Extended Well-Founded Semantics for Paraconsistent Logic Programs *

Chiaki Sakama

ASTEM Research Institute of Kyoto
17 Chudoji Minami-machi, Shimogyo, Kyoto 600 Japan
sakama@astem.or.jp

Abstract

This paper presents a declarative semantics of logic programs which possibly contain inconsistent information. We introduce a multi-valued interpretation of logic programs and present the extended well-founded semantics for paraconsistent logic programs. In this setting, a meaningful information is still available in the presence of an inconsistent information in a program and any fact which is affected by an inconsistent information is distinguished from the others. The well-founded semantics is also extended to disjunctive paraconsistent logic programs.

1 Introduction

Recent studies have greatly enriched an expressive power of logic programming as a tool for knowledge representation. Handling classical negation as well as negation by failure in a program is one of such extension. An *extended logic program*, which is introduced by Gelfond and Lifschitz [GL90], distinguishes two types of negation and enables us to deal with explicit negation as well as default negation in a program. An extended logic program is, however, possibly inconsistent in general, since it contains negative heads as well as positive ones in program clauses. Practically, an inconsistency is likely to happen when we build a large scale of knowledge base in such a logic program. A knowledge base may contain local inconsistencies that would make a program contradictory and yet it may have a natural intended global meaning. However, in an inconsistent program, the answer set semantics proposed in [GL90] implies every formula from the program. This is also the case for most of the traditional logics in which a piece of inconsistent information might spoil the rest of the whole knowledge base.

*In *Proceedings of the 1992 International Conference on Fifth Generation Computer Systems* (FGCS'92), Ohmsha, pp. 592-599, 1992.

To avoid such a situation, the so-called *paraconsistent logics* have been developed which are not destructive in the presence of an inconsistent information [Co74]. From the point of view of logic programming, a possibly inconsistent logic program is called a *paraconsistent logic program*. Blair and Subrahmanian [BS87] have firstly developed a fixpoint semantics of such programs by using Belnap's four-valued logic [Be75]. Recent studies such as [KL89, Fi89, Fi91] have also developed a logic for possibly inconsistent logic programs and provided a framework for reasoning with inconsistency. However, from the point of view of logic programming, negation in these approaches is classical in its nature and the treatment of default negation as well as classical one in paraconsistent logic programming is still left open.

In this paper, we present a framework for paraconsistent logic programming in which classical and default negation are distinguished. The rest of this paper is organized as follows. In Section 2, we first present an application of Ginsberg's lattice-valued logic to logic programming and provide a declarative semantics of paraconsistent logic programs by extending the well-founded semantics of general logic programs. Then we show how the extended well-founded semantics isolates an inconsistent information and distinguishes meaningful information from others in a program. In Section 3, the well-founded semantics is also extended to paraconsistent disjunctive logic programs. Section 4 discusses related work and Section 5 concludes this paper.

2 Well-Founded Semantics for Paraconsistent Logic Programs

2.1 Multi-valued Logic

To present the semantics of possibly inconsistent logic programs, multi-valued logics are often used instead of the traditional two-valued logic. Among them, Belnap's four-valued logic [Be75] is well-known and several researchers have employed this logic to give the semantics of paraconsistent logic programs [BS87, KL89, Fi89, Fi91]. In Belnap's logic, truth values consist of $\{\mathbf{t}, \mathbf{f}, \top, \bot\}$ in which each element respectively denotes *true, false, contradictory*, and *undefined*. Each element makes a complete lattice under a partial ordering defined over these truth values (figure 1).

To represent nonmonotonic aspect of logic programming, however, we need extra truth values which represent default assumption. Such a logic is firstly introduced by Ginsberg [Gi86] in the context of bilattice for default logic. We use this logic to give the semantics of paraconsistent logic programs.[1]

The set $\mathbf{VII} = \{\mathbf{t}, \mathbf{f}, \mathbf{dt}, \mathbf{df}, *, \top, \bot\}$ is the space of truth values in our

[1] [KL89] has also suggested the extensibility of their logic for handling defaults by using Ginsberg's lattice-valued logic.

