf{f} & \mathbf{I}_*(A) = \mathbf{I}^*(A)\text{=}\mathbf{f} \\ \mathbf{u} & \text{otherwise} \end{cases}$$

Theorem 3 Let $\mathbf{I}$ be a 4-valued instance of a P-Datalog program P. Then $\mathbf{I}_* \preccurlyeq \mathbf{I}^*_* \preccurlyeq \mathbf{I}^*$.

The fixpoint construction yields the well-founded semantics for P-Datalog programs. The following theorem is the main result of this paper. It shows that each P-Datalog program has at least one 4-stable model ($\mathbf{I}^*_*$) and that the well-founded semantics coincides with $\mathbf{I}^*_*$.

Theorem 4 For each P-Datalog program P:
- $\triangleright$ $\mathbf{I}^*_*$ is a 4-stable model of P.
- $\triangleright$ $P^{4wf} = \mathbf{I}^*_*$.

We illustrate this computation in our running example:

Example 5 ($\mathbf{I}^*_*$ computation) Consider again the program P_{job} and the database instance $\mathbf{I}$ of the running example 1. Note that for $\mathbf{I_0}$ the value of all facts is **f**, and for each $j \geqslant 1$, $\mathbf{I}_j$ agrees with the input $\mathbf{I}$ on the predicates *supportedby* and *owe*. Therefore we only show the inferred job-facts:

$\mathbf{I}_0 =$ {$\sim$ *job(charles)*, $\sim$ *job(james)*, $\sim$ *job(john)*, $\sim$ *job(joseph)*, $\sim$ *job(kevin)*, $\sim$ *job(paul)*}.
$\mathbf{I}_1 =$ {$\circ$ *job(charles)*, $\circ$ *job(james)*, $\bullet$ *job(john)*, $\circ$ *job(joseph)*, $\sim$ *job(kevin)*, $\circ$ *job(paul)*}.
$\mathbf{I}_2 =$ {$\sim$ *job(charles)*, $\sim$ *job(james)*, $\bullet$ *job(john)*, $\sim$ *job(joseph)*, $\sim$ *job(kevin)*, $\sim$ *job(paul)*}.
$\mathbf{I}_3 =$ {$\circ$ *job(charles)*, $\sim$ *job(james)*, $\bullet$ *job(john)*, $\circ$ *job(joseph)*, $\sim$ *job(kevin)*, $\circ$ *job(paul)*}
$\mathbf{I}_4 =$ {$\sim$ *job(charles)*, $\sim$ *job(james)*, $\bullet$ *job(john)*, $\sim$ *job(joseph)*, $\sim$ *job(kevin)*, $\circ$ *job(paul)*}.
$\mathbf{I}_5 =$ {$\circ$ *job(charles)*, $\sim$ *job(james)*, $\bullet$ *job(john)*, $\circ$ *job(joseph)*, $\sim$ *job(kevin)*, $\circ$ *job(paul)*}
$\mathbf{I}_6 =$ {$\sim$ *job(charles)*, $\sim$ *job(james)*, $\bullet$ *job(john)*, $\sim$ *job(joseph)*, $\sim$ *job(kevin)*, $\circ$ *job(paul)*}.

$\mathbf{I}_* = \mathbf{I}_6$ and $\mathbf{I}^* = \mathbf{I}_5$. Thus $\mathbf{I}^*_* =$ {$\sim$ *job(james)*, $\bullet$ *job(john)*, $\sim$ *job(kevin)*, $\circ$ *job(paul)*}. This is exactly the natural answer for P_{job} we have informally discussed in example 1.

Implementation Issues

We have decided to implement the P-Datalog prover as a separate system and further to integrate it in a relational database system. For now, the P-Datalog prover is a helpful tool for validating the well-founded semantics we have proposed for P-Datalog programs. It has been implemented in Objective Caml (Leroy 2002). The OCaml compiler generates code whose executing time is comparable to a C/C^{++} code, and it includes libraries for several platforms. Those characteristics and also the functional programming qualities allowed us to focus on the difficulties of our application and to develop a preliminary succinct solution.

Acknowledgments

Mônica S.Pais was supported by an individual grant from CAPES-Brazil.

References

Abiteboul, S.; Vianu, V.; and Hull, R. 1995. *Foundations of Databases*. Addison-Wesley.

Blair, H., and Subrahmanian, V. 1989. Paraconsistent logic programming. *Theoretical Computer Science* 135–154.

Carnielli, W. A., and Marcos, J. 2001. A taxonomy of C-systems. In *Proceedings of the II World Congress on Paraconsistency (WCP2000)*, 1–94.

Carnielli, W. A.; Marcos, J.; and de Amo, S. 2000. Formal inconsistency and evolutionary databases. *Logic and Logical Philosophy* 8:115–152.

de Amo, S.; Carnielli, W. A.; and Marcos, J. 2002. A logical framework for integration inconsistent information in multiple databases. *FOIKS 2002, LNCS* 2284:67–84.

Kifer, M., and Lozinskii, E. 1992. A logic for reasoning with inconsistency. *Journal of Automated Reasoning* 179–215.

Leroy, X. 2002. The objective caml system - release 3.06. Documentation and user´s manual.

Lloyd, J. 1993. *Foundations of Logic Programming*. Springer-Verlag.

Pereira, L. M., and Alferes, J. J. 1992. Well founded semantics for logic programs with explicit negation. In *European Conference on Artificial Intelligence*, 102–106.

Przymusinski, T. C. 1990. Well-founded semantics coincides with three-valued stable semantics. *Fundamentae Informaticae, XIII* 445–463.

Sakama, C. 1992. Extended well-founded semantics for paraconsistent logic programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 592–599.

Subrahmanian, V. S. 1994. Amalgamating knowledge bases. *ACM Transaction on Database Systems* 19(2):291–331.

Van Gelder, A. 1989. The alternating fixpoint of logic programs with negation. In *Proceedings of the eighth ACM Symposium on Principles of Database Systems-ACM PODS'89*, 1–10.