

Construction of P-Minimal Models Using Paraconsistent Relational Model

Badrinath Jayakumar^(✉) and Rajshekhar Sunderraman

Department of Computer Science, Georgia State University, Atlanta, GA, USA
{bjayakumar2,raj}@cs.gsu.edu
<http://www.cs.gsu.edu/>

Abstract. Positive extended disjunctive deductive databases are those that contain explicit negation both in the head and body of the clauses. For such databases, paraconsistent minimal models (p-minimal models) have been proposed based on multi-valued logic (four-valued logic). Moreover, the paraconsistent relational model is also based on four-valued logic. In this paper, we propose an algorithm, which converts clauses to equations and solves it, to find p-minimal models using the paraconsistent relational model. In order to accomplish that, we use disjunctive paraconsistent relation model.

Keywords: Inconsistency · Paraconsistent relational model · Fixed-point semantics · Four-valued logic

1 Introduction

The paraconsistent relational model moves a step forward and completes the relational model by representing both positive and negative information for any given relation. The model was first proposed by Bagai and Sunderraman [5]. The authors have given two applications for the paraconsistent relational model: weak well-founded semantics [5] and well-founded semantics [7] for general deductive databases. Bagai and Sunderraman find the models by constructing a system of algebraic equations for the clauses in the database. There are two advantages for this approach: it operates on a set of tuples instead of “tuple-at-a-time” basis and the algebraic expression in the algebraic equation can be optimized based on various laws of equality. The optimizations are similar to the ordinary relations case where selections and projections are pushed deeper into expressions whenever possible [5].

Paraconsistent logic [4,10,13] does not trivialize the result in the presence of inconsistent information. Four-valued logic [8], which is a type of paraconsistent logic, was introduced in logic programming by Blair and Subrahmanian [9]. Three prominent works have been done in positive extended disjunctive deductive databases with respect to inconsistencies: The first, answer set semantics, by Gelfond and Lifschitz [12], trivialize the results in the presence of inconsistencies. The second, p-minimal models, by Sakama and Inoue [17], which is based

on four-valued logic [8], tolerates inconsistencies. In addition to that, for both logic programs and disjunctive logic programs many works have been proposed [1–3, 11], where all of the approaches are based on four-valued logic. The third, the quasi-classic models, by Zhang et al. [19], has stronger inference power than p-minimal models because the quasi-classic models support disjunctive syllogism and disjunction introduction. Moreover, the quasi-classic models are based on quasi-classic logic [14].

In this paper, we use the paraconsistent relational model and propose an algorithm to find p-minimal models for positive extended disjunctive deductive databases. The central idea in arriving at p-minimal models for a given positive extended disjunctive deductive database is to associate paraconsistent relations with the predicate symbols. We then construct a system of algebraic equations for the clauses in positive extended disjunctive deductive databases. The equations are then used to incrementally construct p-minimal models with the help of disjunctive paraconsistent relations.

The rest of this paper is organized as follows: in Sect. 2, we discuss preliminaries to understand the paper; in Sect. 3, we explain the disjunctive paraconsistent relational model; in Sect. 4, we propose an algorithm to find p-minimal models; in Sect. 5, we state the conclusion and future work for the paper.

2 Preliminaries

Before we explain the details of the actual contribution of this paper, we must briefly review positive extended disjunctive deductive databases [16, 17] and the paraconsistent relation model [5, 6] which help to understand the paper.

Given a first order language \mathcal{L} , a disjunctive deductive database P [16] consists of logical inference rules of the form: r (rule) $= l_0 \vee \dots \vee l_n \leftarrow l_{n+1}, \dots, l_m$. A rule is called a positive disjunctive rule if the rule has both head (disjunction of literals) and body (conjunction of literals). Concretely, the rule r is called positive extended disjunctive rule if $l_0, \dots, l_n, l_{n+1}, \dots, l_m$ are literals which are either positive or negative (\neg) atoms. For the given syntax of positive extended disjunctive deductive databases, we reproduce the fixed-point semantics of P [17].

Fixed-Point Semantics. Let P be a positive extended disjunctive deductive database and \mathcal{I} be a set of interpretations, then $\mathcal{T}_P(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} T_P(I)$

$$T_P(I) = \begin{cases} \emptyset, & \text{if } l_{n+1}, \dots, l_m \subseteq I \text{ for some} \\ & \text{ground constraint } \leftarrow l_{n+1} \dots l_m \text{ from } P; \\ \{J \mid \text{for each ground clause} \\ & r_i: l_0 \vee \dots \vee l_n \leftarrow l_{n+1}, \dots, l_m \text{ such that} \\ & \{l_{n+1}, \dots, l_m\} \subseteq I, J = I \cup \bigcup_{r_i} \{l_j\} (0 \leq j \leq n)\}, & \text{otherwise.} \end{cases}$$

In the definition of $T_P(I)$, $\{l_j\} (1 \leq j \leq n)$ is a collection of sets where every set in the collection contains a disjunct. For any positive extended disjunctive

deductive database P , \mathcal{T}_P is finite and $\mathcal{T}_P \uparrow n = \mathcal{T}_P \uparrow \omega$ where n is a successor ordinal and ω is a limit ordinal. For any positive extended disjunctive deductive database P , p-minimal models = $\min(\mu(\mathcal{T}_P \uparrow \omega))^1$ where $\min(\mathcal{I}) = \{I \in \mathcal{I} \mid \nexists J \in \mathcal{I} \text{ such that } J \subset I\}$.