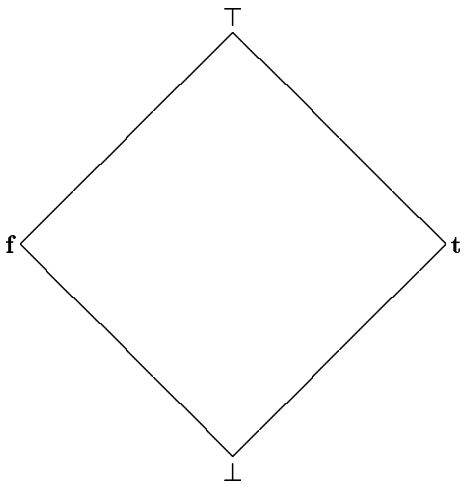

Figure 1: Four-valued logic

seven-valued logic. Here, additional elements **dt**, **df**, and *, are read as *true by default*, *false by default*, and *don't-care by default*, respectively. In **VII**, each element makes a complete lattice under the ordering $\preceq$ such that: $\forall \mathbf{x} \in \mathbf{VII}, \mathbf{x} \preceq \mathbf{x}$ and $\bot \preceq \mathbf{x} \preceq \top$; and for $\mathbf{x} \in \{\mathbf{t}, \mathbf{f}\}, \mathbf{dx} \preceq * \preceq \mathbf{x}$ (figure 2).

A *program* is a (possibly infinite) set of clauses of the form:

$$A \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n$$

where $m, n \geq 0$, each A, $B_i (1 \leq i \leq m)$ and $C_j (1 \leq j \leq n)$ are literals and all the variables are assumed to be universally quantified at the front of the clause. In a program, two types of negation are distinguished; hereafter, $\neg$ denotes a monotonic classical negation, while *not* denotes a nonmonotonic default negation. A *ground clause* (resp. *program*) is a clause (resp. program) in which every variable is instantiated by the elements of the Herbrand universe of a program. Also, such an instantiation is called *Herbrand instantiation* of a clause (resp. program).

An *interpretation* I of a program is a function such that $I : H_B \rightarrow \mathbf{VII}$ where H_B is the Herbrand base of the program. (Throughout of this paper, H_B denotes the Herbrand base of a program.)

A *formula* is defined as usual; (i) any literal L or $\neg L$ is a formula, (ii) for any literal L, $not\, L$ and $not \neg L$ are formulas, and (iii) for any formula F and G, $\forall F$, $\exists F$, $F \vee G$, $F \wedge G$ and $F \leftarrow G$ are all formulas. A formula is *closed* if it contains no free variable. Satisfaction of a formula is also defined as follows.

Definition 2.1 Let P be a program and I be its interpretation. Suppose $I \models F$ denotes that I *satisfies* a formula F, then:

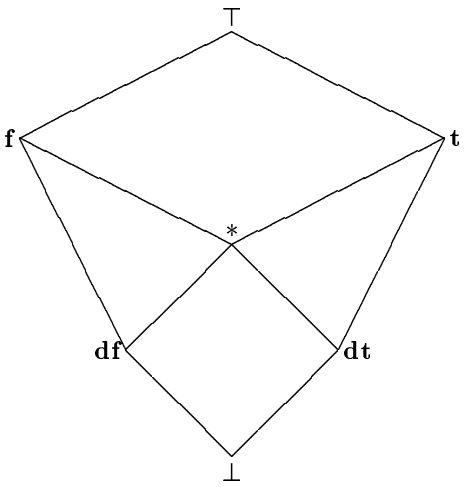

Figure 2: The logic **VII**

1. For any atom $A \in H_B$,
   (a) $I \models A$ if $\mathbf{t} \preceq I(A)$,
   (b) $I \models \neg A$ if $\mathbf{f} \preceq I(A)$,
   (c) $I \models not\, A$ if $\mathbf{df} \preceq I(A) \preceq *$,
   (d) $I \models not \neg A$ if $\mathbf{dt} \preceq I(A) \preceq *$.
2. For any closed formula $\exists F$ (resp. $\forall F$), $I \models \exists F$ (resp. $I \models \forall F$) if $I \models F'$ for some (resp. every) Herbrand instantiation F' of F.
3. For closed formulas F and G,
   (a) $I \models F \vee G$ if $I \models F$ or $I \models G$,
   (b) $I \models F \wedge G$ if $I \models F$ and $I \models G$,
   (c) $I \models F \leftarrow G$ if $I \models F$ or $I \not\models G$. □

The ordering $\preceq$ on truth values is also defined between interpretations. For interpretations I_1 and I_2, $I_1 \preceq I_2$ iff $\forall A \in H_B, I_1(A) \preceq I_2(A)$. An interpretation I is called *minimal*, if there is no interpretation J such that $J \neq I$ and $J \preceq I$. An interpretation I is also called *least*, if $I \preceq J$ for every interpretation J.

An interpretation I is called a *model* of a program if every clause in a program is satisfied in I. Note that in our logic, the notion of model is also defined for an inconsistent set of formulas. For example, a program $\{ p, \neg p \}$ has a model I such that $I(p) = \top$. In particular, an interpretation I of a program is called *consistent* if for every atom A in H_B, $I(A) \neq \top$. A program is called *consistent* if it has a consistent model.

