A paraconsistent logic programming approach for querying inconsistent databases

Sandra de Amo a,*, Mônica Sakuray Pais b,1

a Faculdade de Computação, Universidade Federal de Uberlândia, Computer Science Department, Av. João Naves de Ávila, 2121, Bloco B, Sala 1B77, Campus Santa Mônica, 38400-902 Uberlândia, MG, Brazil
b Departamento de Informática, CEFET-Uriú, Brazil

Received 30 June 2005; received in revised form 16 March 2006; accepted 21 September 2006
Available online 16 November 2006

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 inferring information from inconsistent premises. We also present a bottom-up evaluation method for P-Datalog programs based on an alternating fixpoint operator.

Keywords: Inconsistent information; Logic programming; Paraconsistent logic; Deductive databases; Query languages

1. 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. Roughly, there are two approaches to handle the inconsistency problem in knowledge bases: belief revision [23,29] and paraconsistent logic [11]. 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. 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. Besides these two approaches, there are “hybrid” approaches based on formalisms which associates degrees of belief, reliability or uncertainty to the source knowledge bases [15,9]. In this paper, we introduce P-Datalog, a logic programming language for querying databases containing inconsistencies. Our approach is paraconsistent: inconsistencies are not rejected. Following the arguments presented in [21], our choice is motivated by the assumption that, in most situations, inconsistent information can be useful, unavoidable and even desirable. Thus, in most situations, the goal of retaining all available information is quite legitimate since discarding inconsistent information would imply losing information.

P-Datalog is a language which allows inferring facts from a database \( K \) obtained by integrating local consistent sources which may be globally inconsistent: they may be contradictory with respect to each other. In this paper, we focus our attention in the integrated database, which possibly contains inconsistent information. We are not interested in the integration process, that is, which information is kept in the source databases or how the integrated database is built. These issues have been treated in a previous paper [18], where we proposed a system based on a deductive proof mechanism to integrate information coming from multiple databases. The intuitive idea of the semantics of facts stored in the integrated database \( \mathcal{K} \) is the following: a fact \( A \in \mathcal{K} \) is true (resp. false) if it is true (resp. false) in all source databases. It is inconsistent if it is true in some local source and false in some other. The language P-Datalog allows to infer new facts from the facts stored in the integrated database \( \mathcal{K} \). These new facts inferred are related to the facts which would be inferred in each individual consistent source. If an inferred fact \( A \) is true in the global database (the integrated one), then it would be locally inferred as true in all individual sources: it is a safe information and it is surely true. If it is globally inconsistent then it would be locally inferred as true in some individual sources and as false in others: there is some evidence of \( A \) in \( \mathcal{K} \). 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 [1]. As in Datalog\(^{\neg} \), P-Datalog programs are normal logic programs [25] (the same as the general logic programs of [31]): a set of rules where negation may appear in the body but not in the head of rules. In fact, the main difference between P-Datalog and Datalog\(^{\neg} \) concerns their semantics. 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 of a P-Datalog program are Horn clauses like in Datalog\(^{\neg} \), but their semantics is related to the paraconsistent logic LF11 which was originally introduced in [17,18] as a logical framework to model database integration.\(^{3} \) An answer to a P-Datalog query is a set of facts, where each

\(^{2} \) This paper is a full version of the conference paper [19].

\(^{3} \) In fact, LF11 belongs to a family of paraconsistent logics called Logics of Formal Inconsistency (LFI). An extensive and comprehensive presentation of LF11 and the family of paraconsistent logics LFI can be found in [17] and [14], respectively.
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 well-founded 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 \( i \)). We adapt the ideas of [27,30] originally presented in the context of Datalog\(^\neg\) programs. In this classical setting, 2-valued first-order logic models are augmented with a third truth-value \( u \) (unknown). In our setting, 3-valued LFI1 models are augmented with the truth-value \( u \) (unknown) as well. Thus, the 4-valued semantics for P-Datalog programs we propose is a natural extension of the 3-valued well-founded semantics of Datalog\(^\neg\) programs.

The following example illustrates our approach.

**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 get 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 story above in the following P-Datalog program \( P_{\text{job}} \):

\[
\text{job}(x) \leftarrow \neg \text{owe}(x), \text{supportedby}(x,y), \neg \text{job}(y)
\]

Notice that the unique rule of \( P_{\text{job}} \) follows the syntax of Datalog\(^\neg\). However, its semantics is evaluated according to the semantic laws of the paraconsistent logic LFI1. In this logic, an atomic formula \( R(\vec{x}) \) is verified if its truth-value is \( t \) or \( i \) (in a paraconsistent approach, inconsistencies are not rejected). Thus in the \( P_{\text{job}} \) program, literals \( \text{supportedby}(x,y) \) (in the body) and \( \text{job}(x) \) (in the head) represent information that are true or inconsistent. On the other hand, the literals \( \neg \text{owe}(x) \) and \( \neg \text{job}(y) \) represent negative information. Intuitively, this means that all information sources 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 database:

\[
\mathcal{H} = \{ \circ \text{supportedby}(\text{charles}, \text{joseph}), \circ \text{supportedby}(\text{joseph}, \text{charles}),
\circ \text{supportedby}(\text{paul}, \text{james}), \bullet \text{supportedby}(\text{john}, \text{kevin}),
\circ \text{supportedby}(\text{james}, \text{kevin}), \bullet \text{owe}(\text{james}) \}
\]

The symbols \( \circ \) and \( \bullet \) attached to each fact in the database mean that the fact is sure and controversial, respectively. We notice that the facts stored in the database must be explicitly declared as sure or controversial (by attaching these symbols \( \circ \) and \( \bullet \)). Following the closed-world assumption, facts that are not in the database are considered false. In order to better understand the semantics of each fact stored in the database \( \mathcal{H} \), we can view it as the integration of several local consistent sources \( \mathcal{H}_1, \ldots, \mathcal{H}_n \). In each \( \mathcal{H}_i \), each fact \( \text{supportedby}(a,b) \) and \( \text{owe}(c) \) has truth-value \( t \) or \( f \). The fact \( \text{supportedby}(\text{charles}, \text{joseph}) \) is true in every \( \mathcal{H}_i \) \( (i = 1, \ldots, n) \) and \( \text{owe}(\text{john}) \) is false in every \( \mathcal{H}_i \) \( (i = 1, \ldots, n) \). Whereas \( \text{owe}(\text{james}) \) is true in some \( \mathcal{H}_j \) \( (j = 1, \ldots, n) \) and false in some \( \mathcal{H}_p \) \( (p = 1, \ldots, n) \).