Unlike normal relations where we only retain information believed to be true of a particular predicate, we also retain what is believed to be false of a particular predicate in the paraconsistent relational model. Let a relation scheme Σ be a finite set of attribute names, where for any attribute name $A \in \Sigma$, $\text{dom}(A)$ is a non-empty domain of values for A . A tuple on Σ is any map $t: \Sigma \rightarrow \bigcup_{A \in \Sigma} \text{dom}(A)$, such that $t(A) \in \text{dom}(A)$ for each $A \in \Sigma$. Let $\tau(\Sigma)$ denote the set of all tuples on Σ . An ordinary relation on scheme Σ is thus any subset of $\tau(\Sigma)$. The paraconsistent relation on a scheme Σ is a pair $\langle R^+, R^- \rangle$ where R^+ and R^- are ordinary relations on Σ . Thus R^+ represents the set of tuples believed to be true of R , and R^- represents the set of tuples believed to be false.

Algebraic Operators. Two types of algebraic operators are defined here: (i) Set Theoretic Operators, and (ii) Relational Theoretic Operators.

Set Theoretic Operators. Let R and S be two paraconsistent relations on scheme Σ .

Union. The union of R and S , denoted $R \dot{\cup} S$, is a paraconsistent relation on scheme Σ , given that $(R \dot{\cup} S)^+ = R^+ \cup S^+$, $(R \dot{\cup} S)^- = R^- \cap S^-$.

Complement. The complement of R , denoted $\dot{-}R$, is a paraconsistent relation on scheme Σ , given that $\dot{-}R^+ = R^-$, $\dot{-}R^- = R^+$.

Intersection. The intersection of R and S , denoted $R \dot{\cap} S$, is a paraconsistent relation on scheme Σ , given that $(R \dot{\cap} S)^+ = R^+ \cap S^+$, $(R \dot{\cap} S)^- = R^- \cup S^-$.

Difference. The difference of R and S , denoted $R \dot{-} S$, is a paraconsistent relation on scheme Σ , given that $(R \dot{-} S)^+ = R^+ \cap S^-$, $(R \dot{-} S)^- = R^- \cup S^+$.

Relation Theoretic Operators. Let Σ and Δ be relation schemes such that $\Sigma \subseteq \Delta$ and let R and S be paraconsistent relations on schemes Σ and Δ .

Join. The join of R and S , denoted $R \dot{\bowtie} S$, is a paraconsistent relation on scheme $\Sigma \cup \Delta$, given that $(R \dot{\bowtie} S)^+ = R^+ \bowtie S^+$, $(R \dot{\bowtie} S)^- = (R^-)^{\Sigma \cup \Delta} \cup (S^-)^{\Sigma \cup \Delta}$.

Projection. The projection of R onto Δ , denoted $\dot{\pi}_\Delta(R)$, is a paraconsistent relation on Δ , given that $\dot{\pi}_\Delta(R)^+ = \pi_\Delta(R^+)^{\Sigma \cup \Delta}$, $\dot{\pi}_\Delta(R)^- = \{t \in \tau(\Delta) \mid t^{\Sigma \cup \Delta} \subseteq (R^-)^{\Sigma \cup \Delta}\}$ where π_Δ is the usual projection over Δ of ordinary relations.

Selection. Let F be any logic formula involving attribute names in Σ , constant symbols, and any of these symbols $\{=, \neg, \wedge, \vee\}$. Then, the selection of R by F , denoted $\dot{\sigma}_F(R)$, is a paraconsistent relation on scheme Σ , given that $\dot{\sigma}_F(R)^+ = \sigma_F(R^+)$, $\dot{\sigma}_F(R)^- = R^- \cup \sigma_{\neg F}(\tau(\Sigma))$, where σ_F is a usual selection of tuples satisfying F from ordinary relations.

The following example is taken from Bagai and Sunderraman's paraconsistent relational data model [5].

¹ $\mu(\mathcal{T}_P \uparrow \omega) = \{I \mid I \in \mathcal{T}_P \uparrow \omega \text{ and } I \in \mathcal{T}_P(I)\}$.

Example 1. *Strictly speaking, relation schemes are sets of attribute names. However, in this example we treat them as ordered sequence of attribute names, so tuples can be viewed as the usual lists of values. Let $\{a, b, c\}$ be a common domain for all attribute names, and let R and S be the following paraconsistent relations on schemes $\langle X, Y \rangle$ and $\langle Y, Z \rangle$ respectively:*

$$\begin{aligned} R^+ &= \{(b, b), (b, c)\}, R^- = \{(a, a), (a, b), (a, c)\} \\ S^+ &= \{(a, c), (c, a)\}, S^- = \{(c, b)\}. \end{aligned}$$

Then, $R \bowtie S$ is the paraconsistent relation on scheme $\langle X, Y, Z \rangle$:

$$\begin{aligned} (R \bowtie S)^+ &= \{(b, c, a)\} \\ (R \bowtie S)^- &= \{(a, a, a), (a, a, b), (a, a, c), (a, b, a), (a, b, b), (a, b, c), (a, c, a), \\ &\quad (a, c, b), (a, c, c), (b, c, b), (c, c, b)\} \end{aligned}$$

