This is to certify that the

dissertation entitled

REASONING WITH CONTRADICTORY
DEDUCTIVE DATABASES

presented by

ANASTASIA ANALYTI

has been accepted towards fulfillment
of the requirements for

PhD degree in COMPUTER SCIENCE

Major professor

Date _____

O-12771

REASONING WITH CONTRADICTORY DEDUCTIVE DATABASES

By

Anastasia Analyti

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1994

ABSTRACT

REASONING WITH CONTRADICTORY DEDUCTIVE DATABASES

By

Anastasia Analyti

Deductive databases have become a dominant field of research in recent years because they provide (i) an expressive environment for data modelling, (ii) a single, declarative language for expressing queries, constraints, views, and programs, and (iii) a clear separation of declarative and procedural concepts. A deductive database consists of two parts: a set of known facts, and a set of rules from which new facts can be derived. Consistency of derived facts is not a realistic assumption in many applications. In the presence of contradiction, classical logic fails to give any semantics to the deductive database. Thus, even a single erroneous datum could destroy all meaningful information. The goal of this research is to derive useful information from a set of contradictory rules. In the investigated framework, both negation by default and classical negation are supported. Rules are equipped with a partial order expressing their relative reliability in case of conflict. In this thesis, we propose the *reliable semantics* and *contradiction-free semantics* for contradictory deductive databases. In the proposed semantics, a rule ordering based on reliability is used to choose between conflicting rules. When no choice is possible, the conflicting rules are considered unreliable and their conclusions are blocked. Conclusions from rules that do not contribute to the contradictions are considered reliable and they are used for the derivation of new information. We give equivalent fixpoint and model theoretic characterizations of the proposed semantics. For the contradiction-free semantics we present an equivalent procedural characterization for computing answers to queries.

Both skeptical and credulous types of reasoning are considered. Some of the advantages of the proposed semantics are: (i) they cover a broader domain of logic programs than those semantics proposed earlier, (ii) they are well-defined for every contradictory program in this broader domain, (iii) they extend several semantics proposed earlier, and (iv) they can be computed in polynomial time with respect to the size of the program P when the Herbrand Base of P is finite.

A more general framework is presented where rules are encapsulated into modules. The prospect of contradiction is even stronger when information is distributed in a set of modules. The code of a module is usually hidden from other modules. Thus, modules export their results while they hide the way these results are computed. A partial order expresses the relative reliability of conclusions drawn by these modules. The semantics for this extended framework is called *modular reliable semantics*. We present a fixpoint and model theoretic characterization of the modular reliable semantics. This framework can be used to model multi-agent systems where the knowledge of several experts is represented in a single system.

An application of the reliable semantics to deductive object-oriented databases is also described. In deductive object-oriented databases, rule prioritization can be used (i) to express the fact that specific rules are more reliable than general ones, (ii) to give priorities to inherited rules that are in conflict as a result of multiple inheritance and (iii) to give priorities to class rules that are in conflict as a result of multiple specializations of the same object.

To my parents

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Sakti Pramanik for all his support in my academic and personal life. He helped me build a strong background in the database area. He taught me how to do research and attack problems. He helped me financially in all possible ways and gave me emotional support when I most needed. I would like to thank Dr. Herman Hughes, Dr. William Punch, III, Dr. Gerald Ludden, and Dr. Jacob Plotkin for serving as my committee members and helping me in identifying some of the important issues investigated in this thesis. I would also like to express my appreciation to the Computer Science Department and College of Engineering for their financial support.

Many thanks go to Vasu Vuppala and Walid Tout for answering my numerous Unix-related questions. My friends Argyrios Gerakis, Michael Tornaritis, Mingxia Man, Divesh Srivastava, and Anastassios Petropoulos have been a powerful source of support. Finally, I would like to thank my parents for their encouragement in achieving my academic goals.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Problem Description

A deductive database is a set of sentences in a knowledge representation language. The knowledge representation language that we consider here is *logic programming*. Logic programs represent information about a problem in terms of rules. The idea that logic can be used as a programming language was introduced by Colmerauer and Kowalski [41]. The *semantics* of a logic program specifies what is true or false and can be defined declaratively or procedurally. The *declarative* characterization of a logic program is the specification of its "meaning" in terms of a fixpoint of an operator (*fixpoint semantics*) or in terms of a particular set of interpretations satisfying certain properties (*model theoretic semantics*). The *procedural* characterization of a logic program is given through a query answering algorithm that receives as input a specific query and returns its truth value.

Classical logic has been successful only in representing very precise reasoning such as that found in mathematics. However, human reasoning is often based on conflicting evidence and on assumptions which are not always valid. When a logic program is contradictory, instead of discarding the whole program, we would like to separate the reliable from the unreliable part of the information. This way, useful conclusions can be derived from the reliable information. The goal of this work is to define semantics for logic programs that may be contradictory. The semantics should extend well-known semantics for non-contradictory logic programs. We consider rules to be defaults. Rule prioritization can be viewed as a tool to specify confidence information about these defaults. Some of the reasons for rule prioritization are:

- *Difference in the reliability of sources.* It is possible that a number of sources provide information about a particular topic. If the sources contradict, we wish to use ordering to resolve conflicts.
- *The dominance of specific over general information.* Object-oriented programming is an example where this principle is employed.
- *Regulation.* Regulation can indicate the priority of different conflicting directives. For example, university laws require that foreign students pay out-of-state tuition and TAs pay in-state tuition. However, if a student is both foreign and TA, the directives for TAs are given higher priority than the directives for foreign students.

The prospect of contradiction is even stronger when information is distributed in a set of modules. The code of a module is usually hidden from other modules. Thus, modules export their results while they hide the way these results are computed. When exported results are in conflict, prioritization of results can express higher confidence in some results over others.

1.2 Overview of Semantics for Logic programs

A *normal logic program P* consists of a finite set of clauses of the form: $A \leftarrow L_1,...,L_n$, where A is an atom and $\forall i \leq n$, L_i is a literal, i.e., an atom or its negation. The atom on the left hand side of a rule is called the *head* of the rule and is denoted by $Head_r$. The expression on the right hand side of the rule is called the *body* of the rule and is denoted by $Body_r$. The *Herbrand Universe* of P is the set of all ground (variable-free) terms which can be formed by using the constant and function symbols appearing in P. The *Herbrand Base* of P (HB_P) is the set of all ground atoms which can be formed by using the predicates appearing in P with the terms in the Herbrand Universe of P.

A *2-valued model* of a program P evaluates each literal in HB_P as *true* (value 1) or *false* (value 0). A 2-valued model M is represented as a subset of HB_P where atoms in M are true and atoms not in M are false. In contrast, a *3-valued model* of P evaluates each literal in HB_P as *true* (value 1), *false* (value 0) or *unknown* (value 1/2) and is represented as a subset of $HB_P \cup \sim HB_P$. For any 3-valued model M, there is no atom A s.t. $A \in M$ and $\sim A \in M$. If $A \in M$ then A is true, if

$\sim A \in M$ then A is false, and if neither $A \in M$ nor $\sim A \in M$ then A is unknown. Let M be a 2-valued or 3-valued model of P. The truth value of a set of literals S w.r.t. M is defined as the least truth value of the literals in S. Let $M(S)$ denote the truth value of S w.r.t. M, where S is a literal or a set of literals. Then, $M(Head_r) \geq M(Body_r)$, for every rule r in P.

Initial work on logic programming was only concerned with logic programs with no negation (*Horn logic programs*). Inspired by *SL*-resolution [40] (linear resolution with selection function), Hill [33] developed a linear resolution procedure, called *SLD-resolution* (*SL*-resolution for definite programs). A *goal* in an *SLD*-resolution is of the form $\leftarrow A_1,...,A_n$, where A_i are atoms. An *SLD*-refutation of goal $G_1 = \leftarrow A_1,...,A_n$ is a sequence of goals $G_1,...,G_{n'}$ s.t. $G_{n'} = \leftarrow$ and $\forall i<n'$, G_{i+1} is obtained from $G_i = \leftarrow B_1,...,B_k$ as follows:

(i) B_m is an atom in G_i, called the selected atom,

(ii) $\exists$ rule $C \leftarrow C_1,...,C_l$ in P and substitution θ_i s.t. $B_m\theta_i = C\theta_i$,

(iii) $G_{i+1} = (B_1,...,B_{m-1},C_1,...,C_l,B_{m+1}..,B_k)\theta$.

The sequence of substitutions $\theta_1,...,\theta_{n'-1}$ is an *answer* of goal G_1. The *SLD*-resolution is sound and complete for answering positive queries in Horn logic programs and less expensive than *SL*-resolution.

van Emden and Kowalski [75] presented an equivalent fixpoint characterization of the positive consequences of a Horn logic program. Specifically, let P be a Horn logic program and I an interpretation of P. Then, the van Emden and Kowalski operator $\Psi_P(I)$ is defined as follows: $\Psi_P(I) = \{A \mid \exists$ rule $A \leftarrow A_1,...,A_n$ in P s.t. A_i is true w.r.t. $I, \forall i \leq n\}$.

Let $\Psi_P \uparrow 0(I) = I$, $\Psi_P \uparrow n+1(I) = \Psi(\Psi \uparrow n(I)) \cup \Psi \uparrow n(I)$, and $\Psi_P \uparrow \omega(I) = \cup\{\Psi_P \uparrow n(I) \mid n<\omega\}$, where ω is the first infinite ordinal. The positive consequences of P are defined as the least$_v$[1] fixpoint of Ψ_P which coincides with $\Psi_P \uparrow \omega(\emptyset)$. Thus, the positive consequences of P can be obtained by iterating ω times the Ψ_P operator.

---

[1] We say that a model M of P is the *least*$_v$ model of P iff $M(L) \leq M'(L)$ for any model M' and classical literal L of P.

If logic programming is to be useful in practical situations, it must be able to represent negative information. There are two ways in which one can represent negative information:

(i) By allowing the explicit representation of negative information (*explicit negation*).

(ii) By introducing a metarule which infers negative information (*negation by default*).

According to the first approach, negative information is represented explicitly by allowing classical negation in the heads and bodies of the rules. A disadvantage of this approach is that it is not feasible to include in the logic program every piece of negative information in the problem domain. For example, the amount of negative information that applies to any individual is limitless. For this reason, the idea of introducing a metarule which infers negative information, not explicitly specified in the logic program, was conceived.