We now show a 4-valued model \( \mathcal{J} \) of \( P_{\text{job}} \) which includes the facts of the database \( \mathcal{H} \), i.e., \( \mathcal{J} \) agrees with \( \mathcal{H} \) on the values of \( \text{owe} \) and \( \text{supportedby} \) atoms. This 4-valued model \( \mathcal{J} \) contains the facts \( \text{job}(x) \) which corresponds 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 P-Datalog well-founded semantics of \( P_{\text{job}} \) on input \( \mathcal{K} \). The values of the \( \text{job} \) atoms in the derived database \( J \) are the following:

- **true** (t) : \( \text{job}(\text{paul}) \)
- **false** (f) : \( \text{job}(\text{kevin}), \text{job}(\text{james}) \)
- **inconsistent** (i) : \( \text{job}(\text{john}) \)
- **unknown** (u) : \( \text{job}(\text{charles}), \text{job}(\text{joseph}) \)

This model asserts that \( \text{James} \) surely does not get the job: there is some evidence that he owes to the taxation office. From this we can infer that \( \text{Paul} \) surely get the job. Indeed, \( \text{Paul} \) does not owe any tax return and he is supported by \( \text{James} \) who is not a civil servant. It also can be deduced that \( \text{Kevin} \) does not succeed in getting the job because nobody supports him. In \( \text{John} \)’s case, he does not owe the taxation office but it is controversial that he is supported by \( \text{Kevin} \), who is not a public servant himself. Thus it is controversial that \( \text{John} \) gets the job. On the other hand, it is unknown that \( \text{Charles} \) and \( \text{Joseph} \) 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: \( \text{Charles} \) supports \( \text{Joseph} \) and \( \text{Joseph} \) supports \( \text{Charles} \). The only chance for \( \text{Charles} \) getting the job is if \( \text{Joseph} \) (his only support) does not get it. And vice-versa, the only chance for \( \text{Joseph} \) getting the job is if \( \text{Charles} \) (his only support) does not get it. Therefore it is not possible to infer which one will get the job: either \( \text{Charles} \) or \( \text{Joseph} \). This means that this information is **unknown**: we are not able to infer the existence or nonexistence of any source supporting it.

The answer to our query: ‘‘For which people is there some evidence that they will get the job?’’ is \( \text{Paul} \) and \( \text{John} \). Besides, we know that \( \text{Paul} \) surely gets the job, but in \( \text{John} \)’s case, we only can affirm that it is controversial that he gets the job. This is derived from an inconsistent fact \( (\bullet \text{supportedby}(\text{john}, \text{kevin})) \) in the integrated database \( \mathcal{K} \). That means: (1) From the point of view of local sources \( \mathcal{K}_i \), affirming that \( \text{John} \) is supported by \( \text{Kevin} \), it is inferred that \( \text{John} \) gets the job, (2) from the point of view of sources in \( \mathcal{K}_j \) affirming that \( \text{John} \) is not supported by \( \text{Kevin} \), it is inferred that \( \text{John} \) does not get the job. Therefore, in the integrated database \( \mathcal{K} \), as expected, the information \( \text{job}(\text{john}) \) is **derived** as inconsistent.

Differently from some approaches treating paraconsistent query languages \[26,28,11,29\], our well-founded semantics is a natural extension of the well-founded semantics for Datalog\textsuperscript{−} programs proposed by \[27\].\footnote{Other approaches for associating a meaning to a program by means of a Herbrand model verifying some special properties have been proposed in the literature \[31,10\].} 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 \[30\].

The paper is organized as follows: In Section 2 we briefly describe the basic notions of the logic \text{LFI1}. In Section 3, we introduce P-Datalog programs and generalize the notion of database instance to allow the storage of inconsistent information in databases. In Section 4 we describe the well-founded semantics of a P-Datalog program. In Section 5, we present a bottom-up method for evaluating P-Datalog programs based on an alternating fixpoint operator and briefly discuss its implementation. In Section 6, we discuss some
related work. Finally in Section 7, we briefly discuss the implementation of the method we proposed for evaluating P-Datalog queries and also present our perspectives for further research.

2. LFII: A 3-valued paraconsistent logic

In this section we briefly describe the syntax and semantics of LFII. A detailed presentation can be found in [14,17]. The semantics of a P-Datalog program is based on the semantics of LFII. Even if P-Datalog programs constitute a small fragment of the set of LFII formulas (we only consider Prolog-like Horn clauses), inference in P-Datalog is based on the paraconsistent framework of LFII.

Let $\mathbf{R}$ be a finite signature without functional symbols and $\mathbf{Var}$ a set of variables symbols. We assume that formulas of LFII 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”). A formula of LFII is defined inductively by the following statements (and only by them):

1. If $R$ is a predicate symbol of arity $k$ and $x_1, \ldots, x_k$ are constants or variables, then $R(x_1, \ldots, x_k)$ and $x_1 = x_2$ are atomic formulas or atoms. The former is called a relational atom and the later an equality atom.
2. If $F$, $G$ are formulas and $x$ is a variable then $F \_ G$, $F \_ x$, $\neg F$ 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.

In the remainder, we often will use the symbols $\bullet$ and $\circ$ in two different contexts: a syntactic one and a semantic one. In LFII, the symbols $\bullet$ and $\circ$ are new modalities which allow extending the syntax of first-order formulas. So, in LFII, these symbols are used in a syntactic context. In Example 1, these symbols have been used in a semantic context, in order to identify the truth-value of facts stored in the database. We think that these two forms of using the symbols $\circ$ and $\bullet$ will not cause confusion, because the P-Datalog syntax do not include them in its specification. So, in this paper, the use of $\circ$ and $\bullet$ will be restricted to the semantic context.

Definition 1. Let $\mathbf{R}$ be a finite signature and $\mathcal{F}$ the set of facts defined over $\mathbf{R}$. An interpretation over $\mathbf{R}$ is an application $\delta : \mathcal{F} \rightarrow \{ \text{f(false)}, \text{t(true)}, \text{i(inconsistent)} \}$.

An interpretation of facts can be extended to the propositional sentences in a natural way by using the connective matrices described in Fig. 1. Notice that the major difference between LFII and Kleene’s three-valued logic concerns the way both logics interpret the implication $i \rightarrow v$, where $v \in \{ t, i, f \}$. This can be explained by the fact that in Kleene’s Logic the truth-value $i$ characterizes undefined information, whereas in LFII, it stands for overdefined (inconsistent) information.

The connective $\land$ is derived from $\lor$, $\neg : A \land B = \neg (\neg A \lor \neg B)$ and the connective $\rightarrow$ is derived from $\neg, \lor, \bullet : A \rightarrow B = \neg (A \lor \bullet A) \lor B$. In the next section, we will use a derived connective $\sim$, called negation by default, defined as $\sim A = \neg A \land \neg \bullet A$.

---

5 In Kleene’s Logic, the truth-value of $i \rightarrow v$ is $i$ for all $v \in \{ t, i, f \}$.
The extension of $\delta$ to the quantified sentences is not treated here. The reason is that 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. The details of the semantics of quantifiers can be found in [17].

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

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

Remark 1. We notice that the notion of equivalence between formulas in LFI1 is different from the one we are used in classical logic. We say that two formulas $F$ and $G$ are equivalent if for all interpretations $\delta$, $\delta$ satisfies $F$ if and only if $\delta$ satisfies $G$. In fact, this is the same definition used in classical logic, except that in the latter, being equivalent is the same as having the same truth-value. This is not the case for LFI1. For instance, the formulas $\neg\neg A$ and $A$ are equivalent in LFI1, but they do not have the same truth-value under some interpretations. If $\delta$ is an interpretation such that $\delta(A) = i$, we have $\delta(\neg\neg A) = t$. So, the two formulas have not the same truth-values, but both are satisfied by $\delta$. Notice that $\neg A$ is equivalent to $\neg A \lor \bot$ in LFI1.