2.2 Extended Well-Founded Semantics

The *well-founded semantics* is known as one of the most powerful semantics which is defined for every general logic program [VRS88, Pr89]. The well-founded semantics has also extended to programs with classical negation in [Pr90], however, it is not well-defined for inconsistent programs in which inconsistent models are all thrown away. In this section, we reformulate the well-founded semantics for possibly inconsistent logic programs.

To compute the well-founded model, we first present an interpretation of a program by a pair of sets of ground literals.

Definition 2.2 For a program P, a pair of sets of ground literals $I =< \sigma; \delta >$ presents an interpretation of P in which each literal in I is interpreted as follows:
For a positive literal L,

(i) if L (resp. $\neg L$) is in σ, L is *true* (resp. *false*) in I;

(ii) else if L (resp. $\neg L$) is in δ, L is *false by default* (resp. *true by default*) in I;

(iii) otherwise, neither L nor $\neg L$ is in σ nor δ, L is *undefined*.

In particular, if both L and $\neg L$ are in σ (resp. δ), L is *contradictory* (resp. *don't-care by default*) in I. □

Intuitively, σ presents proven facts while δ presents default facts, and an interpretation of a fact is defined by the *least upper bound* of its truth values in the pair.

Now we extend the constructive definition of the well-founded semantics for general logic programs [Pr89] to paraconsistent logic programs.

Definition 2.3 Let P be a program and $I =< \sigma; \delta >$ be an interpretation of P. For sets T and F of ground literals, the mapping Φ_I and Ψ_I are defined as follows:

$\Phi_I(T) = \{\ A \mid$ there is a ground clause $A \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n$ from P s.t. $\forall B_i\ (1 \leq i \leq m)\ B_i \in \sigma \cup T$ and $\forall C_j\ (1 \leq j \leq n)\ C_j \in \delta\ \}$,

$\Psi_I(F) = \{\ A \mid$ for every ground clause $A \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n$ from P, either $\exists B_i\ (1 \leq i \leq m)$ s.t. $B_i \in \delta \cup F$ or $\exists C_j\ (1 \leq j \leq n)$ s.t. $C_j \in \sigma\ \}$. □

Definition 2.4 Let I be an interpretation. Then,

$$\begin{aligned} T_I \uparrow 0 &= \emptyset; \\ T_I \uparrow n+1 &= \Phi_I(T_I \uparrow n); \\ T_I &= \bigcup_{n<\omega} T_I \uparrow n; \end{aligned}$$

$$
\begin{aligned}
F_I \downarrow 0 &= H_B \cup \neg H_B \quad \text{where } \neg H_B = \{\neg A \mid A \in H_B\};\\
F_I \downarrow n+1 &= \Psi_I(F_I \downarrow n);\\
F_I &= \bigcap_{n<\omega} F_I \downarrow n. \quad \square
\end{aligned}
$$

As in [Pr89], T_I and F_I are the least fixpoints of the monotonic operators Φ_I and Ψ_I, respectively.

Definition 2.5 For every interpretation I, an operator Θ is defined by:

$$
\begin{aligned}
\Theta(I) &= I \cup < T_I; F_I >;\\
I \uparrow 0 &= < \emptyset; \emptyset >;\\
I \uparrow n+1 &= \Theta(I \uparrow n);\\
M_P &= \bigcup_{n<\omega} I \uparrow n. \quad \square
\end{aligned}
$$

Proposition 2.1 M_P is the least fixpoint of the monotonic operator Θ and also a model of P. $\square$

By definition, M_P is uniquely defined for *every* paraconsistent logic program. We call such an M_P the *extended well-founded model* of a program and the meaning of a program represented by such a model is called the *extended well-founded semantics* of a program.

Note that the original fixpoint definition of the well-founded semantics in [Pr89] is three-valued and defined for general logic programs, while our extended well-founded semantics is seven-valued and defined for extended logic programs. Compared with the three-valued well-founded semantics, the extended well-founded semantics handles positive and negative literals symmetrically during the computation of the fixpoint. Further, the extended well-founded model is the least fixpoint of a program under the ordering $\preceq$, while the three-valued well-founded model is the least fixpoint with respect to the ordering $\mathbf{f} < \bot < \mathbf{t}$, which is basically different from $\preceq$.[2]

Example 2.1 (barber's paradox) Consider the following program:

$$shave(b, x) \leftarrow not\, shave(x, x).$$

Then $shave(b, b)$ is undefined under the three-valued well-founded semantics, while $M_P =< \emptyset; \{\neg shave(b, b)\} >$ then $shave(b, b)$ is true by default under the extended well-founded semantics. In other words, the extended well-founded semantics assumes the fact 'the barber shaves himself' without conflicting the sentence in the program. $\square$

[2] This point is also remarked in [Pr89, Pr90]. In terms of the *bilattice* valued logic [Gi86, Fi91], the ordering $<$ is called a *truth ordering*, while the ordering $\preceq$ is called a *knowledge ordering*.