Reiter [39] introduced the *closed world assumption* (*CWA*) metarule. According to the *CWA*, $\sim A$ is inferred from a Horn logic program P if A is not classically provable from P, i.e., A is not true in all 2-valued models of P. However, when negations are present in the bodies of the rules (*normal programs*), the *CWA* generally leads to an inconsistent theory. For example, if $P=\{p\leftarrow\sim q.\}$ then P has two 2-valued models $\{p\}$ and $\{q\}$. Consequently, $CWA(P)=\{\sim p, \sim q\}$ and $P\cup CWA(P)$ is inconsistent. Several semantics for normal programs have been proposed. A weaker form of the *CWA*, called *negation as finite failure* (*NAF*), was developed by Clark [16]. According to *NAF*, a ground literal $\sim A$ is inferred when the proof of A using *SLD*-resolution fails finitely.

Clark, augmented *SLD*-resolution with the *NAF* rule. The augmented resolution was named *SLDNF*-resolution (linear resolution with selection function for definite clauses using negation-as-failure) by Lloyd [45]. For example, in the program $P=\{p\leftarrow\sim q.\}$, the *SLDNF*-resolution will infer that q is false and p is true. Clark defined the model theoretic semantics of a normal program using the *completion* of the program and proved the soundness of the *SLDNF*-resolution w.r.t. this semantics. The idea behind the completion of a program P is that the disjunction of the bodies of the rules with head an atom A can be viewed as necessary condition for A to be true. In other words, if $\{A\leftarrow Body_{r_1},\ldots,A\leftarrow Body_{r_n}\}$ is the set of rules in P with head A then the statement $A\leftrightarrow$

$Body_{r_1} \vee ... \vee Body_{r_n}$ is assumed to hold. This implies that negative information about A (A is false) can be inferred if none of the bodies of the rules with head A is true.

Unfortunately, *Clark's semantics* has some serious drawbacks [61]. One of them is that the Clark's completion of a program P is often inconsistent, i.e., the completion of P may not have any 2-valued models. In this case, the semantics of P is considered undefined. For example, the completion of $P=\{a \leftarrow. \quad p \leftarrow \sim p.\}$ is $\{a \leftrightarrow true. \quad p \leftrightarrow \sim p.\}$ which is inconsistent. Thus, P is not given any semantics even though a should intuitively be true. Moreover, if the meaningless rule $p \leftarrow p$ is added to P then the completion of the new program is $\{a \leftrightarrow true. \quad p \leftrightarrow \sim p \vee p.\}$. The semantics of $P \cup \{p \leftarrow p\}$ is $\{a, p\}$ even though the semantics of P should not have been changed with the addition of rule $p \leftarrow p$.

Apt, Blair and Walker [7] split the program rules of a normal program P into layers $S_1,...,S_n$ so that the negative predicates in the body of a rule in layer S_i are defined in layers below S_i. Then, they define: $I_{S_1}=\Psi_{S_1}{\uparrow}\omega(\emptyset)$, $I_{S_2}=\Psi_{S_2}{\uparrow}\omega(I_{S_1})$, ... $I_{S_n}=\Psi_{S_n}{\uparrow}\omega(I_{S_{n-1}})$.
The literal set I_{S_n} is proved to be a minimal 2-valued model of P. The meaning of the program P is defined as that represented by I_{S_n}. Programs that can be split the above way, i.e., programs free from recursion through negative predicates, are called *stratified*. Though this semantics is widely accepted, not all logic programs with an intuitive meaning are stratified. Przymusinski [57] defined the *perfect model semantics* and extended the previous semantics to a larger subclass of programs, called *locally stratified programs*. A program P is locally stratified if it is possible to split HB_P into disjoint sets $S_1,...,S_\alpha,...$, so that for every rule $A \leftarrow A_1,...,A_n,\sim B_1,...,\sim B_m$ in P, where A_i, B_i are atoms, the following are true:

(i) $\forall i \leq n$, stratum(A_i) $\leq$ stratum(A),

(ii) $\forall j \leq m$, stratum(B_j) $<$ stratum(A),

where stratum(A)=α iff the atom A belongs to S_α.

However, there are programs that are not locally stratified but have an intuitive meaning. For example, the program $P=\{p \leftarrow \sim q,a. \quad q \leftarrow \sim p.\}$ is not locally stratified but the *SLDNF*-resolution will infer that the atoms a, p are false and atom q is true.

van Gelder, Ross and Schlipf [76] define the *well-founded semantics* (*WFS*) which characterizes all normal programs. The *WFS* is the least[2] fixpoint of a monotonic operator $\mathbf{W}_P$. The $\mathbf{W}_P(J)$ operator is defined as follows:

- $\mathbf{T}(J)$= $\{L \mid \exists$ rule r: $L \leftarrow L_1,...,L_n$ in P s.t. $L_i \in T \cup J, \forall i \leq n\}$.
- $\mathbf{F}(J)$ is the greatest set S of classical literals s.t. $\forall L \in S$, if r is a rule in P with $Head_r = L$ then $\exists L' \in Body_r$ s.t. $L' \in S$ or $\sim L' \in J$.
- $\mathbf{W}_P(J) = \mathbf{T}(J) \cup \sim \mathbf{F}(J)$.

The transfinite sequence $\{I_a\}$ is defined as follows: $I_0 = \{\}$, $I_{a+1} = \mathbf{W}_P(I_a)$ and $I_a = \cup\{I_b \mid b < a\}$ if a is a limit ordinal. Let d be the least ordinal s.t. $I_{d+1} = I_d$. The *WFS* of P is the meaning represented by I_d. Since $\mathbf{W}_P$ is monotonic, the least fixpoint of $\mathbf{W}_P$ exists but it may not be reached at ω, where ω is the first infinite ordinal. The *WFS* is a 3-valued model of P, denoted by WFM_P. The *WFS* extends the perfect model semantics [57] for locally stratified programs and gives more intuitive results than the *Clark semantics* [16]. The *WFS* is accepted by a large number of researchers but some argue that it is very "weak" because it does not assign the truth value true or false to all the atoms that should intuitively have one. For example, the *WFS* of $P = \{p \leftarrow \sim q.\ q \leftarrow \sim p.\ a \leftarrow p.\ a \leftarrow q.\}$ is $\{\}$ though a should be true either p is true or q is true.

Gelfond and Lifschitz [26] give the definition of a stable model of a normal program P. They define the *P/I transformation*, where I is a 2-valued interpretation, as follows:

(i) Remove from P all rules that contain in their body a default literal $\sim L$ s.t. $I(L)=1$.

(ii) Remove from the body of the remaining rules of P any default literal $\sim L$ s.t. $I(L)=0$.

A *stable model* is a 2-valued interpretation M that satisfies $least_v(P/M) = M$ where $least_v(P/M)$ is the $least_v$ model of the positive program P/M. It can be shown that a 2-valued interpretation $M = \{L_1,...,L_n\}$ is a stable model o f P iff for each L_i there is at least one rule in P with head L_i whose all body literals are true w.r.t. M and all positive body literals are in $\{L_1,...,L_{i-1}\}$. A disadvantage of the stable model semantics is that a normal program may not have any 2-valued

---

[2] A set I is the *least* element of a set I iff $I \in$ I and $I \subseteq J$, for all $J \in$ I.

stable models. Przymusinski [60] generalizes the definition of a 2-valued stable model to that of a 3-*valued stable model*. He shows that every normal program P has a 3-valued stable model and that the *WFS* of P is the intersection of all of its 3-valued stable models.

All of the above semantics take into account the positions of the atoms in the rules. For example, let $P=\{p \leftarrow \sim q.\}$. Then, P is classically equivalent to $\{p \vee q.\}$ and has two minimal classical models $\{p\}$ and $\{q\}$. Yet, all of the above semantics for normal programs agree that in the intended semantics for P, q is false because there is no information indicating that q is true. Since q is false, p is derived from the unique rule of P. Przymusinski [57] argues that the syntax of the rules determines the relative priorities among atoms for truth value minimization. Specifically, let A be an atom and r a rule in P then

- If $\sim A$ is in $Body_r$ then A has *higher* priority than $Head_r$.
- If A is in $Body_r$ then A has *no less* priority than $Head_r$.

Thus, in $P=\{p \leftarrow \sim q.\}$, the truth value of q is minimized first and q is evaluated as false. Then, p is evaluated as true because of the rule $p \leftarrow \sim q$.

Normal programs support only negation by default. Gelfond and Lifschitz [26] introduce the *extended programs* which contain *explicit negation* in addition to negation by default. Thus, extended programs provide negative information both implicitly (negation by default $\sim$) and explicitly (classical negation $\neg$). Classical negation is needed: (i) in case of incomplete information, since it may not be justified for a particular information to be considered false because of absence of further information (closed world reasoning), (ii) when negative information should be inferred if some conditions are satisfied, for example, $\neg light\_off \leftarrow light\_on$, and (iii) to represent default reasoning and exceptions, for example, some of the exceptions of the general rule $fly(X) \leftarrow bird(X)$ are: $\neg fly(X) \leftarrow ostrich(X)$ and $\neg fly(X) \leftarrow penguin(X)$.

Several semantics for extended programs have been proposed in the literature [60, 27, 20, 52, 55, 54, 79, 21, 77]. Yet, these semantics are not defined for all extended programs. In [60], the *well-founded model* (WFM_P) [76] of an extended program P is computed as that of a normal program after replacing every literal $\neg L$ of P with a new atom $\neg\_L$. Yet, the well-founded model of

an extended program can be *contradictory*. For example, the well-founded model of $P=\{\neg p\leftarrow\leftarrow a.\ \ p\leftarrow.\ \ b\leftarrow.\}$ is $\{\sim a, \neg p, p, b\}$ and because of the contradiction, P is not given any semantics in [60]. However, intuitively, the rule $b\leftarrow$ is not "suspect" for the violation of the constraint $\bot\leftarrow p,\neg p$ and thus b should be true. Moreover, it is possible that the *WFS* of an extended program is not *coherent*. The *coherence property* supports the intuition that if a literal is explicitly false then it should also be false by default. For example, the *WFS* of $P=\{\neg p\leftarrow.\ \ p\leftarrow\neg p.\ \ a\leftarrow\neg p.\}$ is $\{\neg p\}$ indicating that $\neg p$ is true but the truth values of p and a are unknown. However, intuitively, if $\neg p$ is true, p should be false and a should be evaluated as true. The *extended well-founded semantics* [54] extends the *WFS* for normal programs and achieves coherence by enforcing the inference rules $\{\sim L\leftarrow\neg L \mid L\in HB_P\}$ (*coherence rules*).