3. The P-Datalog query language

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 P-Datalog query language which is designed to query databases containing inconsistent information. We assume that the reader is familiar with traditional database terminology [1]. In what follows, we denote by $R(\bar{u})$ the formula $R(u_1, \ldots, u_k)$, where $u_1, \ldots, u_k$ are variables and we denote by $R(\bar{a})$ the ground atom (or fact) $R(a_1, \ldots, a_k)$, where $a_1, \ldots, a_k$ are constants ($k$ is called the arity of $R$).

Definition 3 (Paraconsistent databases). Let $R$ be a database schema (or signature), i.e., a set of relation names (or predicate names) and a set $\text{Dom}$ of constants (the Herbrand
University of its instances). A 3-valued instance over R (or a paraconsistent database) is an interpretation I such that for each R ∈ R the set IR = {ā : I(R(ā)) = t or I(R(ā)) = 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(ā) such that I(R(ā)) = i is intended to be controversial. On the other hand, if I(R(ā)) = t, R(ā) is intended to be a safe information.

Notation: In what follows, we will denote by ◦R(ā) the fact that I(R(ā)) = t and by •R(ā) the fact that I(R(ā)) = i. Here, we use the symbols ◦ and • in a semantic context.

P-Datalog is an extension of Datalog [1]. This well-known deductive query language uses the classical first-order logic as its underlying logic, and a Datalog 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 programs, i.e., first-order clauses with positive and negative literals in their bodies. Negation in P-Datalog (as well as in Datalog) uses the classical first-order logic as its underlying logic, and a Datalog query applies over paraconsistent databases. P-Datalog programs are

Definition 4 (P-Datalog programs). A P-Datalog program is a finite set of rules A ← L1,...,Ln, where A is an atom of the form R(ā), and Li are literals of the form: R(ā) or ~R(ā). R is a relation name and ā is a free tuple of appropriate arity. The atom A is called the head of the rule. The literals L1,...,Ln constitute the body of the rule. 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(ā) where R ∈ sch(P) and ā is a tuple of constants in adom(P) (the Herbrand Base of P). The set of relations which appears 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).

Definition 5 (P-Datalog query). A P-Datalog query is a pair (P, Q(u1,...,un)) where P is a P-Datalog program, Q ∈ idb(P) and u1,...,un 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 Pjob and the 3-valued instance I described in that example constitute a P-Datalog

---

6 In [17] the default negation ~ is referred to as strong negation, since ~ enjoys most of the semantic properties of classical negation. For instance: “for all interpretations I (in the Logic LFI1), I satisfies A if and only if I does not satisfy ~A”. Notice that weak negation ~ does not verify this property.
program \( P \) where \( \text{sch}(P) = \{\text{supportedby, job, owe}\} \), \( \text{dom}(P) = \{\text{charles, john, james, joseph, paul, kevin}\} \) and \( \text{B}(P) = \{\text{owe(charles), owe(joseph), supportedby(charles, joseph), supportedby(joseph, charles), \ldots}\} \). The intensional and extensional schemas are \( \text{edb}(P) = \{\text{supportedby, owe}\} \), \( \text{idb}(P) = \{\text{job}\} \). The pair \((P_{jobs}, \text{job}(x))\) is a P-Datalog query ("For which people is there some evidence that they will get the job?"). The pair \((P_{job}, \text{job}(\text{Kevin}))\) corresponds to the Boolean query "Is there some evidence that Kevin may get the job?"

4. 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 \((P, Q(u_1, \ldots, u_n))\) where \( Q \in \text{idb}(P) \), that is, what their answers are expected to be. Our approach is a natural extension of the well-founded semantics for Datalog\(^\lnot\) [27]. 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 \( \text{Val} = \{\text{true}(t), \text{false}(f), \text{inconsistent}(i), \text{unknown}(u)\} \). In what follows, we assume that the reader is familiar with the notions of lattices, lattice operators, monotonicity and continuity, fixpoints, etc. For details, see [25].

4.1. 4-Valued models

Let us consider the complete lattice \((\text{Val}, \leq)\), where \( f \leq u \leq i \leq t \). We notice that the 4-valued logic introduced by Belnap in [8] can also deal with both inconsistent (i) and incomplete (u) information. The main difference between Belnap’s logic and the underlying 4-valued logic of P-Datalog programs we will introduce in this section concerns the fact that in the latter, the lattice \((\text{Val}, \leq)\) constitutes a total order, whereas in Belnap’s approach the lattices considered are partial orders. As we will see, a total ordering over the set of truth-values \( \text{Val} \) is convenient for generalizing, in a natural way, the 3-valued well-founded semantics of Datalog\(^\lnot\) programs to a 4-valued well-founded semantics for P-Datalog programs.

**Definition 6.** Let \( P \) be a P-Datalog program. A 4-valued instance \( I \) over \( \text{sch}(P) \) is an application \( I: \text{B}(P) \to \{t, f, u, i\} \). If \( I(A) \neq u \) for all \( A \in \text{B}(P) \), we say that \( I \) is total.

The answer of a P-Datalog 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 \( \preceq \) among 4-valued instances over \( \text{sch}(P) \), defined by \( I \preceq J \) iff for each \( A \in \text{B}(P) \), \( I(A) \preceq J(A) \).

The set of 4-valued instances of a P-Datalog program \( P \) is denoted by \( 4-\text{Inst}_P \). It is easy to verify that \((4-\text{Inst}_P, \preceq)\) 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 \( J = \{p, q, \sim s\} \). Let \( J' = \{p, q, \bullet s\} \). Then \( J \preceq J' \).
We extend the 3-valued connective matrices of LF11 (see Fig. 1), to the 4-valued matrices showed in Fig. 2.

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 \( \land \) and \( \neg \) connectives given in Fig. 2.

We notice that the matrix corresponding to the \( \neg \) connective in Fig. 2 is not a straight extension of its counterpart in Fig. 1: in P-Datalog, the truth-value of \( t \rightarrow i \) is \( f \) and in LF11 this truth-value is \( i \). In fact, in the P-Datalog context an inconsistent fact cannot be derived from a set of consistent true facts. This kind of inference is accepted in LF11 (the truth-value of the implication being \( i \)) and so, accepted in the paraconsistent logic.

Let \( I \) be a 4-valued instance and \( A \leftarrow G \) be a P-Datalog rule (here, \( G \) denotes a conjunction of literals of the form \( B \) or \( \neg B \)). Then

\[
I(A \leftarrow G) = \begin{cases} 
  i, & \text{if } I(A) = I(G) = i \\
  t, & \text{if } I(A) > I(G) \quad \text{or} \quad I(A) = I(G) \neq i \\
  f, & \text{otherwise}
\end{cases}
\]

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 \( \text{dom}(P) \). We denote by \( \text{ground}(P) \) the set of instantiated rules of \( P \). A 4-valued instance \( J \) over \( \text{sch}(P) \) satisfies a Boolean combination \( z \) of atoms in \( B(P) \) iff \( J(z) \in \{t,i\} \). A 4-valued model of \( P \) is a 4-valued instance \( J \) over \( \text{sch}(P) \) satisfying each rule in \( \text{ground}(P) \), i.e., the truth-value of each rule in \( \text{ground}(P) \) is \( t \) or \( i \). A 4-valued model \( M \) is minimal iff for all \( M' \subset M \), \( M' \) is not a model.