Now, $\dot{\pi}_{\langle X, Z \rangle}(R \bowtie S)$ becomes the paraconsistent relation on scheme $\langle X, Z \rangle$:

$$\begin{aligned} \dot{\pi}_{\langle X, Z \rangle}(R \bowtie S)^+ &= \{(b, a)\} \\ \dot{\pi}_{\langle X, Z \rangle}(R \bowtie S)^- &= \{(a, a), (a, b), (a, c)\} \end{aligned}$$

Finally, $\dot{\sigma}_{-X=Z}(\dot{\pi}_{\langle X, Z \rangle}(R \bowtie S))$ becomes the paraconsistent relation on scheme $\langle X, Z \rangle$:

$$\begin{aligned} \dot{\sigma}_{-X=Z}(\dot{\pi}_{\langle X, Z \rangle}(R \bowtie S))^+ &= \{(b, a)\} \\ \dot{\sigma}_{-X=Z}(\dot{\pi}_{\langle X, Z \rangle}(R \bowtie S))^- &= \{(a, a), (a, b), (a, c)(b, b), (c, c)\} \end{aligned}$$

□

In the rest of the paper, relations mean paraconsistent relations. In order to find p-minimal models easily in our algorithm, we create a copy for a given relation. For any given relation R , the copy of R is R' . Both R and R' are different relations with the same attributes and the same tuples. R is called an exact relation and R' is called a copy relation. In addition to this, the replica of the relation R is R , where replica R has the same name, the same tuples, and the same attributes. We assume that a relation and its replica can not appear in the same set, but can appear in different sets. If two relations (a relation and its replica) appear in the same set, then we merge the tuples of them and write it as one relation.

In the next section, we explain the disjunctive relation model, which is an adaptation of disjunctive relational model introduced by Jayakumar and Sunderraman [15].

3 Disjunctive Relation

Let a disjunctive relation scheme 2^Σ be a finite set of attribute sets, where for any attribute set $A \in 2^\Sigma$, $\text{dom}(a)$ is a non-empty domain of values for each

$a \in A$. Let $\tau(2^\Sigma)$ denote the set of all tuples on 2^Σ . A disjunctive relation on scheme 2^Σ is thus any subset of $\tau(2^\Sigma)$. A disjunctive relation, DR , over the scheme 2^Σ consists of two components $\langle DR^+, DR^- \rangle$, where $DR^+ \subseteq \tau(2^\Sigma)$ and $DR^- \subseteq \tau(2^\Sigma)$. DR^+ is the component that consists of a set of tuples. Each tuple in this component represents a disjunction of facts. In the case where the tuple is a singleton, we have a definite fact. DR^- is the component that consists of a set of tuples. Each tuple in this component represents a conjunction of facts. In the case where the tuple is a singleton, we have a definite fact. Let T be a tuple in DR , then for all $t \in T$, $Att(t)$ is an attribute set that represents the element in the tuple of the disjunctive relation DR , and let $Att(R)$ be an attribute set of relation R over the scheme Σ .

In the remainder of the section, we define a rename operator, mapping, and necessary definitions, which play a key role in constructing the p-minimal models.

Rename Operator. Rename operator renames the attributes for any relations.

Attribute Rename (Θ). Let R be a relation over scheme Σ and $\Sigma' = \{A_1 \dots A_m, R.A_1 \dots R.A_m\}$. Then,

$$\Theta_{A_1 \dots A_m \rightarrow R.A_1 \dots R.A_m}(R) \text{ and } \Theta_{R.A_1 \dots R.A_m \rightarrow A_1 \dots A_m}(R).$$

This operator (Θ) is used to maintain uniqueness of attributes between any two relations.

Tuple Mapping to Disjunctive Relation. The algebraic equivalent for disjunction (\vee) is union. So, we represent the disjunctive information in P as paraconsistent unions ($\dot{\cup}$) of relations. However, it is not very flexible to construct p-minimal models with paraconsistent unions ($\dot{\cup}$) of relations. So, we map the information in relations to a disjunctive relation DR . Let $R_1 \dots R_n$ be relations over schemes $\Sigma_1 \dots \Sigma_n$ where every $\Sigma_i \subseteq \Sigma$ and $1 \leq i \leq n$. Then a set of attribute sets for any DR that refers $R_1 \dot{\cup} \dots \dot{\cup} R_n$ is $\{\Sigma_1 \dots \Sigma_n\}$. Next, we map the tuples of relations containing paraconsistent unions to a disjunctive relation. For each $t \in T$, T is a tuple for any disjunctive relation (DR). Then $t: \Sigma \rightarrow \cup_{A \in Att(R_i)} dom(A)$ such that $t(A) \in dom(A)$ for every i in $R_1 \dot{\cup} \dots \dot{\cup} R_n$ where $Att(t) = Att(R_i)$. Informally, a disjunctive relation can be considered as a collection of relations. It is intuitive to map each disjunctive relation back to base relations because every $t \in T$ of the disjunctive relation represents the corresponding tuple in the relation.

The following example [15] is very specific, but helps to understand the algorithm clearly.