Gelfond and Lifschitz [27] define the *answer set semantics* for extended programs by extending the *P/I* transformation [26] to extended programs P. Let P be a positive program. Then, $a(P)=_{def} HB_P$ if $least_v(P)$ contains a pair of complementary literals. Otherwise, $a(P)=_{def} least_v(P)$. An *answer set* of an extended program P is a 2-valued interpretation M that satisfies $a(P/M)=M$ (classically negative literals in P/M are considered as new atoms). The *answer-set semantics* of an extended program is defined as the intersection of its answer-sets. If HB_P is an answer set of P then P is called *contradictory*. The answer set semantics of a contradictory program is meaningless because it implies every literal in HB_P. For example, the answer set semantics of $P=\{\neg p\leftarrow.\ \ p\leftarrow.\ \ a\leftarrow.\}$ is HB_P.

The *contradiction removal semantics* (*CRS*), defined in [52, 55], extends the *WFS* for normal programs and avoids contradictions brought about by *CWAs*. For example, the *CRS* of $P=\{\neg p\leftarrow\leftarrow a.\ \ p\leftarrow.\ \ b\leftarrow.\}$ is $\{p, b\}$ which is non-contradictory. Yet, the problem of contradictions is not totally solved since no semantics is given to $P'=\{\neg p\leftarrow.\ \ p\leftarrow.\ \ b\leftarrow.\}$ even though b should be true. The same arguments hold for the *argumentation semantics* defined in [21].

1.3 Our Approaches for Avoiding Contradictions

In this section, we give the underlying intuitions of our approaches for handling contradictions in logic programs. Consider the program

$P=\{r_1: a\leftarrow.\quad r_2: b\leftarrow.\quad r_3: p\leftarrow a.\quad r_4: \neg p\leftarrow.\quad r_5: c\leftarrow p.\quad r_6: d\leftarrow \neg p.\}$.

All literals a, b, p, $\neg p$, c, d are evaluated as true in WFM_P. Thus, according to the *WFS*, both of the complementary literals p and $\neg p$ are true. Moreover, the literals p and $\neg p$ are used to infer literals c and d using rules r_5 and r_6, respectively. However, because of the contradiction and the fact that there is no information about the relative reliability of the rules, the truth values of p and $\neg p$ should be unknown and all derivations using literals p and $\neg p$ should be blocked. Thus, all literals p, $\neg p$, c, d should be evaluated as unknown. Generally speaking, in the desired semantics of an extended program, no literal should be evaluated as true if its derivation is based on contradictory information. Note that according to the *WFS* of P, literal a is true. Though the derivation of a is not based on contradictory information, the literal a contributes to the derivation of the pair of complementary literals $\{p, \neg p\}$. Thus, it is possible that the information of rule r_1 is faulty. In other words, rule r_1 should be considered unreliable and literal a should be evaluated as unknown. It is our belief that in case of contradiction, we should not only restrict inferences from contradictory literals but also from literals (*possible sources of the contradiction*) contributing to the derivation of contradictory literals. Literal b should be evaluated as true because it is neither a source of a contradiction nor its derivation is based on contradictory information. In other words, rule r_2 is reliable.

Since there is no information about the relative reliability of the rules, all rules r_1, r_3, and r_4 are unreliable. In our proposed semantics, if rule r_4 has higher priority than rule r_1 or r_3 then r_4 is considered reliable and $\neg p$ is evaluated as true. This is because the *WFS* of the rules in P with priority no lower than r_4 is non-contradictory. If rule r_1 has higher priority than rule r_3 or r_4 then r_1 is considered reliable and a is evaluated as true. Otherwise, r_1 is considered unreliable.

Let P be an extended program and S a set of literals. A *dependency path* T of a literal L w.r.t. S is a sequence of goals $G_1,...,G_n$ defined as follows: (i) $G_1=\leftarrow L$, (ii) $\forall i\leq n$, all literals in G_i are in

S, and (iii) $\forall 1<i\leq n$, G_i results from G_{i-1} by replacing a literal L' in G_{i-1} with the literals in the body of a rule r in $P \cup \{\sim L \leftarrow \neg L \mid L \in HB_P\}$ which has head L' (this is called *expansion* of L' using rule r). We say that a literal L' *contributes to the derivation* of a pair of complementary literals $\{L, \neg L\}$ in S if there is a dependency path T of L or $\neg L$ w.r.t. S s.t. L' is a member of a goal in T. For example, let $P=\{r_1: p \leftarrow \sim a,\ b.\quad r_2: \neg p \leftarrow.\quad r_3: b \leftarrow.\}$ then a dependency path of p w.r.t. $WFM_P = \{\sim a, p, \neg p, b, \sim\neg a, \sim\neg b\}$ is: $G_1 = p \leftarrow$, $G_2 = \leftarrow \sim a, b$. Since $\{p, \neg p\} \subseteq WFM_P$ and b is in G_2, we say that b contributes to the derivation of $\{p, \neg p\}$ in WFM_P.

Let P be a program with contradictory well-founded model. We will take two approaches in order to avoid the contradiction. According to the first approach, a CWA $\sim L \in WFM_P$ is evaluated as true only if $\sim L$ is *reliable*, i.e., it does not contribute to the derivation of any pair of complementary literals in WFM_P. When all literals in the body of a rule r are true, $Head_r$ is evaluated as true only if r is *reliable*, i.e., $Head_r$ does not contribute to the derivation of any pair of complementary literals in ${\Psi_{P_r}}^{\uparrow\omega}(\emptyset)$, where P_r is the set of rules in P with priority no lower than r. Let I be a set of literals evaluated as true. Rules with head in $\{\neg L \mid L \in I\}$ are called *blocked* w.r.t. I. Literals that are not reliable in P may be reliable in P_I, where P_I equals P minus the blocked rules w.r.t. I. Thus, an iterative process can be defined starting from the empty interpretation until a fixpoint is reached. Specifically, we define the monotonic operator $W_P(I)$ which intuitively contains all literals in WFM_{P_I} which are reliable. The meaning of P is the least fixpoint of $W_P(I)$.

For example, consider the program $P = \{ r_1: \neg a \leftarrow.\quad r_2: a \leftarrow \neg p.\quad r_3: \neg p \leftarrow.\quad r_4: p \leftarrow.\}$ and assume that r_3 has lower priority than r_4. Then, $WFM_P = \{\neg a, \neg p, p, a\}$ which is contradictory. Rule r_1 is unreliable in $P_\emptyset$ because ${\Psi_{P_{r1}}}^{\uparrow\omega}(\emptyset) = \{p, \neg p, a, \neg a\}$ and $Head_{r_1} = \neg a$ contributes to the derivation of $\{a, \neg a\}$. Similarly, rule r_3 is unreliable in $P_\emptyset$. In contrast, rule r_4 is reliable in $P_\emptyset$ because ${\Psi_{P_{r4}}}^{\uparrow\omega}(\emptyset) = \{p\}$ which is consistent. Thus, $W_P(\emptyset) = \{p\}$. Let $I = W_P(\emptyset)$. Since rule r_3 is blocked w.r.t. I, $P_I = \{r_1: \neg a \leftarrow.\quad r_2: a \leftarrow \neg p.\quad r_4: p \leftarrow.\}$. Since $WFM_{P_I} = \{\neg a, p, \sim a, \sim\neg p\}$ which is consistent, rule r_1 is reliable in P_I. Thus, ${W_P}^{\uparrow 3}(\emptyset) = {W_P}^{\uparrow 2}(\emptyset) = \{\neg a, p, \sim a, \sim\neg p\}$ and the semantics of P is $\{\neg a, p, \sim a, \sim\neg p\}$.

When a literal L in a goal of a dependency path T is expanded using a rule r, not all literals in $Body_r$ should always be included in the expansion. For example, let r be a rule in P s.t. $Head_r$ is true independently of the truth value of a literal $L \in Body_r$. Then, L should not be included in the expansion of $Head_r$ using rule r. For example, assume P contains the rule r: $c \leftarrow a, \neg b$ where a and b are the inputs and c is the output of an *OR* gate. Note that if a is true then the head c of r is true independently of the truth value of literal $\neg b$. Thus, the literal $\neg b$ should not be included in the expansion of c using rule r. The same is true in the case of general rules and exceptions. For example, consider the rules r_1: $burns(X) \leftarrow match(X)$, r_2: $match(a) \leftarrow$, and r_3: $\neg burns(a) \leftarrow$. Though we have the information that a does not burn, it is possible that we do not want to consider the information $match(a)$ as unreliable because a may be a wet match. Thus, $match(a)$ should not be included in the expansion of $burns(a)$ using rule r_1. Because of the above arguments, it is our belief that the user should be able to decide which literals in the bodies of the rules should be included in the expansion of a literal in a dependency path.

A problem of this approach is that if the *WFM* of a program P is consistent then every literal $L \in WFM_P$ is considered reliable even when $\neg L$ cannot be false. For example, consider the program $P = \{r_1: p \leftarrow \sim p. \quad r_2: \neg q \leftarrow p. \quad r_3: q.\}$. Since $WFM_P = \{q, \sim\neg p\}$, the semantics of P according to the first approach is $\{q, \sim\neg p\}$. Thus, q is evaluated as true even though for p either true or false, $\neg q$ is derived from rules r_1 and r_2. Since r_3 does not have higher priority than the other rules, it is arguable that q should not be considered reliable and the truth value of q should be unknown. This viewpoint is adopted by our second approach for avoiding contradictions.

In our second approach, the derivation of contradictory beliefs is avoided by expanding the original program P with the *contrapositives* of its rules. The *expanded program* is denoted by $exp(P)$. For example, if

$P = \{$*switch_on*$\leftarrow$.

light_on$\leftarrow$*switch_on*, ~*broken*.

light_off$\leftarrow$.

$\neg$*light_off*$\leftarrow$*light_on*.

$\neg light\_on \leftarrow light\_off.$

}

then $WFM_P = \{switch\_on,\ light\_on,\ light\_off,\ \neg light\_on,\ \neg light\_off,\ \sim\neg switch\_on,\ \sim broken,\ \sim\neg broken\}$ which is contradictory. However, $WFM_{exp(P)} = \{switch\_on,\ broken,\ light\_off,\ \neg light\_on,\ \sim light\_on,\ \sim\neg light\_off,\ \sim\neg switch\_on,\ \sim\neg broken\}$,

where $exp(P) = P \cup \{\ broken \leftarrow switch\_on, \neg light\_on.\quad \neg switch\_on \leftarrow \neg light\_on, \sim broken.\}$.

$WFM_{exp(P)}$ is consistent and gives the intuitive results.