As we claimed in the introduction, LF11 is the underlying logic of P-Datalog. This is justified by the following proposition:

Proposition 1. Let \( J \) be a paraconsistent instance. Then:

1. If \( J \) satisfies a ground P-Datalog rule \( A \leftarrow L_1, \ldots, L_n \) in the P-Datalog semantics (using the matrices of Fig. 2) then \( J \) satisfies the rule in the LF11 semantics (using the matrices of Fig. 1).
2. If \( J \) satisfies a P-Datalog program \( P \) then \( J \) satisfies the LF11 formula corresponding to the finite conjunction of all instantiated rules of \( P \).

\[
\begin{array}{c|c|c|c|c|c|c}
\sim & \lor & t & i & u & f \\
\hline
 t & f & t & t & t & t \\
 i & f & i & t & i & i \\
u & u & u & t & i & u \\
f & t & f & t & i & u \\
\end{array}
\quad
\begin{array}{c|c|c|c|c|c|c}
\land & t & i & u & f \\
\hline
 t & t & t & t & t \\
 i & i & i & i & i \\
u & u & u & u & u \\
f & f & f & f & f \\
\end{array}
\quad
\begin{array}{c|c|c|c|c|c|c}
\rightarrow & t & i & u & f \\
\hline
 t & t & f & f & f \\
 i & i & i & i & i \\
u & u & u & u & u \\
f & f & f & f & f \\
\end{array}
\]

Fig. 2. P-Datalog 4-valued connective matrices.
Proof

(1) Suppose the instance $J$ satisfies the rule $A \leftarrow G$ in P-Datalog, where $A$ is an atom and $G$ is a conjunction of literals of the form $B$ or $\neg B$. By Definition 7, $J(A \leftarrow G) \in \{i, t\}$. By the definition of the semantics of the $\leftarrow$ connective, $J(A) \geq J(G)$:
(a) $J(A) = t$ and $J(G) \in \{t, i, f\} \Rightarrow J$ satisfies $A \leftarrow G$ in LFII, because $J(A \leftarrow G) = t$ in LFII.
(b) $J(A) = i$ and $J(G) \in \{i, f\} \Rightarrow J$ satisfies $A \leftarrow G$ in LFII, because $J(A \leftarrow G) \in \{t, i\}$ in LFII.
(c) $J(A) = f$ and $J(G) = f \Rightarrow J$ satisfies $A \leftarrow G$ in LFII, because $J(A \leftarrow G) = t$ in LFII.

(2) Follows from the fact that a P-Datalog program is a conjunction of rules and each rule is the conjunction of its ground instances. □

4.2. Extended P-Datalog programs

The well-founded semantics of P-Datalog programs is based on the notion of stable models [22]. Stable models are usually defined as fixpoint of an immediate consequence operator. Following the same idea underlying the definition of 3-stable models in [27], 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 has a unique least fixpoint.

Definition 8. An extended P-Datalog program is a P-Datalog program where (1) negative facts $\neg A$ do not appear in the body of rules; (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 $4-T_P : 4-\text{Inst}_P \rightarrow 4-\text{Inst}_P$ defined as follows. Let $J$ be a 4-valued instance and $A \in \text{B}(P)$, then

$$4-T_P(J)(A) = \begin{cases} \max\{J(F_k)\} & \text{for all rules } A \leftarrow F_k \text{ in } \text{ground}(P), \\ f & \text{otherwise.} \end{cases}$$

We define the sequence $\{4-T_{P,i}(\bot)\}_{i \geq 0} = \{I_0, I_1, I_2, \ldots\}$ of 4-valued instances as follows: $I_0 = \bot$, $I_1 = 4-T_P(I_0)$, $I_2 = 4-T_P(I_1), \ldots$

The following lemma says that the immediate consequence operator for an extended program $P$ has a least fixpoint which coincides with the unique minimal 4-valued model of $P$.

---

7 A rule is a closed universally quantified sentence in LFII. Because the Herbrand Universe is finite, the rule reduces to the conjunction of its ground instances.
Lemma 1. Let $P$ be an extended P-Datalog program.

(1) The operator $4\text{-}TP$ is monotonic.
(2) The 4-valued instance $M$ is a 4-valued model of $P$ iff $4\text{-}TP(M) \leq M$.
(3) The sequence $\{4\text{-}TP_i(\bot)\}_{i \geq 0}$ is increasing and converges to the least fixpoint of $4\text{-}TP$.
(4) $P$ has a unique 4-valued minimal model (denoted by $P(\bot)$) that equals the least fixpoint of $4\text{-}TP$.

Proof

(1) Suppose that $A \in B(P)$, $I(A) \leq J(A)$. We have to show that $4\text{-}TP(I(A)) \leq 4\text{-}TP(J(A))$. We have the following possibilities: (a) There are rules of the form $A \leftarrow F$ in $\text{ground}(P)$, where $F$ corresponds to a conjunction of atoms with truth-values $t$, $i$, $u$, $f$. Then $4\text{-}TP(I(A)) = \max\{I(F)\}$ and $4\text{-}TP(J(A)) = \max\{J(F)\}$. By the fact that $I(A) \leq J(A)$, $\max\{I(F)\} \leq \max\{J(F)\}$. So $4\text{-}TP(I(A)) \leq 4\text{-}TP(J(A))$.

(b) There are no rules in $\text{ground}(P)$ with $A$ in the head. In this case, $4\text{-}TP(I(A)) = 4\text{-}TP(J(A)) = f$.

(2) Let $M$ be a 4-valued model of $P$ and let $A \in B(P)$. If there are no rules in $\text{ground}(P)$ with $A$ in the head, then $4\text{-}TP(M(A)) = f \leq M(A)$. Otherwise, for all rules $r_k$ of the form $A \leftarrow F_k$ in $\text{ground}(P)$, $M(A) \leq \{t,i\}$, and so, $\max\{M(F_k)\} \leq M(A)$. Then, $4\text{-}TP(M) \leq M$. (2 $\Rightarrow$): Let $4\text{-}TP(M) \leq M$. We show that $M$ is a 4-model of $P$. Let $r_k$, $0 \leq k \leq l$ the rules in $\text{ground}(P)$ of the form $A \leftarrow F_k$. Then, $\max\{M(F_k)\} = 4\text{-}TP(M(A)) \leq M(A)$. So, $M(F_k) \leq M(A)$ for all $k$. Thus, $M$ satisfies the rules $r_k$.

(3) Because $4\text{-}Inst_P$ is a finite set, the sequence $\{4\text{-}TP_i(\bot)\}_{i \geq 0}$ reaches a fixpoint after a finite number $N$ of steps. Because $4\text{-}TP$ is monotonic, $I_0 \leq I_1 \leq I_2 \cdots \leq I_N$. So the sequence $\{4\text{-}TP_i(\bot)\}_{i \geq 0}$ is increasing and converges to the least fixpoint of $4\text{-}TP(I_N)$.

(4) $M$ is a minimal 4-valued model of $P$ if and only if $M = \inf\{M|M\text{ is a 4-valued model of } P\}$. We show the following:

1. $M = \inf\{M|4\text{-}TP(M) \leq M\}$, by (2).
2. $M = I_N = \text{least fixpoint of } 4\text{-}TP$ (by monotonicity of $4\text{-}TP$, and by the fact that $I_0 \leq M$).