Also it should be noted that the extended well-founded model is the least fixpoint of a program, but *not* necessarily the least model of the program in general.

Example 2.2 $P = \{\ \neg p \leftarrow not\, p,\quad \neg q \leftarrow \neg p,\quad q \leftarrow\ \}$. Then $M_P =< \{\neg p, q, \neg q\}; \{p\} >$ and the truth value of each predicate is $\{\ p \rightarrow \mathbf{f},\ q \rightarrow \top\ \}$. On the other hand, the least model assigns truth values such as $\{\ p \rightarrow \bot,\ q \rightarrow \mathbf{t}\ \}$. □

In fact, the above least model is not the fixpoint of the program. In this sense, our extended well-founded semantics is different from the least fixpoint model semantics of [BS87] (even for a program without nonmonotonic negation). The difference is due to the fact that in their least fixpoint model semantics each fact which cannot be proved in a program is assumed to be undefined, while it possibly has a default value under the extended well-founded semantics. The above example also suggests the fact that for a consistent program P, M_P is not always consistent.

The extended well-founded semantics is also different from Fitting's bilattice-valued semantics [Fi89, Fi91].

Example 2.3 Let $P = \{\ p \leftarrow q,\quad p \leftarrow \neg q,\quad q \leftarrow\ \}$. Then, as pointed out in [Su90], p is unexpectedly contradictory under Fitting's semantics, while $M_P =< \{p, q\}; \{\neg p, \neg q\} >$ then both p and q are true under the extended well-founded semantics. □

Now we examine the behavior of the extended well-founded semantics more carefully in the presence of an inconsistent information.

Example 2.4 Let $P = \{\ a \leftarrow \neg b,\quad \neg b \leftarrow c \wedge not\, b,\quad c \leftarrow\ \}$. Then $M_P =< \{c, a, \neg b\}; \{b, \neg a, \neg c\} >$. Thus, the truth values of c and a are true, while b is false. □

In the above example, when we consider the program $P' = P \cup \{\ \neg a \leftarrow\ \}$, the truth value of a turns contradictory, while truth values of c and b are unchanged. That is, a meaningful information is still available from the inconsistent program.

On the other hand, when we consider the program $P'' = P \cup \{\ \neg c \leftarrow, d \leftarrow\ \}$, the truth value of c is now contradictory, while a and d are true and b is false. Carefully observing this result, however, the truth of a is now less credible than the truth of d, since a is derived from the fact $\neg b$ which is now supported by the inconsistent fact c in the program.

Such a situation also happens in Blair and Subrahmanian's fixpoint semantics [BS87], in which a truth fact is not distinguished even if it is supported by an inconsistent fact in a program. In the next section, we refine the extended well-founded semantics to distinguish such suspicious truth facts from others.

2.3 Reasoning with Inconsistency

When a program contains an inconsistent information, it is important to detect a fact affected by such an information and distinguish it from other meaningful information in a program. In this section, we present such skeptical reasoning under the extended well-founded semantics.

First we introduce one additional notation. For a program P and each literal L from H_B, L^Γ is called a *suffixed literal* where Γ is a collection of sets of ground literals (possibly preceded by *not*). Informally speaking, each element in Γ presents a set of facts which are used to derive L in P (it is defined more precisely below). An interpretation of such a suffixed literal L^Γ is supposed to be the same with the interpretation of L.

Definition 2.6 Let P be a program and $I =< \sigma; \delta >$ be an interpretation in which σ (resp. δ) is a set of suffixed literals (resp. a set of ground literals). For a set T (resp. F) of suffixed literals (resp. ground literals), the mapping Φ_I^s and Ψ_I^s are defined as follows:

$\Phi_I^s(T) = \{\ A^\Gamma \mid$ there are k ground clauses $A \leftarrow B_{l1} \wedge \ldots \wedge B_{lm} \wedge not\, C_{l1} \wedge \ldots \wedge not\, C_{ln}$ $(1 \leq l \leq k)$ from P s.t. $\forall B_{li}$ $(1 \leq i \leq m)$ $B_{li}^{\Gamma_{li}} \in \sigma \cup T$ and $\forall C_{lj}$ $(1 \leq j \leq n)$ $C_{lj} \in \delta$ and $\Gamma = \bigcup_l \{\{B_{l1}, \ldots, B_{lm}, not\, C_{l1}, \ldots, not C_{ln}\} \cup \gamma_{l1} \cup \ldots \cup \gamma_{lm} \mid \gamma_{li} \in \Gamma_{li}\ \}$.