Not all of the contradictions can be resolved this way. For example, consider

$P = \{switch\_on \leftarrow.$

$light\_off \leftarrow.$

$light\_on \leftarrow.$

$\neg light\_off \leftarrow light\_on.$

$\neg light\_on \leftarrow light\_off.$

}

then $exp(P)=P$ and $WFM_{exp(P)} = \{switch\_on,\ light\_on,\ light\_off,\ \neg light\_on,\ \neg light\_off,\ \sim\neg switch\_on\}$ which is contradictory. To avoid the contradiction, we take the following approach. When all literals in the body of a rule r are true, the head L of r is evaluated as true only if $\neg L$ is *c-unfounded* w.r.t. r, i.e., it cannot be derived from the set of rules in exp(P) with priority no lower than r and the coherence rules. Let I be a set of literals evaluated as true. Then, rules with head in $\{\neg L |\ L \in I\}$ are considered *blocked* w.r.t. I. Literals that were not c-unfounded in P may be c-unfounded in P_I, where P_I equals P minus the blocked rules w.r.t. I. Thus, an iterative process can be defined starting from the empty interpretation until a fixpoint is reached, similarly to the first approach.

Not all contrapositives of a rule should always be considered. Let r be a rule in P s.t. $Head_r$ is true independently of the truth value of a literal L in the body of r. Then, the contrapositive of r with head $\neg L$ should not be added to $exp(P)$. For example, assume P contains the rule r: $c \leftarrow a, \neg b$ where a and b are the inputs and c is the output of an *OR* gate. Note that if input a is true then

output c is true independently of the truth value of input $\neg b$. Because of this, the contrapositive $b \leftarrow a, \neg c$ of r is meaningless. Consider the general rule $burns(X) \leftarrow match(X)$. If we know that something does not burn we might not want to derive (from the contrapositive of the rule) that it is not a match because it may be a wet match. Because of the above arguments it is our belief that the user should decide if the contrapositive of a rule should be considered or not.

A problem of the second approach for avoiding contradictions is that by taking the contrapositives of the rules, some of the priorities of the atoms based on their positions in the rules are lost. For example, consider

$P = \{r_1: fly(X) \leftarrow bird(X), \sim ab_1(X).$

$r_2: \neg fly(X) \leftarrow penguin(X), \sim ab_2(X).$

$r_3: ab_1(X) \leftarrow penguin(X), \sim ab_2(X).$

$r_4: penguin(a) \leftarrow.$

$\}$

then $WFM_P = \{penguin(a), \sim ab_2(a), ab_1(a), \neg fly(a), \sim\neg penguin(a), \sim\neg ab_2(a), \sim\neg ab_1(a), \sim fly(a)\}$ which is non contradictory. The semantics of P according to the first approach coincides with WFM_P which is the intuitively correct semantics. However, $WFM_{exp(P)} = \{penguin(a), \sim\neg penguin(a)\}$ and thus, meaningful information in P is lost in $WFM_{exp(P)}$. This is because in P, the atom $ab_2(a)$ is given higher priority for truth value minimization than the atom $ab_1(a)$. Consequently, in WFM_P, $ab_2(a)$ is evaluated as false and $ab_1(a)$ is evaluated as true. However, by taking the contrapositive of rule r_3 in $exp(P)$, this syntactically determined priority is lost.

In both of the above approaches, some of the conflicts can be resolved by considering the partial ordering of rules (*rule prioritization*). For example, if the rule $\neg fly(X) \leftarrow penguin(X)$ is given higher priority than the more general rule $fly(X) \leftarrow bird(X)$ then we will correctly derive that penguin a does not fly. However, not all of the conflicts can be resolved using the priorities of the rules.

1.4 Dissertation Outline

The Chapters in this thesis are outlined as follows:

Chapter 2 develops the *reliable semantics* (*RS*) for extended programs with rule prioritization and integrity constraints (*EPP*). *RS* avoids contradictions by taking the first approach presented in section 1.3. The concepts of reliable rule and reliable default literal w.r.t. an interpretation are developed. These are used in the declarative definition of *RS*. We give both a fixpoint and model theoretic characterization of the reliable semantics and prove that they are equivalent. The model theoretic characterization of the *RS* of an *EPP*, *P* is given by defining the *stable r-models* of *P*. We prove that *RS* is the least stable *r*-model of *P*. *RS* is shown to be a generalization of (i) the well-founded semantics for normal programs [76], (ii) extended well-founded semantics for non-contradictory extended programs [54], and (iii) ordered logic for ordered logic programs [24, 43]. Other related work is also discussed. Appendix B contains a prolog program which works as an *RS* inference engine for propositional *EPPs*.

In Chapter 3, the reliable semantics is extended to a class of programs, called *prioritized modular logic programs* (*PMPs*). The semantics of the extended class is called *modular reliable semantics* (*MRS*). A *PMP* consists of a set of modules and a partial order $<_{def}$ on the predicate definitions (M,p), where M is a module and p is a predicate exported by M. When a conflict occurs, $<_{def}$ expresses our relative confidence in the predicate definitions contributing to the conflict. The concept of reliable indexed literal w.r.t. an interpretation is developed. This is used in the declarative definition of *MRS*. We present both a fixpoint and model theoretic characterization of the reliable semantics and prove that they are equivalent. The model theoretic characterization of the *MRS* of *P* is given by defining the *stable m-models* of *P*. *MRS* is shown to be the least stable *m*-model of *P*. We show that under certain conditions a *PMP* can be translated to an equivalent extended program with rule prioritization. The use of *MRS* for modelling multi-agent systems is

presented. Related work on combining multiple deductive databases and maintaining consistency in a distributed environment is discussed.

Chapter 4 develops the *contradiction-free semantics* (*CFS*) for extended programs with rule prioritization. *CFS* avoids contradictions by taking the second approach presented in section 1.3. The concept of *c*-unfounded literal w.r.t. a rule and an interpretation is developed. This is used in the declarative definition of *CFS*. We give both a fixpoint and model theoretic characterization of *CFS* and prove that they are equivalent. The model theoretic characterization of the *CFS* of a program *P* is given by defining the *stable c-models* of *P*. *CFS* is shown to be the least stable *c*-model of *P*. We show that *CFS* generalizes (i) the well-founded semantics for normal programs [76], (ii) ordered logic for ordered logic programs [24, 43], and (iii) strong belief revision semantics for extended programs [79]. Other related work is also discussed. The *SLCF*-resolution (linear resolution with selection function for contradiction-free semantics) for computing answers for extended programs with rule prioritization is presented. The *SLCF*-resolution is shown to be sound and complete w.r.t. *CFS*. Appendix A contains the proofs of section 4.3. Appendix B contains a proof procedure and the corresponding prolog program which works as an *CFS* inference engine for propositional programs.

Chapter 5 describes an application of *RS* to deductive object-oriented databases. An object-oriented logic programming language, called *reliable object logic* (*ROL*) is presented. In *ROL*, data and behavior are encapsulated into classes which are structured in a generalization lattice. Object-registration rules are used to register an object to a class or to exclude it from a class. Method rules define the behavior of the objects of a class. Method rules are inherited from the superclasses to subclasses. Every *ROL* program *P* can be translated to an equivalent *EPP*, *P'*. The *RS* of *P* is defined as the *RS* of *P'*.

Parts of the thesis have been published as follows. The content of Chapter 2 appears in [3, 6]. The content of Chapter 3 appears in [5]. The content of Chapter 4 appears in [4].

CHAPTER 2

RELIABLE SEMANTICS FOR EXTENDED LOGIC PROGRAMS WITH RULE PRIORITIZATION

2.1 Introduction

Several semantics for extended programs have been proposed in the literature [60, 27, 20, 52, 55, 54, 79, 21, 77]. Yet, these semantics are not defined for all extended programs. In [60], the *well-founded model* [76] of an extended program P is computed as that of a normal program after replacing every literal $\neg L$ of P with a new atom $\neg\_L$. However, the well-founded model of an extended program can be *contradictory*. For example, the well-founded model of P={$\neg p \leftarrow \sim a.$ $p \leftarrow.$ $b \leftarrow.$} is {$\sim a$, $\neg p$, p, b} and because of the contradiction, P is not given any semantics in [60]. Yet, intuitively, the rule $b \leftarrow$ is not "suspect" for the violation of the constraint $\bot \leftarrow p, \neg p$ and thus b should be true.

The *contradiction removal semantics* (*CRS*), defined in [52, 55], extends the *well-founded semantics* [76] and avoids contradictions brought about by *CWAs*. For example, the *CRS* of P={$\neg p \leftarrow \sim a.$ $p \leftarrow.$ $b \leftarrow.$} is {p, b} which is non-contradictory. Yet, the problem is not totally solved since no semantics is given to P' ={$\neg p \leftarrow.$ $p \leftarrow.$ $b \leftarrow.$} even though b should be true. The same arguments hold for the *argumentation semantics* defined in [21].

Prioritization of defaults is investigated in [42, 24, 25, 11, 12, 43, 64, 65, 66]. Yet, negation by default is not considered in these works. In [42, 24, 25, 43], alternative semantics for *ordered logic* programs are presented. A default in an ordered logic program is a unidirectional rule. In [11, 12], a default is a clause, that is, there is no distinction between the head and the body of a default rule. The work in [65, 66] is the most general from the point of view that defaults are general

formulas and when a default instance cannot be satisfied, partial satisfaction of it, is considered. A conceptualization of both implicit and explicit preferences on data is given in [35].

An *extended program with rule prioritization* (*EPP*) consists of a set of partially ordered rules and a set of constraints. In this chapter, we define the *reliable semantics* (*RS*) for *EPPs*. *RS* extends the well-founded semantics [76] and the extended well-founded semantics [54] to *EPPs*. The *RS* of an *EPP* is always defined and does not violate any constraint. Every *EPP* has at least one *stable r-model*. The *RS* of a program P is the least fixpoint of a monotonic operator and the least stable *r*-model of P. An ordered logic program can be seen as an *EPP* which is free of default literals, $S_r=\{\}$ $\forall$ rule r, and all constraints are of the form: $\bot\leftarrow L,\neg L$. If P is an ordered logic program then the *RS* of P coincides with the *skeptical c-partial model* of P [24] and is a subset of the *well-founded partial model* of P [43]. When the Herbrand base of an *EPP* is finite, the complexity of computing *RS* is polynomial w.r.t. the size of the program.

2.2 *r*-models for Extended Programs with Rule Prioritization

Our alphabet contains a finite set of constant, predicate and variable symbols from which terms and atoms are constructed in the usual way. A *classical literal* is either an atom A or its classical negation $\neg A$. The classical negation of a literal L is denoted by $\neg L$ and $\neg(\neg L)=L$. The symbol ~ stands for negation by default and $\sim(\sim L)=L$. A *default literal* is denoted by $\sim L$, where L is a classical literal.