4.3. 4-Stable models

According to [27], the semantics of a Datalog$^-$ program $P$ is an appropriate 3-valued model 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 instance (a 3-valued instance). We denote by $P_I$ the program obtained from $P$ by adding to $P$ unit clauses $A \leftarrow$ for each $A$ such that $I(A) = t$, and clauses $A \leftarrow i$ for each $A$ such that $I(A) = i$. From now on, we suppose that our programs include these clauses corresponding to the facts of $I$.

Let $I$ be a 4-valued instance over $\text{sch}(P)$. The positivised ground version of $P$ according to $I$ (denoted $pg(P,I)$), is the P-Datalog program obtained from $\text{ground}(P)$ by replacing each negative literal $\neg A$ by $I(\neg A)$ (i.e., by its respective truth-value: $t$, $f$, $u$, $i$). So, $pg(P,I)$ is an
extended P-Datalog program, i.e., a program without negation. By Lemma 1, the least fixpoint \( pg(P, I)(\bot) \) 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 \( pg(P, I)(\bot) \) by \( \text{conseq}_P(I) \), i.e., \( \text{conseq}_P(I) \) is the least fixpoint of the extended P-Datalog program \( pg(P, I) \).

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

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

**Example 4 (4-Stable model).** Consider the P-Datalog program \( P_{job} \) and the input instance \( J \) given in the Example 1. Let us check that \( J \) is a 4-stable model of \( P_{job} \). For this, we have to compute \( \text{conseq}(J) \) and show that \( \text{conseq}(J) = J \). The program \( P' = pg(P, J) \) is

\[
\text{job(charles)} \leftarrow \text{t, supportedby(charles,joseph), u} \\
\text{job(joseph)} \leftarrow \text{t, supportedby(joseph,charles), u} \\
\ldots
\text{supportedby(paul,james)} \leftarrow \\
\text{supportedby(charles,joseph)} \leftarrow \\
\text{supportedby(john,kevin)} \leftarrow \text{i} \\
\ldots
\]

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 \( 4-T_P^1(\bot) = \{\sim \text{job(charles)}, \sim \text{job(joseph)}, \sim \text{job(paul)}, \sim \text{job(john)}, \sim \text{job(james)}, \sim \text{job(kevin)}\} \). We can verify that \( 4-T_P^2(\bot) = 4-T_P^3(\bot) = \{\circ \text{job(paul)}, \bullet \text{job(john)}, \sim \text{job(james)}, \sim \text{job(kevin)}\} \). Thus \( \text{conseq}_P(J) = J \) and \( J \) is a 4-stable model of \( P \). The instance \( J \) coincides with \( I \) (given in the Example 1) for the atoms \( \text{supportedby} \) and \( \text{owe} \).

### 4.4. 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 3). 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 \( P^{4\text{wf}} \).

### 5. 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 [30], that converges to the well-founded semantics. The idea of the method is as follows. We define an alternating sequence \( \{I_i\}_{i \geq 0} \) of 4-valued instances that are underestimates and overestimates of the facts known in every 4-stable model of \( P \). The alternating sequence \( \{I_i\}_{i \geq 0} \) is defined as follows: \( I_0 = \bot \) and \( I_{i+1} = \text{conseq}_P(I_i) \), for \( i \geq 0 \).

We notice that each \( I_i \) is constructed starting from the total instance \( \bot \) by repeated applications of \( \text{conseq}_P \). Thus each \( I_i \) is a total instance, i.e., has no undefined truth-values.

**Theorem 1.** The operator \( \text{conseq}_P \) is antimonotonic. That is, if \( I \preceq J \) then \( \text{conseq}_P(J) \preceq \text{conseq}_P(I) \).

**Proof.** Let us suppose that \( I \preceq J \) and let \( A \in \mathcal{B}(P) \). We will show that \( \text{conseq}_P(J)(A) \leq \text{conseq}_P(I)(A) \). We remind that \( \text{conseq}_P(I) \) is the least fixpoint of the extended P-Datalog program \( pg(P, I) \), denoted by \( pg(P, I)(\bot) \). We have the following cases:

1. \( \text{conseq}_P(J)(A) = t \).

   We affirm that for all \( n \geq 1 \), if \( n \) is such that \( 4-T^n_{pg(P, J)}(\bot)(A) = t \), and \( \forall j, n, 4-T^j_{pg(P, J)}(\bot)(A) = t \), then \( \text{conseq}_P(I)(A) = t \). This assertion is proved by induction on \( n \).

   - **Induction base:** \( n = 1 \). By our hypothesis, in \( \text{ground}(P) \) there is a rule of the form:
     \[ A \leftarrow B_1, \ldots, B_n, \text{ where } B_k \text{ are atoms, } J(B_k) = f \forall k, 1 \leq k \leq n. \]
     By \( I \preceq J \), \( I(B_k) = f \forall k, 1 \leq k \leq n \).
     Thus \( \text{conseq}_P(I)(A) = t \) and the induction base is proved.

   - **Induction step:** by our hypothesis, in \( \text{ground}(P) \) there exists a rule of the form:
     \[ A \leftarrow B_1, \ldots, B_n, D_1, \ldots, D_m, \text{ where } B_k, D_g \text{ are atoms, } J(B_k) = f \forall k, 1 \leq k \leq n \]
     and \( 4-T^n_{pg(P, J)}(\bot)(D_g) = t \forall g, 1 \leq g \leq m \).
     By \( I \preceq J \), \( I(B_k) = f \forall k, 1 \leq k \leq n \).
     By the induction step hypothesis, \( \text{conseq}_P(I)(D_g) = t \forall g, 1 \leq g \leq m \).
     Thus \( \text{conseq}_P(I)(A) = t \).
     The induction is proved and \( \text{conseq}_P(I)(A) = t \).

2. \( \text{conseq}_P(J)(A) = i \).

   We affirm that for all \( n \geq 1 \), if \( n \) is such that \( 4-T^n_{pg(P, J)}(\bot)(A) = i \), and \( \forall j, n, 4-T^j_{pg(P, J)}(\bot)(A) = i \), then \( \text{conseq}_P(I)(A) \geq i \). This assertion is proved by induction on \( n \).

   - **Induction base:** \( n = 1 \). By our hypothesis, in \( \text{ground}(P) \) there is a rule of the form:
     \[ A \leftarrow B_1, \ldots, B_n, c_1, \ldots, c_p, \text{ where } B_k \text{ are atoms and } c_g \text{ are truth-values, } J(B_k) = f \forall k, 1 \leq k \leq n, c_g \geq i \forall g, 1 \leq g \leq p, \text{ there is a } w, c_w = i, 1 \leq w \leq p. \]
     By \( I \preceq J \), \( I(B_k) = f \forall k, 1 \leq k \leq n \).
     Thus \( \text{conseq}_P(I)(A) \geq i \) and the induction base is proved.

   - **Induction step:** By our hypothesis, in \( \text{ground}(P) \) there is a rule of the form:
     \[ A \leftarrow B_1, \ldots, B_n, D_1, \ldots, D_m, \text{ where } B_k, D_g \text{ are atoms, } J(B_k) = f \forall k, 1 \leq k \leq n \]
     and \( 4-T^n_{pg(P, J)}(\bot)(D_g) \geq i \forall g, 1 \leq g \leq m \).
     By \( I \preceq J \), \( I(B_k) = f \forall k, 1 \leq k \leq n \).
By the induction step hypothesis, \( \text{conseq}_P(\mathcal{I})(D_g) \geq i \) \( \forall g, 1 \leq g \leq m \). Thus \( \text{conseq}_P(\mathcal{I})(A) = i \).