$\Psi_I^s(F) = \{\ A \mid$ for every ground clause $A \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n$ from P, either $\exists B_i$ $(1 \leq i \leq m)$ s.t. $B_i \in \delta \cup F$ or $\exists C_j$ $(1 \leq j \leq n)$ s.t. $C_j^{\Gamma_j} \in \sigma\ \}$. $\square$

The least fixpoint M_P^s of a program is similarly defined by using the mapping Φ_I^s and Ψ_I^s instead of Φ_I and Ψ_I, respectively in the previous section. Clearly, M_P^s is also a model of P and we call such M_P^s the *suspicious well-founded model*.

Example 2.5 Let $P = \{\ p \leftarrow q \wedge not\, r,\ \ p \leftarrow \neg r,\ \ q \leftarrow s,\ \ \neg r \leftarrow,\ \ s \leftarrow\ \}$. Then, $M_P^s =< \{p^{\{\{q,s,not\, r\},\{\neg r\}\}}, q^{\{\{s\}\}}, \neg r^{\{\emptyset\}}, s^{\{\emptyset\}}\}; \{\neg p, \neg q, r, \neg s\} >$. $\square$

Definition 2.7 Let P be a program and M_P^s be its suspicious well-founded model. For a suffixed literal L^Γ in M_P^s, if every set in Γ contains a literal L' or $\neg L'$ such that L' is contradictory in M_P^s, L is called *suspicious*. $\square$

We consider a proven fact to be suspicious if every proof of the fact includes an inconsistent information. In another words, if there is at least one proof of a fact which contains no inconsistent information, we do not consider such a fact to be suspicious. A proven fact which is not suspicious is called *sure*.

Note that we do not consider any fact derived from true and false by default information to be suspicious, since such a don't-care information just presents that both positive and negative facts are failed to prove in a program and does not present any inconsistency by itself.

The following proposition presents that a fact which is derived using a suspicious fact is also suspicious.

Proposition 2.2 Let P be a program and L^{Γ} be a suffixed literal in M_P^s. If each set in Γ contains a suspicious fact, then the truth value of L is also suspicious.

Proof: Suppose that each set γ in Γ contains a suspicious fact A. Then A has its own derivation histories Γ' such that each γ' in Γ' contains a literal which is contradictory in M_P^s. By definition, $\gamma' \subseteq \gamma$ then γ also contains the contradictory literal. □

Now reasoning under the *suspicious well-founded semantics* is defined as follows.

Definition 2.8 Let P be a program and M_P^s be its suspicious well-founded model. Then, for each atom A such that A^{Γ} (resp. $\neg A^{\Gamma}$) is in M_P^s, A is called *true with suspect* (resp. *false with suspect*) if A (resp. $\neg A$) is suspicious and $\neg A$ (resp. A) is not sure in M_P^s.

On the contrary, if A (resp. $\neg A$) is suspicious but $\neg A$ (resp. A) is sure in M_P^s, then A is false (resp. true) in M_P^s without suspect. □

In particular, if A is both true and false with suspect, A is contradictory with suspect.

Example 2.6 (cont. from Example 2.4) Let $P = \{\ a \leftarrow \neg b,\ \ \neg b \leftarrow c \wedge not\, b,\ \ c \leftarrow,\ \ \neg c \leftarrow,\ \ d \leftarrow \}$. Then, M_P^s is $< \{c^{\{\emptyset\}},\ \neg c^{\{\emptyset\}},\ d^{\{\emptyset\}},\ a^{\{\{\neg b, c, not\, b\}\}},\ \neg b^{\{\{c, not\, b\}\}}\}; \{b, \neg a, \neg d\} >$. Thus, d is true, c is contradictory, while a and b are true with suspect and false with suspect, respectively. □

In the above example, if a new fact b is added to P, this fact now holds for sure then b becomes true without suspect.

3 Extension to Disjunctive Programs

The semantics of logic programs is recently extended to disjunctive logic programs which contain incomplete information in a program. The well-founded semantics is also extended to disjunctive logic programs by several authors [Ro89, BLM90, Pr90]. In paraconsistent logic programming, [Su90] has also extended the fixpoint semantics of [BS87] to paraconsistent disjunctive logic programs. In this section, we present the extended well-founded semantics for paraconsistent disjunctive logic programs.

A *disjunctive program* is a (possibly infinite) set of the clauses of the form:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n$$

where $l > 0, m, n \geq 0$, each A_i, B_j and C_k are literals and all the variables are assumed to be universally quantified at the front of the clause. The notion of a ground clause (program) is also defined in the same way as in the previous section. Hereafter, we use the term *normal program* to distinguish a program which contains no disjunctive clause.

As in [Sa89], we consider the meaning of a disjunctive program by a set of its *split programs*.