Example 2. Let R_1 , R_2 and C are relations over schemes $\{X\}$, $\{Y, Z\}$ and $\{X, Y, Z\}$ and domain for every attribute is $\{a, b, c\}$. Then, we have the following equation:

$$(\dot{\pi}_{\{X, Y, Z\}}(R_1(X) \dot{\cup} R_2(Y, Z)))[X, Y, Z] = (\dot{\pi}_{\{X, Y, Z\}}(C(X, Y, Z)))^+[X, Y, Z]$$

where $C^+ = \{(a, b, c)\}$, $R_1^- = \{(b)\}$ and $R_2^+ = \{(a, c), (b, c)\}$

Solution. Before the tuples of C are distributed to R_1 and R_2 , it is imperative to note that R_1 and R_2 contain definite tuples, which are not disjunctive (conjunctive). The first step is to map the definite tuples of R_1 and R_2 to a disjunctive relation. The definite tuples have no disjunction (conjunction) in any disjunctive relation. So, we rename the attributes (Θ) of R_1 and R_2 . Then we map the definite tuples to DR .

In the rest of the paper, we differentiate positive and negative parts of a relation (disjunctive relation) with a double line in every relation (disjunctive relation) diagram. Also, we call relations in left hand side of the equation as base relations.

$$DR = \begin{array}{|c|c|} \hline \{R_1.X\} & \{R_2.Y, R_2.Z\} \\ \hline (b) & \\ \hline & (a, c) \\ \hline & (b, c) \\ \hline \end{array} \quad \text{The next step is to distribute the tuples from}$$

C to each individual relation in any union after applying Θ to R_1 and R_2 . It is necessary to apply Θ before the distribution of tuples from C because we changed the attributes of R_1 and R_2 before we map the definite tuples.

$$R_1 = \begin{array}{|c|} \hline \{X\} \\ \hline (a) \\ \hline (b) \\ \hline \end{array} \quad \text{and} \quad \neg R_2 = \begin{array}{|c|} \hline \{Y, Z\} \\ \hline (b, c) \\ \hline (a, c) \\ \hline (b, c) \\ \hline \end{array}$$

The next step is to again rename (Θ) the attributes.

$$R_1 = \begin{array}{|c|} \hline \{R_1.X\} \\ \hline (a) \\ \hline (b) \\ \hline \end{array} \quad \text{and} \quad \neg R_2 = \begin{array}{|c|} \hline \{R_2.Y, R_2.Z\} \\ \hline (b, c) \\ \hline (a, c) \\ \hline (b, c) \\ \hline \end{array}$$

Then we map the newly added tuples of $R_1 \cup \neg R_2$ to DR .

$$DR = \begin{array}{|c|c|} \hline \{R_1.X\} & \{R_2.Y, R_2.Z\} \\ \hline (a) & \vee & (b, c) \\ \hline (b) & & \\ \hline & (a, c) \\ \hline & (b, c) \\ \hline \end{array}$$

□

This state of DR is base DR . To reiterate, DR^+ contains tuples which in turn contain disjunction. From the base DR , multiple DR can be obtained by applying disjunction in tuples. Each newly created DR from the base DR should not lose any tuple set; otherwise, it leads to incorrect models. The following definition addresses the issue.

Proper Disjunctive Relation (PDR). Let DR be a base disjunctive relation. A proper disjunctive relation is a set, which contains all disjunctive relations that can be formed from DR by applying disjunction in tuples. Concretely, for every disjunctive relation (DR_i), which is obtained from DR by applying disjunction, $\tau(DR^+) = \tau(DR_i^+)$ where $1 \leq i \leq (2^n - 1)^{\tau(DR^+)}$ such DR_i is a PDR^i .

Example 3. Continuing from Example 2.

Solution. The next step is to create a set of proper disjunctive relation from DR .

$$PDR = \{PDR^1, PDR^2, PDR^3\} \text{ where } PDR^1 =$$

$\{R_1.X\}$	$\{R_2.Y, R_2.Z\}$
(a)	
(b)	
	(a, c)
	(b, c)

 $PDR^2 =$

$\{R_1.X\}$	$\{R_2.Y, R_2.Z\}$
	(b, c)
(b)	
	(a, c)
	(b, c)

 $PDR^3 =$

$\{R_1.X\}$	$\{R_2.Y, R_2.Z\}$
(a)	\vee (b, c)
(b)	
	(a, c)
	(b, c)

The size of PDR is 3. Correspondingly, there should be three replicas of base relations. We sometimes superscript the set with a number in order to show the difference between any two sets that looks the same.

$$\begin{aligned} & \{(\pi_{\{X,Y,Z\}}(R_1(R_1.X) \dot{\cup} R_2(R_2.Y, R_2.Z)))[X, Y, Z]\}^1, \\ & \{(\pi_{\{X,Y,Z\}}(R_1(R_1.X) \dot{\cup} R_2(R_2.Y, R_2.Z)))[X, Y, Z]\}^2, \\ & \{(\pi_{\{X,Y,Z\}}(R_1(R_1.X) \dot{\cup} R_2(R_2.Y, R_2.Z)))[X, Y, Z]\}^3. \end{aligned}$$

For every p in PDR , reverse map tuples to a set of base relations.