The induction is proved and we can conclude that \( \text{conseq}_P(\mathcal{I})(A) \geq i \).

(3) \( \text{conseq}_P(\mathcal{J})(A) = u \).

We affirm that for all \( n \geq 1 \), if \( n \) is such that \( 4^{-T_{P_R}(P,A)}(\perp)(A) = u \), and \( \forall j, j \geq n \), \( 4^{-T_{P_R}(P,A)}(\perp)(A) = u \), then \( \text{conseq}_P(\mathcal{I})(A) \geq u \). This assertion is proved by induction on \( n \).

- Induction base: \( n=1 \). By our hypothesis, in \( \text{ground}(P) \) there is a rule of the form: \( A \leftarrow \sim B_1, \ldots, \sim B_n, c_1, \ldots, c_p \), where \( B_k \) are atoms and \( c_g \) are truth-values, \( \mathcal{J}(B_k) \in \{f,u\} \) \( \forall k \), \( 1 \leq k \leq n \), and there is a \( w \), \( 1 \leq w \leq p \), \( c_w = u \) or there is a \( y \), \( 1 \leq y \leq n \), \( \mathcal{J}(B_k) = u \).

By \( \mathcal{I} \preceq \mathcal{J} \), \( \mathcal{I}(B_k) \in \{f,u\} \) \( \forall k \), \( 1 \leq k \leq n \).

Thus \( \text{conseq}_P(\mathcal{I})(A) \geq u \) and the induction base is proved.

- Induction step: by our hypothesis, in \( \text{ground}(P) \) there is a rule of the form: \( A \leftarrow \sim B_1, \ldots, \sim B_n, D_1, \ldots, D_m \), where \( B_k, D_g \) are atoms, \( \mathcal{J}(B_k) \in \{f,u\} \) \( \forall k \), \( 1 \leq k \leq n \) and \( 4^{-T_{P_R}(P,A)}(\perp)(D_g) \geq u \) \( \forall g \), \( 1 \leq g \leq m \).

By \( \mathcal{I} \preceq \mathcal{J} \), \( \mathcal{I}(B_k) \in \{f,u\} \) \( \forall k \), \( 1 \leq k \leq n \).

By the induction step hypothesis, \( \text{conseq}_P(\mathcal{I})(D_g) \geq u \) \( \forall g \), \( 1 \leq g \leq m \).

Thus \( \text{conseq}_P(\mathcal{I})(A) \geq u \).

The induction is proved and we can conclude that \( \text{conseq}_P(\mathcal{I})(A) \geq u \).

(4) \( \text{conseq}_P(\mathcal{J})(A) = f \).

As \( \text{conseq}_P(\mathcal{I})(A) \geq f \), we conclude that \( \text{conseq}_P(\mathcal{I})(A) \geq \text{conseq}_P(\mathcal{J})(A) \).

From Theorem 1, we can easily see that in the alternating sequence \( \{I_i\}_{i \geq 0} \) we have

\[
I_0 \preceq I_2 \preceq \cdots \preceq I_{2i} \preceq I_{2i+2} \preceq \cdots \preceq I_{2i+1} \preceq I_{2i-1} \preceq \cdots \preceq I_3 \preceq I_1
\]

(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: \( I_{2k_0} = I_{2k_0+2} = \cdots = I_{2k_0+4} \) and \( I_{2j_0+1} = I_{2j_0+3} = \cdots = I_{2j_0+5} = \cdots \) for some \( k_0 \geq 0 \) and some \( j_0 \geq 0 \).

Let \( I_* \) be the least upper bound of the increasing sequence: \( I_* = \text{lub}\{I_{2i}\}_{i \geq 0} \), and let \( I^* \) be the greatest lower bound of the decreasing sequence:

\[ I^* = \text{glb}\{I_{2i+1}\}_{i \geq 0}. \]

From (1), it follows that \( I_* \geq I^* \).

**Lemma 2.** Let \( I \) be a 4-valued instance of a \( P \)-Datalog program. Then \( \text{conseq}_P(I_*) = I^* \) and \( \text{conseq}_P(I^*) = I_* \).

**Proof.** Let \( \{I_i\}_{i \geq 0} \) be an alternating sequence of 4-valued instances, where \( I_* = I_{2k} = I_{2k+2} \) and \( I^* = I_{2k+1} = I_{2k+3} \), \( k \geq 0 \). We need to show that \( \text{conseq}_P(I^*) = I_* \) and \( \text{conseq}_P(I_*) = I^* \).

In \( \{I_i\}_{i \geq 0} \) we have \( I_{i+1} = \text{conseq}_P(I_i) \), \( I_{2k} = \text{conseq}_P(I_{2k-1}) \), \( I_{2k+1} = \text{conseq}_P(I_{2k}) \), \( I_{2k+2} = \text{conseq}_P(I_{2k+1}) \), \( I_{2k+3} = \text{conseq}_P(I_{2k+2}) \). Thus, \( I_* = \text{conseq}_P(I^*) \) and \( I^* = \text{conseq}_P(I_*) \).

From the 4-valued instances \( I_* \) and \( I^* \) we can define the 4-valued instance \( I_0 \) which coincides with the well-founded semantics of a \( P \)-Datalog program, as we will see in Theorem 3.
Definition 12. Let \( \Gamma^* \) be a 4-valued instance of a P-Datalog program \( P \), consisting of the facts known in both \( \Gamma \) and \( \Gamma^* \), that is

\[
\Gamma^*(A) = \begin{cases} 
  t, & \text{if } \Gamma(A) = \Gamma^*(A) = t \\
  i, & \text{if } \Gamma(A) = \Gamma^*(A) = i \\
  f, & \text{if } \Gamma(A) = \Gamma^*(A) = f \\
  u, & \text{otherwise}
\end{cases}
\]

Theorem 2. Let \( \mathbf{I} \) be a 4-valued instance of a P-Datalog program \( P \). Then \( \mathbf{I} \leq \mathbf{I}^*_v \leq \Gamma^* \).

Proof. By the definition of \( \mathbf{I}^*_v \), \( \mathbf{I} \leq \mathbf{I}^*_v \leq \Gamma^* \) is verified for all cases except when \( \mathbf{I}(A) = i \) and \( \Gamma^*(A) = t \). We show that this cannot happen. Let us suppose that \( \mathbf{I}(A) = i \) and \( \Gamma^*(A) = t \).

By the fact that \( \text{conseq}_{\mathit{P}}(\mathbf{I}_v) = \Gamma^* \), \( \text{conseq}_{\mathit{P}}(\Gamma^*) = \mathbf{I}_v \), it follows that \( \text{conseq}_{\mathit{P}}(\Gamma^*)(A) = i \) and \( \text{conseq}_{\mathit{P}}(\mathbf{I}_v)(A) = t \). We can prove by induction on \( n \) that: for all \( n \geq 1 \), if \( n \) is such that \( 4^n T_{pg(P,\mathbf{I}_v)}(\perp)(A) = t \), and \( \forall j, j \geq n, 4^n T_{pg(P,\mathbf{I}_v)}(\perp)(A) = t \), then \( \text{conseq}_{\mathit{P}}(\Gamma^*)(A) \in \{f, t\} \). So, we can conclude that \( \text{conseq}_{\mathit{P}}(\Gamma^*)(A) \in \{f, t\} \). Contradiction. \( \square \)

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 \( \langle \Gamma^*_v \rangle \) and that the well-founded semantics coincides with \( \Gamma^*_v \).