Definition 3.1 Let P be a disjunctive program and G be a ground clause from P of the form:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n \quad (l \geq 2).$$

Then G is *split* into $2^l - 1$ sets of clauses $G_1, \ldots, G_{2^l-1}$ such that for any non-empty subset S_i of $\{A_1, \ldots, A_l\}$;

$$G_i = \{\ A_j \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n \mid A_j \in S_i\ \}.$$

A *split program* of P is a ground normal program which is obtained from P by replacing each disjunctive clause G with its split clauses G_i. □

Example 3.1 Let $P = \{\ p \vee \neg q \leftarrow not\, r,\ \ s \leftarrow p,\ \ s \leftarrow \neg q\ \}$. Then there are three split programs of P as follows:

$$\begin{aligned} P_1 &= \{\ p \leftarrow not\, r,\ \ s \leftarrow p,\ \ s \leftarrow \neg q\ \}, \\ P_2 &= \{\ \neg q \leftarrow not\, r,\ \ s \leftarrow p,\ \ s \leftarrow \neg q\ \}, \\ P_3 &= \{\ p \leftarrow not\, r,\ \ \neg q \leftarrow not\, r,\ \ s \leftarrow p,\ \ s \leftarrow \neg q\ \}. \quad \square \end{aligned}$$

Intuitively, each split program presents a possible world of the original program in which each disjunction is interpreted in either exclusive or inclusive way. The following proposition holds from the definition.

Proposition 3.1 Let P be a disjunctive program and P_i be its split program. If I is a model of P_i, I is also a model of P. □

The extended well-founded models of a disjunctive program are defined by those of its split programs.

Definition 3.2 Let P be a disjunctive program. Then M_P is called the *extended well-founded model* of P if M_P is the extended well-founded model of some split program of P. □

Clearly, the above definition reduces to the extended well-founded model of a normal program in the absence of disjunctive clauses in a program.

A disjunctive program has multiple extended well-founded models in general and each atom possibly has different truth value in each model. In classical two-valued logic programming, a ground atom is usually assumed to be true (resp. false) if it is true (resp. false) in every minimal model of a program. In our multi-valued setting, we define an interpretation of an atom under the extended well-founded semantics as follows.

Definition 3.3 Let P be a disjunctive program, $M_P^1, \ldots, M_P^n$ be its extended well-founded models and $M_P^i(A)(i = 1, \ldots, n)$ be the truth value of an atom A in M_P^i. Then an atom A in P has a truth value μ under the *extended well-founded semantics* if $M_P^1(A) = \ldots = M_P^n(A) = \mu$. □

Example 3.2 For the program P in Example 3.1, there are three extended well-founded models such that $M_P^1 =< \{p, s\}; \{\neg p, q, \neg q, r, \neg r, \neg s\} >$, $M_P^2 =< \{\neg q, s\}; \{p, \neg p, q, r, \neg r, \neg s\} >$ and $M_P^3 =< \{p, \neg q, s\}; \{\neg p, q, r, \neg r, \neg s\} >$. Then s is true and r is don't-care by default in P under the extended well-founded semantics, while truth values of p and q are not uniquely determined. □

When a program has inconsistent models as well as consistent ones, however, it seems natural to prefer consistent models and consider truth values in such models.

Example 3.3 Let $P = \{\ p \leftarrow,\ \ \neg p \vee q \leftarrow\ \}$. Then the extended well-founded models of P are $M_P^1 =< \{p, \neg p\}; \{q, \neg q\} >$, $M_P^2 =< \{p, q\}; \{\neg p, \neg q\} >$ and $M_P^3 =< \{p, \neg p, q\}; \{\neg q\} >$ where only M_P^2 is consistent. □

In the above example, a rational reasoner seems to prefer the consistent model M_P^2 to M_P^1 and M_P^3, and interprets both p and q to be true. The extended well-founded semantics for such a reasoner is defined bellow.

Definition 3.4 Let P be a disjunctive program such that $M_P^1, \ldots, M_P^n$ $(n \neq 0)$ are its *consistent* extended well-founded models. Then an atom A in P has a truth value μ under the *rational extended well-founded semantics* if $M_P^1(A) = \ldots = M_P^n(A) = \mu$. □

Proposition 3.2 Let P be a disjunctive program such that it has at least one consistent extended well-founded model. If an atom A has a truth value μ under the extended well-founded semantics, then A has also the truth value μ under the rational extended well-founded semantics, but not vice versa. □

The suspicious well-founded semantics presented in Section 2.3 is also extensible to disjunctive programs in a similar way.

4 Related Work

Alternative approaches to paraconsistent logic programming based upon the stable model semantics [GL88] are recently proposed in [PR91, GS92a]. These approaches have improved the result of [GL90] in the sense that stable models are well-defined in inconsistent programs. However, these semantics still inherit the problem of the stable model semantics and there exists a

program which has no stable model and yet it contains a meaningful information. For example, a program $\{ p \leftarrow, \quad q \leftarrow not\, q \}$ has no stable model, while it has an (extended) well-founded model in which p is true. This observation presents that the stable model semantics is not always useful to isolate pathological information in a program.