Finally, rename (Θ) each attribute name of every relation back to its old name in every replica. Hence, R_1 attribute is $\langle X \rangle$ and R_2 attribute is $\langle Y, Z \rangle$. \square

To individualize the relation, we have the following definition.

Relationalize. Let $R_1 \dot{\cup} R_2 \dot{\cup} \dots \dot{\cup} R_n$ and $R_1, R_2 \dots R_n$ be relations on scheme Σ .

$$Relationalize(\pi_{\{\Sigma\}}(R_1 \dot{\cup} R_2 \dot{\cup} \dots \dot{\cup} R_n)[\Sigma]) := \{R_1, R_2 \dots R_n\}$$

The relationalize operator removes the unions from relations and the projection for the expression. By doing so, the operator produces a set of relations. If there is a select operation associated with the expression, then apply the operation before *Relationalize* is applied. *Relationalize* is in accordance to $\{l_i\}$ (defined in Preliminaries section) [17].

Example 4. Continuing from Example 3. In this example, we relationalize only one replica $(\{(\pi_{\{X,Y,Z\}}(R_1(X) \dot{\cup} R_2(Y, Z)))[X, Y, Z]\}^1)$.

Solution. $\text{Relationalize}(\dot{\pi}_{\{X,Y,Z\}}(R_1 \dot{\cup} \dot{R}_2)[X, Y, Z]) = \{R_1, R_2\}$ where

$$R_1 = \begin{array}{|c|} \hline \{X\} \\ \hline (a) \\ \hline (b) \\ \hline \end{array} \text{ and } \dot{R}_2 = \begin{array}{|c|} \hline \{Y, Z\} \\ \hline (a, c) \\ \hline (b, c) \\ \hline \end{array} \text{ or } R_2 = \begin{array}{|c|} \hline \{Y, Z\} \\ \hline (a, c) \\ \hline (b, c) \\ \hline \end{array}$$

□

During p-minimal models construction, we encounter a set of redundant relation sets. In order to remove it, we define the following.

Minimize. Let $\{R_{1_1} \dots R_{1_m}\}$ and $\{R_{2_1} \dots R_{2_n}\}$ be two sets of relations where $m \leq n$.

$$\begin{aligned} \text{Minimize}(\{\{R_{1_1} \dots R_{1_n}\}, \{R_{2_1} \dots R_{2_m}\}\}) &:= \{\{R_{1_1} \dots R_{1_m}\} \mid R_{1_i} = R_{2_j} \\ &\wedge \text{Att}(R_{1_i}) = \text{Att}(R_{2_j}) \wedge \tau(R_{1_i}) = \tau(R_{2_j}) \text{ such that } \forall i, 1 \leq i \leq m \wedge \exists j, 1 \leq j \leq n\}. \end{aligned}$$

By using the definitions and operators in this section, we propose an algorithm in the following section.

4 P-Minimal Models for Positive Extended Disjunctive Deductive Databases

By using the algebra of the relational model, we present a bottom up method for constructing p-minimal models for the positive extended disjunctive deductive database. The algorithm that we present in this section is an extension of the algorithm proposed by Bagai and Sunderraman [5]. The reader is requested to refer to p-minimal models [17] and paraconsistent logics [18] in order to supplement additional knowledge. P-minimal models construction involves two steps. The first step is to convert P into a set of relation definitions for the predicate symbols occurring in P . These definitions are of the form

$$U_r = D_{U_r}$$

where U_r is the paraconsistent union of disjunctive head predicate symbols of P , and D_{U_r} is an algebraic expression involving predicate symbols of P . Here r refers to the equation number, $1 \leq r \leq N$, where N refers to a total number of equations. The second step is to iteratively evaluate the expressions in these definitions to incrementally construct the relations associated with the predicate symbols. The first step is called SERIALIZE and the second step is called Model Construction.

Algorithm. SERIALIZE

Input. A positive extended disjunctive deductive database clause $l_0 \vee \dots \vee l_n \leftarrow l_{n+1} \dots l_m$. For any i , $0 \leq i \leq m$, l_i is either of the form $p_i(A_{i1} \dots A_{ik_i})$ or $\neg p_i(A_{i1} \dots A_{ik_i})$, and let V_i be the set of all variables occurring in l_i .

Output. An algebraic expression involving paraconsistent relations.

Method. The expression is constructed by the following steps:

1. For each argument A_{ij} of literal l_i , construct argument B_{ij} and condition C_{ij} as follows:
 - (a) If A_{ij} is a constant a , then B_{ij} is any brand new variable and C_{ij} is $B_{ij} = a$.
 - (b) If A_{ij} is a variable, such that for each k , $1 \leq k < j$, $A_{ik} \neq A_{ij}$, then B_{ij} is A_{ij} and C_{ij} is true.
 - (c) If A_{ij} is a variable, such that for some k , $1 \leq k < j$, $A_{ik} = A_{ij}$, then B_{ij} is a brand new variable and C_{ij} is $A_{ij} = B_{ij}$.
2. Let \hat{l}_i be the atom $p_i(B_{i1} \dots B_{ik_i})$, and F_i be the conjunction $C_{i1} \wedge \dots \wedge C_{ik_i}$. If l_i is a positive literal, then Q_i is the expression $\dot{\pi}_{V_i} \dot{\sigma}_{F_i}(\hat{l}_i)$. Otherwise, let Q_i be the expression $\dot{\neg} \dot{\pi}_{V_i}(\dot{\sigma}_{F_i}(\hat{l}_i))$.
 As a syntatic optimisation, if all conjuncts of F_i are true (i.e. all arguments of l_i are distinct variables), then both $\dot{\sigma}_{F_i}$ and $\dot{\pi}_{V_i}$ are reduced to identity operations, and are hence dropped from the expression $\dot{\sigma}_{F_i}$.
3. Let U be the union ($\dot{\cup}$) of the Q_i 's thus obtained, $0 \leq i \leq n$. The output expression is $(\dot{\sigma}_{F_1}(\dot{\pi}_{DV}(U))) [B_{01} \dots B_{n_{kn}}]$ where DV is the set of distinct variables occurring in all l_i .
4. Let E be the natural join ($\dot{\bowtie}$) of the Q_i 's thus obtained, $n+1 \leq i \leq m$. The output expression is $(\dot{\sigma}_{F_1}(\dot{\pi}_{DV}(E))) [B_{01} \dots B_{n_{kn}}]$. As in step 2, if all conjuncts are true, then $\dot{\sigma}_{F_1}$ is dropped from the output expression.

From the algebraic expression of the algorithm, we then construct a system of equations.

For any positive extended disjunctive deductive database P , $\text{EQN}(P)$ is a set of all equations of the form $U_r = D_{U_r}$, where U_r is a union of the head predicate symbols of P , and D_{U_r} is the paraconsistent union ($\dot{\cup}$) of all expressions obtained by the algorithm *SERIALIZE* for clauses in P with the same U_r in their head. If all literals in the head are the same for any two rules, then U_r is the same for the two rules.

The final step is then to construct the model by incrementally constructing the relation values in P . For any positive extended disjunctive deductive database, P_E is the non disjunctive-facts (clauses in P without bodies), and P_B is the disjunctive rules (clauses in P with bodies). P_E^* refers to a set of all ground instances of clauses in P_E . Then, $P_I = P_E^* \cup P_B$.

The following algorithm finds p-minimal models for P .

ALGORITHM. Model Construction

Input. A positive extended disjunctive deductive database (P)

Output. P-minimal models for P .

Method: The values are computed by the following steps:

1. (Initialization)
 - (a) Compute $\text{EQN}(P_I)$ using the algorithm *SERIALIZE* for each clause in P_I .

- (b) SModel = \emptyset , For each predicate symbol p in P_E , set

$$p^+ = \{(a1 \dots ak) \mid p(a1 \dots, ak) \in P_E^*\}, \text{ and } p^- = \emptyset \text{ or} \\ p^- = \{(a1 \dots ak) \mid \neg p(a1, \dots ak) \in P_E^*\} \text{ and } p^+ = \emptyset$$

SModel = p

End for.

2. (Rule Application)

- (a) For every SModel (SModel $\neq \emptyset$), create copies of the relations in SModel and replace the SModel with the copies. DModel = \emptyset .
- (b) For every equation r of the form $U_r = D_{U_r}$, create DR_r and insert the tuples from the copies in SModel into the corresponding exact relation in the equation r . Apply Θ to every relation in U_r and map the definite tuples for the relations in U_r to DR_r . Again, apply Θ to every relation in U_r . Compute the expression D_{U_r} and set the relations in U_r with $D_{U_r}^+$.
- (c) Apply Θ to every relation in U_r , map the newly added tuples of U_r to DR_r and create a set of proper disjunctive relations (PDR_r) from the DR_r .
- (d) Delete all tuples for the relations in U_r and create multiple replicas of U_r , which is denoted by the set C_r , where $|C_r| = |PDR_r|$.
- (e) Re-map each p in PDR_r to C where $C \in C_r$.
 For every $C \in C_r$,
 $C = \text{Relationalize}(C)$
 For every $R \in C$
 $R = \Theta(R)$
 End For.
 End For.
 DModel = DModel $\cup C_r$ /* Merging relations of every equation */
- (f) Once all equations are evaluated for the current SModel, perform the following: (i) for every $M \in \text{DModel}$ and for every exact relation for SModel that is not in M , create the exact relation in M ; and (ii) for every $M \in \text{DModel}$ and for every exact relation for SModel that is in M , insert the tuples from the copy relation in SModel into the exact relation of M . Then add DModel to TempModel.
- (g) Once every SModel is applied, start from step 2 (a) with
 SModel = $\text{Minimize}(\text{TempModel})$ and stop when there is no change in SModel.

3. P-models: rewrite the set of relations in SModel as a set of literals. P-minimal models = $\min(\text{P-models})$ ($\min()$ is defined in Preliminaries).

It is very intuitive from the algorithm that if the computation of D_{U_r} is empty for any SModel, then discard the SModel. We found that the algorithm should be extended a little to accommodate for disjunctive facts, duplicate variables in disjunctive literals, and constants in disjunctive literals.

The following example shows that how the algorithm works.