Theorem 3. For each P-Datalog program \( P \), (a) \( \Gamma^*_v \) is a 4-stable model of \( P \) and (b) \( \Gamma^*_v \) equals the well-founded semantics of \( P \) (P4wf).

Proof. For statement (a), we need to show that \( \text{conseq}_{\mathit{P}}(\Gamma^*_v) = \Gamma^*_v \).

From Theorems 1 and 2, we can easily conclude that:

\[
\mathbf{I}_v \leq \text{conseq}_{\mathit{P}}(\Gamma^*_v) \leq \Gamma^*
\]

We affirm that \( \text{conseq}_{\mathit{P}}(\Gamma^*_v)(A) = \Gamma^*_v(A) \) for all \( A \in \mathcal{B}(P) \). Indeed: (1) If \( \Gamma^*_v(A) \in \{f, t, i\} \), by the definition of \( \Gamma^*_v \) and from (2), we can conclude that \( \text{conseq}_{\mathit{P}}(\Gamma^*_v)(A) = \Gamma^*_v(A) \).

(2) Let \( \Gamma^*_v(A) = u \). By the definition of \( \Gamma^*_v \) we have the following possibilities:

(2a) \( \mathbf{I}_v(A) = i \) and \( \Gamma^*(A) = t \). By Theorem 2, this case does not happen.

(2b) \( \mathbf{I}_v(A) = f \) and \( \Gamma^*(A) \in \{i, t\} \). This can be written as \( \text{conseq}_{\mathit{P}}(\Gamma^*)(A) = f \) and \( \text{conseq}_{\mathit{P}}(\mathbf{I}_v)(A) \geq i \). By definition, \( \text{conseq}_{\mathit{P}}(\mathbf{I}_v) \) is the least fixpoint of \( pg(P,\mathbf{I}_v)(\perp) \). We affirm that for all \( n \geq 1 \), if \( n \) is such that \( 4^n T_{pg(P,\mathbf{I}_v)}(\perp)(A) \geq i \), and \( \forall j, j \geq n, 4^n T_{pg(P,\mathbf{I}_v)}(\perp)(A) = 4^n T_{pg(P,\mathbf{I}_v)}(\perp)(A) \), then \( \text{conseq}_{\mathit{P}}(\Gamma^*_v)(A) = u \). This assertion can be proved by induction on \( n \). So, we can conclude that \( \text{conseq}_{\mathit{P}}(\Gamma^*_v)(A) = \mathbf{I}_v(A) = u \). For statement (b), we will show that \( P4wf = \Gamma^*_v \). Let \( A \in \mathcal{B}(P) \).

- Let \( P4wf(A) = t(i, f) \). By definition, \( P4wf \) is the 4-valued instance consisting of facts that are true (resp. false, inconsistent) in all 4-stable models of \( P \). From statement (1), we can affirm that \( \Gamma^*_v(A) \) is a 4-stable model. So \( \Gamma^*_v(A) = P4wf(A) \).
- Let \( P4wf(A) = u \). So, there exist 4-stable models \( M_1 \) and \( M_2 \) such that \( M_1(A) \neq M_2(A) \). Let us suppose that \( \Gamma^*_v(A) = t(i, f) \). We affirm that for all 4-stable model \( M \) of program \( P \), and for all \( i \geq 0 \), we have:
\[ I_{2j} \preceq M \preceq I_{2j+1}. \] (3)

This assertion can be proved by induction on \( i \) and its proof does not present any difficulty.

By (3), \( I_* \preceq M_j \preceq I^* \), for \( j = 1, 2 \).

- If \( I_*(A) = t \), then \( I_+(A) = t \), and by \( I_+(A) \preceq M_j(A) \), we conclude that \( M_1(A) = M_2(A) = t \).
- If \( I_+(A) = i \), then \( I_+(A) = I^*(A) = i \), and by \( I_+(A) \preceq M_j(A) \preceq I^*(A) \), we conclude that \( M_1(A) = M_2(A) = i \).
- If \( I_+(A) = f \), then \( I^*(A) = f \), and by \( I^*(A) \geq M_j(A) \), we conclude that \( M_1(A) = M_2(A) = f \).

Contradiction. So, \( I^*_+(A) = u \). □

We illustrate this computation in our running example:

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

\[ \begin{align*}
I_0 &= \{ \neg \text{job(charles)}, \neg \text{job(james)}, \neg \text{job(john)}, \neg \text{job(joseph)}, \neg \text{job(kevin)}, \neg \text{job(paul)} \}. \\
I_1 &= \{ \odot \text{job(charles)}, \text{job(james)}, \bullet \text{job(john)}, \odot \text{job(joseph)}, \neg \text{job(kevin)}, \odot \text{job(paul)} \}. \\
I_2 &= \{ \neg \text{job(charles)}, \odot \text{job(james)}, \bullet \text{job(john)}, \odot \text{job(joseph)}, \neg \text{job(kevin)}, \odot \text{job(paul)} \}. \\
I_3 &= \{ \odot \text{job(charles)}, \text{job(james)}, \bullet \text{job(john)}, \odot \text{job(joseph)}, \neg \text{job(kevin)}, \odot \text{job(paul)} \}. \\
I_4 &= \{ \text{job(charles)}, \text{job(james)}, \bullet \text{job(john)}, \odot \text{job(joseph)}, \neg \text{job(kevin)}, \odot \text{job(paul)} \}. \\
I_5 &= I_3 \text{ and } I_6 = I_4. \text{ So, } I_* = I_6 \text{ and } I^*_+ = I_5. \text{ Thus } I^*_+ = \{ \neg \text{job(james)}, \bullet \text{job(john)}, \neg \text{job(kevin)}, \odot \text{job(paul)} \}. \text{ This is exactly the natural answer for } P_{job} \text{ we have informally discussed in Example 1.}
\]

6. Related work

The problem of inconsistent information management arising from the integration of heterogeneous sources has been widely studied in recent research. We can distinguish three main approaches to handle the inconsistency problem in knowledge bases: a consistency-based approach (where inconsistency is eliminated), a paraconsistent logic approach (where inconsistency is not rejected and inference methods can draw plausible conclusions from it) and a hybrid approach (formalisms which do not reject any information but instead associate degrees of belief, reliability or uncertainty to each source knowledge base).