Pereira and Alferes [PA92] present an extended well-founded semantics for normal logic programs which contain explicit negation as well as default negation. Compared with ours, their semantics is restrictive in the sense that they do not allow inconsistency in a program. Moreover, they assume the *coherency principle* such that *if not L is true, then $\neg L$ is also true*. This principle, however, often causes an unintuitive behavior in a program. For instance, consider again the barbar's paradox $shave(b, x) \leftarrow not\, shave(x, x)$ in which the truth value of $shave(b, b)$ is undefined under their extended well-founded semantics. When we get a new knowledge $\neg shave(b, b)$, we expect the truth value of $shave(b, b)$ to be false in the updated program, since it is initially undefined and is lately turned out to be false. But in the presence of the coherency principle, this is not the case. The newly added $\neg shave(b, b)$ implies $not\, shave(b, b)$, which derives $shave(b, b)$ by the clause. As a result, the program becomes inconsistent. This observation suggests that the coherency principle is not always appropriate (although it is possible to include this property to our formalism, if desired).

Wagner [Wa91] has also introduced a logic for possibly inconsistent logic programs with two kinds of negation. His logic is paraconsistent and not destructive in the presence of an inconsistent information, but it is still restricted and different from our lattice valued logic.

Several studies have also been done from the standpoint of contradiction removal in extended logic programs. Kowalski and Sadri [KS90] have extended the answer set semantics of [GL90] in an inconsistent program by giving higher priorities to negative conclusions in a program. This solution is rather ad-hoc and also easily simulated in our framework by giving higher priorities to negative facts in a program. Another approaches such as [PAA91] and [DR91] consider removing contradiction brought about by default assumptions. For instance, consider a program $\{ p \leftarrow not\, q, \quad \neg p \leftarrow r, \quad r \leftarrow \}$. This program has an inconsistent well-founded model, however, it often seems legal to prefer the fact $\neg p$ to p, since p is derived by the default assumption $not\, q$, while its negative counterpart $\neg p$ is derived by the proven fact r. Then they present program transformations for taking back such a default assumption to generate a consistent well-founded model. In our framework, such a distinction is also achieved as follows. Consider a suspicious well-founded model of the program $< \{p^{\{\{not\, q\}\}}, \neg p^{\{\{r\}\}}, r^{\{\emptyset\}}\}; \{q, \neg q, \neg r\} >$ where a fact p has a default fact in its derivation history while $\neg p$ does not, then we can prefer the fact $\neg p$ as a more reliable one. These approaches [PAA91, DR91] further discuss contradiction removal in the context of belief revision or abductive framework, but from the point of view of paraconsistent logic programming, they provide no solution for an inconsistent pro-

gram such as $\{ p \leftarrow, \neg p \leftarrow, q \leftarrow \}$. Another approaches in this direction are [In91, GS92b] in which the meaning of an inconsistent program is assumed to be a collection of maximally consistent subsets of the program.

5 Concluding Remarks

In this paper, we have presented the extended well-founded semantics for paraconsistent logic programs. Under the extended well-founded semantics, a contradictory information is localized and a meaningful information is still available in an inconsistent program. Moreover, a suspicious fact which is affected by an inconsistent information can be distinguished from others by the skeptical well-founded reasoning. The extended well-founded semantics proposed in this paper is a natural extension of the three-valued well-founded semantics and it is well-defined for every possibly inconsistent extended logic program. Compared with other paraconsistent logics, it can treat both classical and default negation in a uniform way and also simply be extended to disjunctive paraconsistent logic programs.

This paper has centered on a declarative semantics of paraconsistent logic programs, but a proof procedure of the extended well-founded semantics is achieved in a straightforward way as an extension of the SLS-procedure [Pr89]. That is, each fact which is true/false in a program have a successful SLS-derivation in a program, while a default fact in a program has a failed derivation. A fact which is inconsistent in a program has a successful derivation from its positive and negative goals. The proof procedure for the suspicious well-founded semantics is also achieved by checking consistency of each literal appearing in a successful derivation. These procedures are sound and complete with respect to the extended well-founded semantics and also computationally feasible.

Acknowledgments I would like to thank V. S. Subrahmanian and John Grant for useful correspondence on the subject of this paper.

References

[Be75] Belnap, N. D., A Useful Four-Valued Logic, in *Modern Uses of Multiple-Valued Logic*, J. M. Dunn and G. Epstein (eds.), Reidel Publishing, 8-37, 1975.