An *extended program with rule prioritization* (*EPP*) is a tuple $P=<R_P,IC_P,<_R>$. R_P is a finite set of rules r: $L_0\leftarrow L_1,...,L_m,\sim L_{m+1},...,\sim L_n$, where r is a label and L_i are classical literals. Every rule r has a corresponding set $S_r \subseteq Body_r$, called the *preliminary suspect set* of r. IC_P is a finite set of constraints $\bot\leftarrow L_1,...,L_k$, where L_i are classical literals. The precise meaning of S_r will be given in the definitions. Intuitively, when a constraint $\bot\leftarrow L_1,...,L_k$ is violated, the rules used in the *last* step of the derivation of L_i are considered "suspect." If a rule r is "suspect" for a constraint violation then the rules and *CWAs* used in the last step of the derivation of literals in S_r are also "suspect."

The values of S_r depend on the reasons for a constraint violation. There are two basic views on the reasons for the violation of a constraint $\perp \leftarrow L_1,...,L_k$: (v1) rules used in the *last* step of the derivation of L_i are incomplete[1] or (v2) *CWAs* and/or rules used in *some* step of the derivation of L_i are unreliable. According to the first view (v1), the skeptical meaning of the program $P' = \{r_1: a \leftarrow.\ \ r_2: b \leftarrow.\ \ r_3: p \leftarrow a.\ \ r_4: \neg p \leftarrow b.\}$ is $\{a, b\}$. Rules r_1 and r_2 are not used in the *last* step of the derivation of p, $\neg p$ and thus according to (v1), they are reliable. On the contrary, rules r_3 and r_4 are used in the *last* step of the derivation of p, $\neg p$ and thus they are considered incomplete, i.e., r_3 should be $p \leftarrow a, \sim b$ or r_4 should be $\neg p \leftarrow b, \sim a$. Consequently, the literals a, b are evaluated as true whereas both p and $\neg p$ are undefined. View (v1) is implied by ordered logic [24, 43] and vivid logic [77] and becomes explicit in our framework when $S_r = \{\}\ \forall$ rule r. For example, if $S_{r_i} = \{\}$, $\forall i \leq 4$ then rules r_3 and r_4 are "suspect" for the violation of $\perp \leftarrow p, \neg p$. However, rules r_1 and r_2 are not "suspect" because the literals a and b do not belong to S_{r_3} or S_{r_4}. According to the second view (v2), which is more conservative than (v1), the skeptical meaning of P' is $\{\}$. This is because all rules in P' are used in the derivation of p, $\neg p$ and thus all rules in P' are considered unreliable. View (v2) becomes explicit in our framework when $S_r = Body_r$, $\forall$ rule r. For example, if $S_{r_i} = Body_{r_i}$, $\forall i \leq 4$ then not only rules r_3, r_4 but also r_1, r_2 are "suspect" for the constraint violation.

Other views corresponding to $S_r \neq \{\}$ and $S_r \neq Body_r$ for a rule r are also possible. For example, consider the program $P' = \{r_1: a \leftarrow.\ \ r_2: b \leftarrow.\ \ r_3: \neg p \leftarrow.\ \ r_4: p \leftarrow a, b.\}$. If $S_{r_4} = \{a\}$ then rule r_1 is "suspect" for the violation of $\perp \leftarrow p, \neg p$ and rule r_2 is not. Consequently, the skeptical meaning of P' is $\{b\}$. Similarly, if $S_{r_4} = \{b\}$ then the skeptical meaning of P' is $\{a\}$.

The relation $<_R \subseteq R_P \times R_P$ is a strict partial order (irreflexive, asymmetric and transitive), denoting the relative reliability of the rules. Let r and r' be two rules. The notation $r < r'$ means that r is less reliable than r', that is, $r < r'$ iff $(r, r') \in <_R$. The notation $r \not< r'$ means that r is not less reliable that r'. Note that, $r \not< r$ since $<_R$ is irreflexive. Intuitively, a rule r is considered *reliable* if

---

[1] We say that a rule is *incomplete* if not all possible exceptions are enumerated in its body.

it is not "suspect" for any constraint violation caused by rules with priority no lower than r. Thus, deciding if a rule r is reliable depends only on the rules $r' \not< r$.

The set of instantiated classical literals of P is called the *Herbrand Base* (HB_P) of P. The constraints in $BC_P=\{\bot\leftarrow L,\neg L \mid L\in HB_P\}$ are called *basic constraints*. We assume that $BC_P \subseteq IC_P$. An *EPP*, P, is called *extended* iff $IC_P = BC_P$ and $<_R=\{\}$. An extended program P is called *normal* iff rules in P are free of classically negative literals. If S is a set of literals then $\sim S=_{def}\{\sim L \mid L\in S\}$ and $\neg S=_{def}\{\neg L \mid L\in S\}$.

The instantiation of an *EPP*, P, is defined as follows: The instantiations of R_P and IC_P are defined the usual way. Let r_{inst} and r'_{inst} be instances of rules r and r' in P then $r_{inst}<r'_{inst}$ iff $r<r'$. In the rest of the paper, we assume that programs have been instantiated and thus all rules are propositional.

Example 2.2.1 (credit confusion problem): Consider the following *EPP*, $P=<R_P,IC_P,<_R>$:

R_P={ /* If Ann is a foreign student (resp. teaching assistant) then she needs 12 (resp. 6) credits */

r_1: need_credits(ann,12)←foreign_stud(ann).

r_2: need_credits(ann,6)←TA(ann).

r_3: TA(ann).

r_4: foreign_stud(ann). where $S_{r_i}=\{\}, \forall i\leq 4\}$,

IC_P={ *ic*: ⊥←need_credits(ann,6), need_credits(ann,12). } and $r_1<r_2$.

Every classical model of R_P violates the constraint *ic*. Rule r_2 is considered reliable because no constraint violation is caused by rules with priority no lower than r_2, that is, r_2, r_3 and r_4. Rule r_1 is "suspect" for the violation of *ic* caused by r_1, r_2, r_3 and r_4 because $Head_{r_1}\in Body_{ic}$. Consequently, rule r_1 is unreliable. Since $S_{r_1}=\{\}$ and $S_{r_2}=\{\}$, the rules r_3 and r_4 with heads TA(ann) and foreign_stud(ann) are not "suspect" for the violation of *ic*. Consequently, r_3 and r_4 are reliable. The literals TA(ann), foreign_stud(ann) and need_credits(ann,6) should be true in the desired semantics of P because they are derived from the reliable rules r_2, r_3 and r_4.

$S_{r_1}=\{\}$ expresses that rule r_1 is incomplete, i.e., not all possible exceptions are enumerated in the body of r_1. Consequently, though rule r_1 is "suspect", there is no reason to suspect rule r_4

which is used for the derivation foreign_stud(ann)$\in Body_{r_1}$. In contrast, if S_{r_1}= {foreign_stud(ann)} then the rule r_4 is "suspect" for the constraint violation. Consequently, r_4 is unreliable and the truth value of foreign_stud(ann) is undefined. If $r_1<r_4$ or $r_2<r_4$ or $r_3<r_4$ then r_4 is reliable independently of the value of S_{r_1} and thus foreign_stud(ann) is evaluated as true. Similarly, if $S_{r_2}=\{\}$ or $r_1<r_3$ or $r_2<r_3$ or $r_4<r_3$ then r_3 is reliable and TA(ann) is evaluated as true. Otherwise, r_3 is considered unreliable and the truth value of TA(ann) is undefined.

Example 2.2.2 (unloading the gun [31]): Consider the following *EPP*, $P=<R_P,IC_P,<_R>$:

$R_P=\{$ r_1: $loaded(t_1)\leftarrow loaded(t_0)$. ... r_n: $loaded(t_n)\leftarrow loaded(t_{n-1})$.

r_0: $loaded(t_0)$. r_{n+1}: $\neg loaded(t_n)$. where $S_{r_i}=\{\}$, $\forall i\leq n+1\}$,

$IC_P=BC_P$ and $r_i< r_0$ and $r_i< r_{n+1}$, $\forall i\in\{1,...,n\}$ $\}$.

Rules $r_1,...,r_n$ are instances of the default rule "if a gun is loaded at time t_i then it will still be loaded at time t_{i+1}." Rules r_0 and r_{n+1} represent the facts that the gun is loaded at time t_0 and it is found unloaded at time t_n. Note that every classical model of R_P violates the constraint $\bot\leftarrow loaded(t_n),\neg loaded(t_n)$. The rules r_0 and r_{n+1} are reliable because they have higher priority than $r_1,...,r_n$ and they do not generate any constraint violation. Since $S_{r_n}=\{\}$, the rules $r_1,...,r_{n-1}$ are not "suspect" for the constraint violation even though they are used in the derivation of $loaded(t_n)$. So, rules $r_1,...,r_{n-1}$ are reliable and the only unreliable rule is r_n. This implies that the gun remained loaded until t_{n-1}.

If $S_{r_i}=\{loaded(t_{i-1})\}$, $\forall 1\leq i\leq n$, then all rules r_j, $j=1,...,n$, are "suspect" for the constraint violation because they are used in the derivation of $loaded(t_j)$, $j=1,...,n$. Consequently, all rules $r_1,...,r_n$ are unreliable. This implies that the gun was unloaded some time between t_1 and t_n but we do not know exactly when.

Definition 2.2.1 (interpretation): Let P be an *EPP*. A set $I=T\cup\sim F$ is an interpretation of P iff T and F are disjoint subsets of HB_P. An interpretation I is *consistent* iff there is no constraint $\bot\leftarrow L_1,...,L_n$ in P s.t. $L_i\in T$, $\forall i\leq n$. An interpretation I is *coherent* iff it satisfies the *coherence property*: $L\in F$ if $\neg L\in T$.

In interpretation $I=T\cup\sim F$, T contains the *classically true* literals, $\neg T$ contains the *classically false* literals and F contains the literals *false by default*. When I is a consistent interpretation, there is no L such that $L\in T$ and $\neg L\in T$ because this will violate the basic constraint $\bot\leftarrow L,\neg L$. The *coherence property* first appeared in [54] and it expresses the intuition that if a literal is classically false then it is also false by default.

Definition 2.2.2 (truth valuation of a literal): A literal L is true (resp. false) w.r.t. an interpretation I iff $L\in I$ (resp. $\sim L\in I$). A literal that is neither true nor false w.r.t. I, it is undefined w.r.t. I.