In what follows, we discuss some recent papers which deal with the subject of integrating multi-source information and querying the resulting integrated database which maybe contains inconsistencies. The methods are grouped following the three main approaches mentioned above. We begin the discussion by summarizing the main issues treated in this paper.
6.1. Our approach

The present paper follows the paraconsistent logic approach. We assume that data integration has already been achieved following the method we proposed in [18]. This method follows a paraconsistent approach for data integration and is based on a tableau proof system of LFI1 [13]. It supposes the existence of \( n \) source databases \( K_1, K_2, \ldots, K_n \), where each database \( K_i \) satisfies a set of integrity constraints \( C_i \) (properties expressed by first-order formulas). The method builds an integrated database \( K \) which satisfies the set \( \{C_1, C_2, \ldots, C_n\} \) of integrity constraints, where the notion of satisfiability is taken from the paraconsistent logic LFI1. The integrated database \( K \) is a paraconsistent database: each fact stored in it can be either sure (positive or negative) or inconsistent. In the present paper, we propose a deductive language P-Datalog which allows to query this integrated database containing inconsistencies. The querying process semantics follows the paraconsistent semantics of the logic LFI1.

6.2. Consistent-based approaches

In [4,3] a logic framework based on annotated predicate logic [23] is proposed for obtaining consistent answers when querying a database with inconsistent information with respect to a set of integrity constraints. A consistent answer to a given query \( q \) is the one that every database repair would give in response to \( q \). A procedure for building consistent answers without actually repairing the inconsistent database is described. In [2], a method based on the classical tableau system proof for first-order logic is introduced for producing consistent answers in a relational database that may violate given integrity constraints. Roughly speaking, these answers are those obtained in all repaired versions of the database (a repaired version is obtained by inserting or deleting information from the original database in order to make it consistent with the integrity constraints). In [12], a method for specifying the database repairs of a mediated integration system under the paradigm “Local-as-View” for data integration has been proposed. The repairs are specified by means of a disjunctive logic program. The consistent answers to queries posed to such a system are computed by running a disjunctive logic program together with the specification of database repairs. Following this same line, in [20], a generic logic programming framework for computing consistent answers to queries posed to a data integration system (in which inconsistency possibly arises) is proposed. This approach is also based on the specification of repairs and queries by means of a logic program. The main objective is optimizing the evaluation of queries expressed as logic programs.

In [7], one proposes a method to specify database repairs using simple classical normal disjunctive logic programs with a stable model semantics. The database predicates in these programs contain annotations as extra arguments. The annotations come from the classical annotated predicated logic, contrarily to annotated or paraconsistent logic programs [11,29]. Even though their method produces all the possible minimal sets of changes required to restore the consistency of a relational database, the main goal of this method is to obtain the consistent answers to a first-order query. Our language P-Datalog is designed to query paraconsistent databases, i.e., where inconsistent information is kept inside the database with a special truth-value which distinguishes it from safe information. When using P-Datalog to query a paraconsistent database, one can specify the type of answers we are interested in: consistent or inconsistent, or both. P-Datalog classifies each
fact in a query answer as true, false, inconsistent or unknown. In [5], the Belnap 4-valued logic [8] is used to represent different degrees of contradiction and partial information. The lattice of truth-values they use is different from ours. The intuition behind their semantics is that contradictory data corresponds to inadequate information about the real world, and therefore it should be minimized. In [6], an abductive method for coherent integration of independent datasources is proposed. The method is based on SLDNF-Resolution. The general idea consists in computing a list of data-facts that should be inserted to the concatenated database or deleted from it in order to restore its consistency.

6.3. Hybrid approaches

In [16] a majority merging operator is used in the definition of a logic that allows us to reason with data coming from different sources. The possible inconsistencies that would come from the integration process are solved according to that majority operator. In our approach, we have assumed that our data have already been integrated and in this integrated database, inconsistent information has been detected and marked. In [9], several methods allowing inference of information coming from different sources are studied. The methods proposed can be grouped in two categories: those following a coherent-based approach (where inconsistencies are discarded) and those where inconsistent information is kept. The later ones are closer to our approach since they do not rely on restoring consistency in the integrated database. The authors suppose the knowledge bases are sets of formulas which are stratified, that is, there exists a total ordering between the sources based on a notion of reliability: source $\mathcal{K}_j$ is more reliable than source $\mathcal{K}_i$ if $j > i$. In our approach, the information kept in the integrated database is not stratified, since in most situations, there is no means of attaching a degree of reliability to information coming from different sources (for instance, simple information like client addresses may have different values in two different databases, and there is no means of saying that the first database is more reliable than the second one). Besides, the problem we propose is different: our integrated database is a set of (possibly inconsistent) facts and we propose a language to query this set of facts, producing a new set of facts, where each element is clearly identified as positive sure information, negative sure information or inconsistent information.

6.4. Paraconsistent approach

The approach proposed in [28] is close to the approach we introduced in this paper. Among all the methods we mentioned so far for querying databases containing inconsistent information, this is the only one based on a paraconsistent logic programming framework, like ours. This work proposes a deductive query language and a declarative and fixpoint-based well-founded semantics for this language. The underlying logic is a paraconsistent logic with seven truth-values: true, inconsistent, false, indifferent, true by default, false by default, and unknown. An interpretation satisfies a formula if the truth-value it associates to the formula is true or inconsistent. The main differences between this approach and ours are the following: (1) The underlying logic is far more complex than LFII since it involves seven truth-values, (2) the body of the clauses contains explicit negation and default negation, and the head may contain a negative atom. Thus the language is not a simple extension of Datalog\textsuperscript{−}. P-Datalog has been designed to extend to a paraconsistent context the classical Datalog\textsuperscript{−} query language used for querying consistent
databases. This has been achieved by augmenting the three-value semantics of Datalog\(^{-}\) with only one new value (inconsistent).

7. Conclusion and further research

In this paper, we have introduced P-Datalog, a deductive query language for querying databases obtained by the concatenation of several sources, which together can contain inconsistent information. We have provided a declarative well-founded semantics which can be evaluated by means of an alternating fixpoint. 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 [24]. 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.

Several work remains to be done. We will extend the syntax of P-Datalog in order to allow literals of the form \(\circ A\), \(\bullet A\) and \(\neg A\) in the body of the rules. The \(\circ\), \(\bullet\) and \(\neg\) operators are non-monotonic and that will imply in modifications on the definition of extended P-Datalog program. Another direction for further research consists in relating the well-founded semantics of Datalog\(^{-}\) queries posed to the source databases with the well-founded semantics of P-Datalog we have proposed in this paper. How the answers of Datalog\(^{-}\) queries submitted to the (consistent) source databases \(D_1, D_2, \ldots, D_n\) are related to the answer of a P-Datalog query submitted to the integrated database \(D = D_1 \cup \ldots \cup D_n\). More precisely, given a P-Datalog query \(Q\) over \(D\), there exists Datalog\(^{-}\) queries \(Q_1, Q_2, \ldots, Q_n\) over \(D_1, \ldots, D_n\), respectively, such that the integration of their answers (using the integration method introduced in [18]) coincides with the answer of the P-Datalog \(Q'\). A complete investigation of how Datalog\(^{-}\) queries over the consistent sources relate to P-Datalog queries over the concatenated database would formalize the intuitive idea we gave of P-Datalog query answering in Example 1.

Finally, we intend to present a more efficient implementation of P-Datalog, based on a proof system for the LF1 logic. In [13], a sound and complete tableau proof system for LF1 has been developed and in [18], a database integration method based on this proof system has been introduced. We intend to use this proof system or an equivalent resolution proof system for evaluating the well-founded semantics of P-Datalog queries.

Acknowledgements

We would like to thank the anonymous referees whose really careful reading, useful comments and corrections have lead to significant improvement of our work.

References