[BLM90] Baral, C., Lobo, J. and Minker, J., Generalized Disjunctive Well-Founded Semantics for Logic Programs, CS-TR-2436, Univ. of Maryland, 1990.

[BS87] Blair, H. A. and Subrahmanian, V. S., Paraconsistent Logic Programming, *Proc. Conf. on Foundations of Software Technology and Theoretical Computer Science (LNCS 287)*, 340-360, 1987.

[Co74] Costa, N. C. A. da, On the Theory of Inconsistent Formal Systems, *Notre Dame J. of Formal Logic 15*, 497-510, 1974.

[DR91] Dung, P. M. and Ruamviboonsuk, P., Well-Founded Reasoning with Classical Negation, *Proc. 1st Int. Workshop on Logic Programming and Nonmonotonic Reasoning*, 120-132, 1991.

[Fi89] Fitting, M., Negation as Refutation, *Proc. 4th Annual Symp. on Logic in Computer Science*, 63-69, 1989.

[Fi91] Fitting, M., Bilattices and the Semantics of Logic Programming, *J. of Logic Programming 11*, 91-116, 1991.

[Gi86] Ginsberg, M. L., Multivalued Logics, *Proc. of AAAI'86*, 243-247, 1986.

[GL88] Gelfond, M. and Lifschitz, V., The Stable Model Semantics for Logic Programming, *Proc. 5th Int. Conf. on Logic Programming*, 1070-1080, 1988.

[GL90] Gelfond, M. and Lifschitz, V., Logic Programs with Classical Negation, *Proc. 7th Int. Conf. on Logic Programming*, 579-597, 1990.

[GS92a] Grant, J. and Subrahmanian, V. S., Reasoning in Inconsistent Knowledge Bases, draft manuscript, 1992.

[GS92b] Grant, J. and Subrahmanian, V. S., The Optimistic and Cautious Semantics for Inconsistent Knowledge Bases, draft manuscript, 1992.

[In91] Inoue, K., Extended Logic Programs with Default Assumptions, *Proc. 8th Int. Conf. on Logic Programming*, 490-504, 1991.

[KL89] Kifer, M. and Lozinskii, E. L., RI: A Logic for Reasoning with Inconsistency, *Proc. 4th Annual Symp. on Logic in Computer Science*, 253-262, 1989.

[KS90] Kowalski, R. A. and Sadri, F., Logic Programs with Exception, *Proc. 7th Int. Conf. on Logic Programming*, 598-613, 1990.

[PAA91] Pereira, L. M., Alferes, J. J. and Aparicio, N., Contradiction Removal within Well-Founded Semantics, *Proc. 1st Int. Workshop on Logic Programming and Nonmonotonic Reasoning*, 105-119, 1991.

[PA92] Pereira, L. M., Alferes, J. J., Well-Founded Semantics for Logic Programs with Explicit Negation, *Proc. ECAI'92*, 102-106, 1992.

[Pr89] Przymusinski, T. C., Every Logic Program has a Natural Stratification and an Iterated Least Fixed Point Model, *Proc. 8th ACM Symp. on Principle of Database Systems*, 11-21, 1989.

[Pr90] Przymusinski, T. C., Extended Stable Semantics for Normal and Disjunctive Logic Programs, *Proc. 7th Int. Conf. on Logic Programming*, 459-477, 1990.

[PR91] Pimentel, S. G. and Rodi, W. L., Belief Revision and Paraconsistency in a Logic Programming Framework, *Proc. 1st Int. Workshop on Logic Programming and Nonmonotonic Reasoning*, 228-242, 1991.

[Ro89] Ross, K., The Well-Founded Semantics for Disjunctive Logic Programs, *Proc. 1st Int. Conf. on Deductive and Object Oriented Databases*, 352-369, 1989.

[Sa89] Sakama, C., Possible Model Semantics for Disjunctive Databases, *Proc. 1st Int. Conf. on Deductive and Object Oriented Databases*, 337-351, 1989.

[Su90] Subrahmanian, V. S., Paraconsistent Disjunctive Deductive Databases, *Proc. 20th Int. Symp. on Multiple-valued Logic*, 339-345, 1990.

[Su90] Subrahmanian, V. S., Y-Logic: A Framework for Reasoning about Chameleonic Programs with Inconsistent Completions, *Fundamenta Informaticae XIII*, 465-483, 1990.

[VRS88] Van Gelder, A., Ross, K. and Schlipf, J. S., Unfounded Sets and Well-Founded Semantics for General Logic Programs, *Proc. 7th ACM Symp. on Principle of Database Systems*, 221-230, 1988.

[Wa91] Wagner, G., A Database Needs Two kinds of Negation, *Proc. 3rd Symp. on Mathematical Fundamentals of Database and Knowledge Base Systems (LNCS 495)*, 357-371, 1991.