Example 5. Let P be a positive extended disjunctive deductive database. It has the following facts and rules:

$$\begin{aligned} & r(a, c), p(a), p(c), \neg f(a, b), s(c) \\ & g(X) \vee \neg p(X) \leftarrow r(X, Y), s(Y) \\ & g(X) \vee \neg p(X) \leftarrow \neg f(X, Y) \end{aligned}$$

Solution. After step 1 (a) in initialization, EQN(P_I) returns:

$$(U_1)(\dot{\pi}_{\{X\}}(g(X) \dot{\cup} \neg p(X))[X] = (\dot{\pi}_{\{X\}}(r(X, Y) \dot{\bowtie} s(Y)))^+[X] \dot{\cup} (\dot{\pi}_{\{X\}}(\neg f(X, Y)))^+[X])$$

After step 1 (b) in initialization, SModel = $\{r, p, s, f\}$ where $r = \frac{\frac{\{X, Y\}}{(a, c)}}{(a, c)} p = \frac{\frac{\{X\}}{(a)}}{(c)}$

$$s = \frac{\{Y\}}{(c)} f = \frac{\frac{\{X, Y\}}{(a, b)}}{(a, b)}$$

After step 2 (a), SModel = $\{r', p', s', f'\}$ (COPIES) where

$$r' = \frac{\frac{\{X, Y\}}{(a, c)}}{(a, c)} p' = \frac{\frac{\{X\}}{(a)}}{(c)} s' = \frac{\{Y\}}{(c)} f' = \frac{\frac{\{X, Y\}}{(a, b)}}{(a, b)}$$

In step 2 (b), there is only one SModel and one equation. It is necessary to insert the tuples from the copies in SModel to the corresponding relations in the equation. DModel = \emptyset . Then map the definite tuples to DR_1 for the current SModel. Compute the expression and assign it to U_1 .

$$DR_1 = \frac{\frac{\{g.X\}}{(a)} \{p.X\}}{(c)}$$

By step 2 (c), map the newly added (disjunctive) tuples to DR_1 .

$$DR_1 = \frac{\frac{\{g.X\}}{(a)} \{p.X\}}{(a) \vee (a)} = \frac{\frac{\{g.X\}}{(a)} \{p.X\}}{(a)} \vee \frac{\frac{\{g.X\}}{(a)} \{p.X\}}{(a)}$$

$$PDR_1 = \{PDR_1^1, PDR_1^2, PDR_1^3\}$$

$$PDR_1^1 = \frac{\frac{\{g.X\}}{(a)} \{p.X\}}{(a) \vee (a)} PDR_1^2 = \frac{\frac{\{g.X\}}{(a)} \{p.X\}}{(a)} PDR_1^3 = \frac{\frac{\{g.X\}}{(a)} \{p.X\}}{(a)}$$

We skip a step (2 (d)) here. Map every p in PDR_1 back to a set of base relation. We write after relationalizing the set of relations and applying Θ (step 2 (e)).

$$C_1 = \{\{g, p\}^1, \{p\}^2, \{g, p\}^3\}$$

$$\{g, p\}^1 g \frac{\frac{\{X\}}{(a)}}{(a)} p \frac{\frac{\{X\}}{(a)}}{(c)} \{p\}^2 p \frac{\frac{\{X\}}{(a)}}{(c)} \{g, p\}^3 g \frac{\frac{\{X\}}{(a)}}{(a)} p \frac{\frac{\{X\}}{(a)}}{(c)}$$

DModel = DModel \cup C_1

By step 2 (f), DModel = $\{\{g, p, r, s, f\}^1, \{p, r, s, f\}^2, \{g, p, r, s, f\}^3\}$

$$\begin{aligned}
 \{g, p, r, s, f\}^1 & g \frac{\frac{\{X\}}{(a)}}{(a)} p \frac{\frac{\{X\}}{(a)}}{(c)} r \frac{\frac{\{X, Y\}}{(a, c)}}{(a, c)} s \frac{\frac{\{Y\}}{(c)}}{(c)} f \frac{\frac{\{X, Y\}}{(a, b)}}{(a, b)} \\
 \{p, r, s, f\}^2 & p \frac{\frac{\{X\}}{(a)}}{(c)} r \frac{\frac{\{X, Y\}}{(a, c)}}{(a, c)} s \frac{\frac{\{Y\}}{(c)}}{(c)} f \frac{\frac{\{X, Y\}}{(a, b)}}{(a, b)} \\
 \{g, p, r, s, f\}^3 & g \frac{\frac{\{X\}}{(a)}}{(a)} p \frac{\frac{\{X\}}{(a)}}{(c)} r \frac{\frac{\{X, Y\}}{(a, c)}}{(a, c)} s \frac{\frac{\{Y\}}{(c)}}{(c)} f \frac{\frac{\{X, Y\}}{(a, b)}}{(a, b)}
 \end{aligned}$$

Add DModel to TempModel.

By step 2 (g), SModel = *Minimize*(TempModel). The algorithm stops when there is no change in SModel. We skip further iterations and go to the final step (3). In the final step, we first rewrite the relation in the form of literals,

$$\begin{aligned}
 \text{P-models} = & \{\{g(a), p(a), p(c), \neg p(a), r(a, c), s(c), \neg f(a, b)\}, \{p(a), p(c), \neg p(a), \\
 & r(a, c), s(c), \neg f(a, b)\}, \{g(a), p(a), p(c), r(a, c), s(c), \neg f(a, b)\}\}.
 \end{aligned}$$

Then, p-minimal models = $\{\{p(a), p(c), \neg p(a), r(a, c), s(c), \neg f(a, b)\}, \{g(a), p(a), p(c), r(a, c), s(c), \neg f(a, b)\}\}$. This result is the same for **fixed-point semantics** defined in Preliminaries section.