An interpretation I can be seen equivalently as a function from the set of ground classical literals to $\{0,1/2,1\}$, where $I(L)=1$ when L is true w.r.t. I, $I(L)=0$ when L is false w.r.t. I and $I(L)=1/2$ when L is undefined w.r.t. I. Both views of an interpretation, as a set and as a function, will be used in the paper. Note that, $I(\sim L)=1-I(L)$, for any literal L. If I is a coherent interpretation then $I(L)=1$ implies $I(\neg L)=0$. We define $I(\emptyset)=_{def} 1$ and $I(S)=_{def} \min\{I(L)|\ L\in S\}$ where S is a non-empty set of literals.

The *coherence operator* (*coh*) transforms an interpretation to a coherent one.

Definition 2.2.3 (coh operator [54]): Let $I=T\cup\sim F$ be an interpretation of an *EPP*. *coh*(I) is the coherent interpretation $T\cup\sim F'$, where $F'=F\cup\{L|\ \neg L\in T\}$.

Let I be a set of literals known to be true. In Definitions 2.2.6 and 2.2.8, the concepts of *reliable default literal* and *reliable rule* w.r.t. I are defined. These concepts are used in the fixpoint computation of the *RS* of an *EPP*. In *RS*, a default literal $\sim L$ is true by *CWA* only if $\sim L$ is a reliable default literal w.r.t. *RS* and a rule r is used for the derivation of $Head_r$ only if r is a reliable rule w.r.t. *RS*.

The next definition expresses that a rule r should be blocked if $\neg Head_r$ is known to be true.

Definition 2.2.4 (blocked rule): Let I be a literal set. A rule r is *blocked* w.r.t. I iff $\neg Head_r\in I$.

To decide if a default literal is reliable w.r.t. I, all possible constraint violations should be considered. For this, the set of literals Pos_I is computed. Intuitively, Pos_I is the possibly inconsistent well-founded model of R_P when rules are blocked as indicated in I and coherence is enforced. Specifically, Pos_I is the least fixpoint of the monotonic operator $\mathbf{PW}_I$ which resembles the W operator of the well-founded semantics [76]. When P is a normal program, $\mathbf{PW}_\emptyset \equiv \mathbf{W}$.

Definition 2.2.5 (possible literal set w.r.t. I): Let P be an *EPP* and I,J be sets of literals. The *possible literal set* w.r.t. I, Pos_I, is defined as follows:

- $\text{PT}_{I,J}(T)=\{L \mid \exists$ rule r: $L \leftarrow L_1,...,L_n$ in P s.t. r is not blocked w.r.t. I and $L_i \in T \cup J$, $\forall i \leq n\}$.
- $\mathbf{PT}_I(J)= \cup\{\text{PT}_{I,J} \uparrow^a(\emptyset) \mid a<\omega\}$, where ω is the first infinite ordinal.
- $\mathbf{PF}(J)$ is the greatest set S of classical literals s.t. $\forall L \in S$, if r is a rule in P s.t. $Head_r=L$ then $\exists L' \in Body_r$ s.t. $L' \in S$ or $\sim L' \in J\}$).
- $\mathbf{PW}_I(J)=coh(\mathbf{PT}_I(J) \cup \sim \mathbf{PF}(J))$.
- Pos_I is the least fixpoint of the operator $\mathbf{PW}_I$.

A default literal $\sim K$ is reliable w.r.t. I if there is no constraint violation caused by Pos_I that depends on $\sim K$. In other words, $\sim K$ is reliable if it is not "suspect" for any constraint violation. If r is a rule with $Body_r \subseteq \text{Pos}_I$ and a constraint violation depends on $Head_r$ then the constraint violation depends also on all literals in S_r. If a constraint violation depends on a default literal $\sim K$ then the constraint violation depends also on $\neg K$.

Definition 2.2.6 (dependency set w.r.t. I, reliable default literal): Let P be an *EPP*, L be a literal and I be a set of literals.

- The *dependency set* of L w.r.t. I, $\text{Dep}_I(L)$, is the least set $\text{D}(L)$ such that:
  - if L is the default literal $\sim K$ then $\{\sim K\} \subseteq \text{D}(\sim K)$ and $\text{D}(\neg K) \subseteq \text{D}(\sim K)$.
  - if $\exists$ r: $L \leftarrow L_1,...,L_n$ in P s.t. $Body_r \subseteq \text{Pos}_I$ then $\{L\} \subseteq \text{D}(L)$ and $\forall L_i \in S_r$, $\text{D}(L_i) \subseteq \text{D}(L)$.
- A default literal $\sim K$ is *unreliable* w.r.t. I iff $\exists$ $\bot \leftarrow L_1,...,L_n$ in P s.t. $\sim K \in \text{Dep}_I(L_i)$, for an $i \leq n$ and $L_j \in \text{Pos}_I$, $\forall j \in \{1,...,n\}-\{i\}$. Otherwise, $\sim K$ is *reliable* w.r.t. I.

Note that only the dependency sets of literals in S_r are considered in the computation of $Dep_I(Head_r)$. This is because even if r is "suspect" for a constraint violation caused by Pos_I, the rules and *CWAs* used in the derivation of $Body_r \neg S_r$ are not necessarily "suspect" for this constraint violation.

Example 2.2.3: Consider the *EPP*, $P=<R_P, IC_P, <_R>$:

$R_P=\{r_1: fly.\quad r_2: \neg fly \leftarrow \sim bird.$ with $S_{r_2}=\{\sim bird\}\}$, $IC_P=BC_P$ and $<_R=\{\}$.

The default literal $\sim bird$ is unreliable w.r.t. $I=\emptyset$ because $\bot \leftarrow fly, \neg fly$ is a constraint, $\sim bird \in Dep_I(\neg fly)$ and $fly \in Pos_I = coh(\{fly, \neg fly, \sim bird\}$. In case that $S_{r_2}=\{\}$, $\sim bird$ is reliable w.r.t. I because $\sim bird \notin Dep_I(\neg fly)$ and $\sim bird \notin Dep_I(fly)$.

To decide if a rule r is reliable w.r.t. I, all possible constraint violations caused by rules with priority no lower than r, should be considered. For this, the set of literals $Pos_{r,I}$ is computed. Intuitively, $Pos_{r,I}$ is the set of literals proved by $\{\sim L \leftarrow \neg L \mid L \in HB_P\}$ and the rules $r' \not< r$ when rules are blocked as indicated in I and the truth value of $Body_{r'} \neg S_{r'}$ is as indicated in Pos_I.

Let r, r' be rules. We define $r \equiv_R r'$ when $r''<r$ iff $r''<r'$, $\forall$ rule r''. The equivalence relation $\equiv_R$ partitions the rules in P into equivalence classes. The equivalence class of a rule r is denoted by $[r]$. When $r \equiv_R r'$, the set of rules with priority no lower than r is the same as the set of rules with priority no lower than r'. So, if $r \equiv_R r'$ then $Pos_{r,I} = Pos_{r',I}$. In other words, the literal set $Pos_{r,I}$ corresponds to the class of rules $[r]$.

Definition 2.2.7 (possible literal set w.r.t. $[r]$ and I): Let P be an *EPP*, r be a rule and I be a literal set. The *possible literal set* w.r.t. $[r]$ and I, $Pos_{r,I}$, is defined as follows:

- $P_{r,I}(Pos) = coh(\{Head_{r'} \mid \exists$ rule r' in P s.t. (i) $r' \not< r$, (ii) r' is not blocked w.r.t. I, (iii) $S_{r'} \subseteq Pos$ and (iv) $Body_{r'} \neg S_{r'} \subseteq Pos_I\})$
- $Pos_{r,I} = \cup\{P_{r,I}{}^{\uparrow a}(\emptyset) \mid a<\omega\}$, where ω stands for the first infinite ordinal.

A rule r is reliable w.r.t. I if there is no constraint violation caused by $\mathrm{Pos}_{r,I}$ that depends on r. Intuitively, r is reliable if it is not "suspect" for any constraint violation caused by rules $r' \not< r$. If a constraint violation depends on a rule r'' then the constraint violation depends also on all rules $r' \not< r$ with (i) $Head_{r'} \in S_{r''}$ or $\sim\neg Head_{r'} \in S_{r''}$, (ii) $S_{r'} \subseteq \mathrm{Pos}_{r,I}$ and (iii) $Body_{r'} \neg S_{r'} \subseteq \mathrm{Pos}_I$. In the computation of RS, the derivation of literals in $Body_{r'} \neg S_{r'}$ may be based on *CWAs* and rules $r' < r$ that are in conflict with r. This is because these *CWAs* and rules are not necessarily "suspect" for this conflict and thus not necessarily unreliable. Since only rules $r' \not< r$ are used in the computation of $\mathrm{Pos}_{r,I}$, the truth value of $Body_{r'} \neg S_{r'}$ should be as indicated by Pos_I. Intuitively, the truth value of $Body_{r'} \neg S_{r'}$ is independent of $<_R$.

Definition 2.2.8 (dependency set w.r.t. [r] and I, reliable rule): Let P be an *EPP*, r be a rule, L be a literal and I be a literal set.

- The *dependency set* of L w.r.t. [r] and I, $\mathrm{Dep}_{r,I}(L)$, is the least set $\mathrm{D}(L)$ such that:
  - if L is the default literal $\sim K$ then $\{\sim K\} \subseteq \mathrm{D}(\sim K)$ and $\mathrm{D}(\neg K) \subseteq \mathrm{D}(\sim K)$.
  - if r': $L \leftarrow L_1,\ldots,L_n$ in P s.t. (i) $r' \not< r$, (ii) $S_{r'} \subseteq \mathrm{Pos}_{r,I}$ and (iii) $Body_{r'} \neg S_{r'} \subseteq \mathrm{Pos}_I$ then $\{L\} \subseteq \mathrm{D}(L)$ and $\forall L_i \in S_{r'}$, $\mathrm{D}(L_i) \subseteq \mathrm{D}(L)$.
- A rule r is *unreliable* w.r.t. I iff (i) $S_r \subseteq \mathrm{Pos}_{r,I}$, (ii) $Body_r \neg S_r \subseteq \mathrm{Pos}_I$ and (iii) $\exists\ \bot \leftarrow L_1,\ldots,L_n$ in P s.t. $Head_r \in \mathrm{Dep}_{r,I}(L_i)$, for an $i \leq n$ and $L_j \in \mathrm{Pos}_{r,I}$, $\forall j \in \{1,\ldots,n\}-\{i\}$. Otherwise, r is *reliable* w.r.t. I.