5 Conclusion

In this paper, we proposed an algorithm to find p-minimal models for any positive extended disjunctive deductive database. We also used a disjunctive relational model to represent the relations containing paraconsistent unions [15].

Though we find the model for any given positive extended disjunctive deductive database, the algorithm does not find models for the databases with recursions and constraints, which could be a good future work for this algorithm. It would be very interesting to analyze the algorithm by allowing default negation in program P . We observe that we have not proven the correctness and complexities of the algorithm. We have also left that for future work. In query-intensive applications, this precomputation of the model enables efficient processing of subsequent queries. The creation of many proper disjunctive databases are expensive, given the p-minimal models computation, and are probably not worth the extra computation.

References

1. Alcântara, J., Damásio, C.V., Pereira, L.M.: A declarative characterisation of disjunctive paraconsistent answer sets. In: ECAI, vol. 16, p. 951. Citeseer (2004)
2. Alcântara, J., Damásio, C.V., Moniz, L.M.: Paraconsistent logic programs. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 345–356. Springer, Heidelberg (2002)
3. Arieli, O.: Paraconsistent declarative semantics for extended logic programs. *Ann. Math. Artif. Intell.* **36**(4), 381–417 (2002)
4. Arieli, O.: Distance-based paraconsistent logics. *Int. J. Approximate Reasoning* **48**(3), 766–783 (2008)
5. Bagai, R., Sunderraman, R.: A paraconsistent relational data model. *Int. J. Comput. Math.* **55**(1–2), 39–55 (1995)
6. Bagai, R., Sunderraman, R.: Bottom-up computation of the fitting model for general deductive databases. *J. Intell. Inf. Syst.* **6**(1), 59–75 (1996)
7. Bagai, R., Sunderraman, R.: Computing the well-founded model of deductive databases. *Int. J. Uncertainty Fuzziness Knowl. Based Syst.* **4**(2), 157–175 (1996)
8. Belnap Jr., N.D.: A useful four-valued logic. In: Michael Dunn, J., Epstein, G. (eds.) *Modern Uses of Multiple-Valued Logic*, pp. 5–37. Springer, Netherlands (1977)
9. Blair, H.A., Subrahmanian, V.: Paraconsistent logic programming. *Theor. Comput. Sci.* **68**(2), 135–154 (1989)
10. Carnielli, W., Coniglio, M.E., Marcos, J.: Logics of formal inconsistency. In: Gabbay, D.M., Guenther, F. (eds.) *Handbook of Philosophical Logic*, pp. 1–93. Springer, Netherlands (2007)
11. Damásio, C.V., Pereira, L.M.: A survey of paraconsistent semantics for logic programs. In: Besnard, P., Hunter, A. (eds.) *Reasoning with Actual and Potential Contradictions*, pp. 241–320. Springer, Netherlands (1998)
12. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. *New Gener. Comput.* **9**(3–4), 365–385 (1991)
13. Hunter, A.: Paraconsistent logics. In: Besnard, P., Hunter, A. (eds.) *Reasoning with Actual and Potential Contradictions*, pp. 11–36. Springer, Netherlands (1998)
14. Hunter, A.: Reasoning with contradictory information using quasi-classical logic. *J. Logic Comput.* **10**(5), 677–703 (2000)
15. Jayakumar, B., Sunderraman, R.: Paraconsistent relational model: a quasi-classic logic approach. In: *IJCAI Workshop 13 Ontologies and Logic Programming for Query Answering*, Buenos Aires, Argentina, pp. 82–90, July 2015
16. Minker, J., Seipel, D.: Disjunctive logic programming: a survey and assessment. In: Kakas, A.C., Sadri, F. (eds.) *Computational Logic: Logic Programming and Beyond*. LNCS (LNAI), vol. 2407, pp. 472–511. Springer, Heidelberg (2002)
17. Sakama, C., Inoue, K.: Paraconsistent stable semantics for extended disjunctive programs. *J. Logic Comput.* **5**(3), 265–285 (1995)
18. Subrahmanian, V.: Paraconsistent disjunctive deductive databases. In: *Proceedings of the Twentieth International Symposium on Multiple-Valued Logic*, pp. 339–346. IEEE (1990)
19. Zhang, Z., Lin, Z., Ren, S.: Quasi-classical model semantics for logic programs – a paraconsistent approach. In: Rauch, J., Raś, Z.W., Berka, P., Elomaa, T. (eds.) *ISMIS 2009*. LNCS, vol. 5722, pp. 181–190. Springer, Heidelberg (2009)

Multi-disciplinary Trends in Artificial Intelligence
9th International Workshop, MIWAI 2015, Fuzhou, China,
November 13-15, 2015, Proceedings
Bikakis, A.; Zheng, X. (Eds.)
2015, XX, 458 p. 167 illus. in color., Softcover
ISBN: 978-3-319-26180-5