Similarly to $\mathrm{Pos}_{r,I}$, if $r \equiv_R r'$ then $\mathrm{Dep}_{r,I}(L) = \mathrm{Dep}_{r',I}(L)$ $\forall$ literal L. Note that only the dependency sets of literals in $S_{r'}$ are considered in the computation of $\mathrm{Dep}_{r,I}(Head_{r'})$. This is because even if r' is "suspect" for a constraint violation caused by $\mathrm{Pos}_{r,I}$, the rules and *CWAs* used in the derivation of $Body_{r'} \neg S_{r'}$ are not necessarily "suspect" for this constraint violation.

Note that if $S_{r'} = Body_{r'}$ $\forall$ rule r' then $\mathrm{Pos}_{r,I}$ does not contain literals whose derivation is based on *CWAs*. This implies that no rule r is considered unreliable merely due to constraint violations caused by *CWAs*. Intuitively, when $S_{r'} = Body_{r'}$ $\forall$ rule r', every rule is given higher

priority than the *CWAs*. In Example 2.2.4, we show that this is not true when there is a literal $L \in Body_{r'} \neg S_{r'}$ for a rule r' and $L \in Pos_I$.

Example 2.2.4: Let P be as in Example 2.2.3, i.e.,

$R_P=\{r_1: fly.\quad r_2: \neg fly \leftarrow \sim bird.$ with $S_{r_2}=\{\sim bird\}\}$, $IC_P=BC_P$ and $<_R=\{\}$.

In Example 2.2.3, we showed *~bird* is unreliable w.r.t. $I=\emptyset$. Thus, we expect the literal *~bird* to be evaluated as unknown. The rule r_1 is reliable w.r.t. I because $\text{Pos}_{r_1,I} = coh(\{fly\})$ and $Head_{r_1} \notin \text{Dep}_{r_1,I}(\neg fly)$. Thus, we expect the literal *fly* to be evaluated as true. Rule r_2 is reliable w.r.t. I because $S_{r_2}=\{\sim bird\}$ is not a subset of $\text{Pos}_{r_2,I} = coh(\{fly\})$.

In case that $S_{r_2}=\{\}$ then *~bird* is reliable w.r.t. I and $\text{Pos}_{r_1,I} = \text{Pos}_{r_2,I} = coh(\{fly, \neg fly\})$. The rule r_1 is unreliable w.r.t. I because $Head_{r_1} \in \text{Dep}_{r_1,I}(fly)$. The rule r_2 is unreliable w.r.t. I because $Head_{r_2} \in \text{Dep}_{r_2,I}(\neg fly)$ and $Body_{r_2} \neg S_{r_2} = Body_{r_2} \subseteq \text{Pos}_I$. Thus, we expect the literals *fly*, $\neg fly$ to be evaluated as unknown and the literal *~bird* to be evaluated as true.

Note that when $S_{r_2}=\{\sim bird\}$, rule r_1 is reliable and *~bird* is unreliable w.r.t. I. Intuitively, if $S_{r_2}=\{\sim bird\}$ then rule r_1 is given higher priority than *CWA ~bird*. This is not true when $S_{r_2}=\{\}$, i.e., rule r_1 is unreliable and *~bird* is reliable w.r.t. I.

Example 2.2.5: Let P be as in Example 2.2.1 and $I=\emptyset$. Then, rule r_2 is reliable w.r.t. I since $\text{Pos}_{r_2,I}$ = *coh*({TA(ann), foreign_stud(ann), need_credits(ann,6)}) and $Head_{r_2} \notin \text{Dep}_{r_2,I}$(need_credits(ann,12))={}. Though $\text{Pos}_{r_4,I}$ = *coh*({TA(ann), foreign_stud(ann), need_credits(ann,6), need_credits(ann,12)}) violates the constraint *ic*, rule r_4 is reliable w.r.t. I since $Head_{r_4} \notin \text{Dep}_{r_4,I}$(need_credits(ann,X))= {need_credits(ann,X)} for X=6,12. Similarly to r_4, rule r_3 is reliable w.r.t. I. Rule r_1 is unreliable w.r.t. I since need_credits(ann,6)$\in \text{Pos}_{r_1,I}$, $Head_{r_1} \in \text{Dep}_{r_1,I}$(need_credits(ann,12)) and $Body_{r_1} \neg S_{r_1} = Body_{r_1} \subseteq \text{Pos}_I$.

If S_{r_1}={foreign_stud(ann)} then the rule r_4 is unreliable w.r.t. I because $Head_{r_4} \in \text{Dep}_{r_4,I}$(need_credits(ann,12)). However, for any value of S_{r_1}, if $r_2 < r_4$ then $\text{Pos}_{r_4,I} = coh$({TA(ann), foreign_stud(ann)}) and thus rule r_4 is reliable w.r.t. I. Similarly, for any value of S_{r_1}, if $r_1 < r_4$ or $r_3 < r_4$ then rule r_4 is reliable w.r.t. I.

Definition 2.2.9 (truth valuation of a rule): Let P be an *EPP*. A rule r is *r-true* w.r.t. an interpretation I iff: (i) $I(Head_r) \geq I(Body_r)$ or (ii) $I(Body_r)=1/2$ and $I(\neg Head_r)=1$ or (iii) $I(Body_r)=1$ and ($I(Head_r)=1/2$ or $I(\neg Head_r)=1$) and r is unreliable w.r.t. I.

Definition 2.2.10 (*r*-model): Let P be an *EPP*. A consistent, coherent interpretation I of P is an *r-model* of P iff every rule in P is r-true w.r.t. I.

Example 2.2.6: Let P be as in Example 2.2.4. Then, $M=coh(\{fly\})$ is an r-model of P. We will show that *~bird* is not true in any r-model of P. Let M' be an r-model of P. Then, $fly \in M'$ because r_1 is reliable w.r.t. $\emptyset$ and consequently r_1 is reliable w.r.t. $M' \supseteq \emptyset$. This implies that $\neg fly \notin M'$ because M' is a consistent interpretation of P. So, $\sim bird \notin M'$ because otherwise $\neg fly \in M'$ since r_2 is reliable w.r.t. $M' \supseteq \emptyset$. The literal $\sim\neg fly$ should also belong to M' because M' is a coherent interpretation. In case that $S_{r_2}=\{\}$, the r-models are $M_1=\{\sim bird\}$, $M_2=coh(\{fly,\sim bird\})$ and $M_3=coh(\{\neg fly,\sim bird\})$.

Example 2.2.7: Let P be as in Example 2.2.1. Then, $M=coh(\{$TA(ann), foreign_stud(ann), need_credits(ann,6)$\})$ is an r-model of P. We will show that M is the unique r-model of P. In Example 2.2.5, we showed that rules r_2, r_3 and r_4 are reliable w.r.t. $I=\emptyset$. Let M' be an r-model of P. Then, r_2, r_3 and r_4 are reliable w.r.t. $M' \supseteq \emptyset$. So, the literals TA(ann), foreign_stud(ann), need_credits(ann,6) belong to M'. The literal need_credits(ann,12)$\notin M'$ because otherwise M' will violate the constraint *ic*.

Let P be a normal program and I be an interpretation as defined in [58, 60]. In [60], a rule r is true w.r.t. I iff $I(Head_r) \geq I(Body_r)$. Since P is a normal program, rules do not contain classically negative literals and the only constraints are the basic constraints. So, every rule in P is reliable w.r.t. $I' = I \cup \{\sim\neg A | A$ is an atom of $P\}$ and conditions (ii) and (iii) in Definition 2.2.9 are not satisfied by I', for all rules in P. This implies that a rule r in P is r-true w.r.t. I' iff r is true w.r.t. I.

Proposition 2.2.1: Let P be a normal program. M is a model of P iff $M \cup \{\sim\neg A | A$ is an atom of $P\}$ is an r-model of P.

The partial order $<_R'$ is an extension of the partial order $<_R$ iff $(r,r') \in <_R$ implies $(r,r') \in <_R'$. Let $P=<R_P,IC_P,<_R>$ and $P'=<R_P,IC_P,<_R'>$ be *EPPs*, where $<_R'$ is an extension of $<_R$. It is desirable that any r-model of P' is an r-model of P. This is because, if the reliabilities of rules r and r' cannot be compared then both $r<r'$ and $r'<r$ are possible. So, any extension of $<_R$ is possible to express the relative reliability of the rules in R_P.

Proposition 2.2.2: Let $P=<R_P,IC_P,<_R>$ be an *EPP* and $<_R'$ be an extension of $<_R$. Every r-model of $P'=<R_P,IC_P,<_R'>$ is an r-model of P.

Proof: Let M be an r-model of P'. Then, M is a consistent, coherent interpretation of P. If r is a rule in P then r is r-true w.r.t. M in P'. We will show that r is r-true w.r.t. M in P. It is enough to show that if r is unreliable w.r.t. M in P' then r is unreliable w.r.t. M in P. Assume that r is unreliable w.r.t. M in P'. Since $<_R'$ is an extension of $<_R$, the set of rules with priority no lower than r in P' is a subset of that in P. So, Pos_M, $\text{Pos}_{r,M}$ and $\text{Dep}_{r,M}(L)$ in P' are subsets of the corresponding sets in P. This implies that r is unreliable w.r.t. M in P. ◊

2.3 Reliable Semantics

In this Section, we define the *reliable model*, *stable r-models* and *reliable semantics* of an *EPP*, P. We define the reliable model of P as the least fixpoint of a monotonic operator. We show that reliable model of P is the least stable r-model of P.

2.3.1 Reliable Model

In the computation of reliable model of P, RM_P, a default literal $\sim L$ is true by *CWA* only if $\sim L$ is reliable w.r.t. RM_P. A rule r is used for the derivation of $Head_r$ only if r is a reliable rule w.r.t. RM_P.

The definition of an r-unfounded set for an *EPP* extends that of an unfounded set for a normal program [76]. If S is an r-unfounded set w.r.t. a literal set J then $\forall L \in S$, $\sim L$ is reliable w.r.t. J. Note that if P is a normal program then all default literals of P are reliable w.r.t. any literal set J.

Definition 2.3.1 (*r*-unfounded set): Let P be an *EPP* and J be a set of literals. A set S of classical literals is *r*-unfounded w.r.t. J iff $\forall L \in S$, (i) if r is a rule in P with $Head_r = L$ then $\exists L' \in Body_r$ s.t. $L' \in S$ or $\sim L' \in J$ and (ii) $\sim L$ is reliable w.r.t. J.

The $\mathbf{W}_P$ operator for *EPPs* extends the $\mathbf{W}_P$ operator for normal programs [76]. This is because if P is a normal program and J a literal set then (i) every rule is reliable w.r.t. J and (ii) a set S is an *r*-unfounded set w.r.t. J iff S is an unfounded set w.r.t. J.

Definition 2.3.2 ($\mathbf{W}_P$ operator): Let P be an *EPP* and J be a set of literals. We define:

- $\mathbf{T}_J(T) = \{L \mid \exists$ rule $r: L \leftarrow L_1,\ldots,L_n$ in P s.t. (i) $L_i \in T \cup J$, $\forall i \leq n$ and (ii) r is reliable w.r.t. $J\}$.
- $\mathbf{T}(J) = \cup\{\mathbf{T}_J \uparrow^a(\emptyset) \mid a < \omega\}$, where ω is the first limit ordinal.
- $\mathbf{F}(J)$ is the greatest *r*-unfounded set w.r.t. J.
- $\mathbf{W}_P(J) = coh(\mathbf{T}(J) \cup \sim \mathbf{F}(J))$.

The sequence $\{\mathbf{T}_J \uparrow^a\}$ is monotonically increasing (w.r.t. $\subseteq$). So, $\mathbf{T}(J)$ is the least fixpoint of the operator $\mathbf{T}_J$. The union of two *r*-unfounded sets w.r.t. an interpretation J is an *r*-unfounded set w.r.t. J. So, $\mathbf{F}(J)$ is the union of all *r*-unfounded sets w.r.t. J. We define the transfinite sequence $\{I_a\}$ as follows: $I_0 = \{\}$, $I_{a+1} = \mathbf{W}_P(I_a)$ and $I_a = \cup\{I_b \mid b < a\}$ if a is a limit ordinal.

Proposition 2.3.1: Let P be an *EPP*. $\{I_a\}$ is a monotonically increasing (w.r.t. $\subseteq$) sequence of consistent, coherent interpretations of P.

Proof: We will show that $\mathbf{W}_P$ is a monotonic operator. Let I,J be interpretations of P s.t. $I \subseteq J$. $\mathbf{T}(I) \subseteq \mathbf{T}(J)$ follows from the fact that if a rule r is reliable w.r.t I then r is reliable w.r.t. J. $\mathbf{F}(I) \subseteq \mathbf{F}(J)$ follows from the fact that if a default literal $\sim L$ is reliable w.r.t. I then $\sim L$ is reliable w.r.t. J. Since *coh* is a monotonic operator, $\mathbf{W}_P$ is a monotonic operator and $\{I_a\}$ is a monotonically increasing sequence w.r.t. $\subseteq$.

We will prove by induction that for all a, there is no constraint ic' s.t. $Body_{ic'} \subseteq I_a$. This is true for $a = 0$. Assume that it is true for ordinals $< a$. We will prove that it is true for a.

Assume first that $a=b+1$ is a successor ordinal. Let $a'=b'+1$ be the first ordinal s.t. $T_{I_b}{}^{\uparrow a'}(\emptyset)$ is inconsistent. If $T_{I_b}{}^{\uparrow a'}(\emptyset)$ violates a basic constraint then let ic be one of the violated basic constraints. Otherwise, let ic be any of the violated constraints. Let $R_{b,a'} = \{r \mid$ rule r is fired in the computation of $T(I_c)$ for $c< b$ or in the computation of $T_{I_b}{}^{\uparrow c}(\emptyset)$ for $c\leq a'\}$. Let $K\in Body_{ic}$ s.t. $K\in T_{I_b}{}^{\uparrow a'}(\emptyset)$, $K\notin I_b$ and $K\notin T_{I_b}{}^{\uparrow b'}(\emptyset)$. Such a literal exists since a' is the first ordinal s.t. $T_{I_b}{}^{\uparrow a'}(\emptyset)$ is inconsistent.

Case 1: *The constraint ic is the basic constraint* $\bot\leftarrow K, \neg K$.

Choose the smallest c' s.t. $I_{c'+1}$ contains $\neg K$. We will show that there is no L s.t. $L\in F(I_c)$, $c< c'$, and $\sim L\in Dep_{I_c}(\neg K)$. Since $\forall c< c'$, $\neg K\notin I_c$ and there is no literal K' s.t. $K'\in I_c$ and $\neg K'\in T_{I_b}{}^{\uparrow b'}(\emptyset)$, it follows that $\forall c< c'$, $K\in Pos_{I_c}$. If there is L, c s.t. $L\in F(I_c)$, $c < c'$, and $\sim L\in Dep_{I_c}(\neg K)$ then $\sim L$ is unreliable w.r.t. I_c which is a contradiction (all literals in $\sim F(I_c)$ are reliable w.r.t I_c). Similarly, there is no L s.t. $L\in F(I_c)$, $c< b$, and $\sim L\in Dep_{I_c}(K)$. This implies that there is a rule $r_m\in R_{b,a'}$ which is used in the derivation of K, $\neg K\in Pos_{r_m,I_b}$ and $Head_{r_m}\in Dep_{r_m,I_b}(K)$. Moreover, $S_{r_m}\subseteq Pos_{r_m,I_b}$ and $Body_{r_m}\neg S_{r_m}\subseteq Pos_{I_b}$. Thus, rule r_m is unreliable w.r.t. I_b which is a contradiction.

Case 2: *The constraint ic is not a basic constraint.*

Then, there is no literal K' s.t. $K'\in I_c$, $c< b$, and $\neg K'\in T_{I_b}{}^{\uparrow a'}(\emptyset)$. This implies that $\forall c<b$, $T_{I_b}{}^{\uparrow a'}(\emptyset)\subseteq Pos_{I_c}$ and consequently, $Body_{ic}\subseteq Pos_{I_c}$. We will show that there is no L s.t. $L\in F(I_c)$, $c<b$, and $\sim L\in Dep_{I_c}(K)$, for a $K\in Body_{ic}$. Assume that there is L, c s.t. $L\in F(I_c)$, $c< b$, and $\sim L\in Dep_{I_c}(K)$, for a $K\in Body_{ic}$. Then, $Body_{ic}\subseteq Pos_{I_c}$ and $\sim L\in Dep_{I_c}(K)$, for a $K\in Body_{ic}$. Consequently, $\sim L$ is unreliable w.r.t. I_c which is a contradiction (all literals in $\sim F(I_c)$ are reliable w.r.t I_c). Let R_{ic} be the set of the rules $r\in R_{b,a'}$ which are used in the derivation of literals in $Body_{ic}$ and $Head_r\in Dep_{r,I_b}(K)$, for a $K\in Body_{ic}$. Let $r_m\in R_{ic}$ be s.t. there is no $r\in R_{ic}$ and $r<r_m$. Then, $Body_{ic}\subseteq Pos_{r_m,I_b}$ and $Head_{r_m}\in Dep_{r_m,I_b}(K)$, for a $K\in Body_{ic}$. Moreover, $S_{r_m}\subseteq Pos_{r_m,I_b}$, $Body_{r_m}\neg S_{r_m}\subseteq Pos_{I_b}$. Consequently, rule r_m is unreliable w.r.t. I_b which is a contradiction.

So, I_a does not violate any constraint.

Let a be a limit ordinal and assume that there is constraint ic in P s.t. $Body_{ic} \subseteq I_a$. Then, there is a successor ordinal $b+1<a$ s.t. $Body_{ic} \subseteq I_{b+1}$. This is a contradiction because of the inductive hypothesis. So, I_a is consistent for all a.

We will prove by induction that for all a, there is no literal L s.t. $L \in I_a$ and $\sim L \in I_a$. The proof is similar to that of the well-founded semantics [76]. It is true for $a=0$. Assume that it is true for ordinals $<a$. We will prove that it is true for a.

Assume first that $a=b+1$ is a successor ordinal. We will prove by a second induction that for all a', there is no literal L s.t. $L \in T_{I_b}{}^{\uparrow a'}(\emptyset)$ and $\sim L \in I_a$. This is true for $a'=0$. Assume this is true for ordinals $<a'$. Let $a' = b'+1$ is a successor ordinal. Let S be any set of classical literals that has a non-empty intersection with $T_{I_b}{}^{\uparrow a'}(\emptyset)$. Choose the smallest c s.t. I_{c+1} has a non-empty intersection with S and the smallest c' s.t. $T_{I_c}{}^{\uparrow c'+1}(\emptyset)$ has a non-empty intersection with S. Note that $c<b$ or $c=b$ and $c'<a'$. Let $L \in T_{I_c}{}^{\uparrow c'+1}(\emptyset) \cap S$. Then, L is derived from a rule r s.t. $Body_r \subseteq T_{I_c}{}^{\uparrow c'}(\emptyset) \cup I_c$. From hypothesis, there is no literal $K \in Body_r$, s.t. $\sim K \in I_b$. Moreover, from the way r is defined, there is no classical literal K in $Body_r$ s.t. $K \in S$. So, S is not r-unfounded w.r.t. I_b. This implies that $T_{I_b}{}^{\uparrow a'}(\emptyset) \cap F(I_b) = \emptyset$. So, $T(I_b) \cap F(I_b) = \emptyset$. Moreover, there is no classical literal L s.t. $L \in T(I_b)$ and $\neg L \in T(I_b)$, because I_a does not violate any constraint. So, there is no literal L s.t. $L \in I_a$ and $\sim L \in I_a$.

Let a be a limit ordinal and assume that there is L s.t. $L \in I_a$ and $\sim L \in I_a$. Then, there is a successor ordinal $b+1<a$ s.t. $L \in I_{b+1}$ and $\sim L \in I_{b+1}$. This is a contradiction because of the inductive hypothesis.

I_a is a coherent interpretation, for all a, because of the *coh* operator in the definition of W_P. Proposition 2.3.1 follows. ◊

Since $\{I_a\}$ is monotonically increasing (w.r.t. $\subseteq$), there is a smallest countable ordinal d s.t. $I_d = I_{d+1}$ [23].

Proposition 2.3.2: Let P be an *EPP*. Then, I_d is an r-model of P.

Proof: From Proposition 2.3.1, I_d is a consistent, coherent interpretation. Let r be a rule in P. We will show that r is r-true w.r.t. I_d.

(i) If $I_d(Body_r)=1/2$ and $I_d(Head_r)=0$ then $I_d(\neg Head_r)=1$ because otherwise $I_d(Head_r)=1/2$.

(ii) If $I_d(Body_r)=1$ and $I_d(Head_r)=1/2$ then r is unreliable w.r.t. I_d because otherwise, from the definition of $\mathbf{T}(I_d)$, $I_d(Head_r)=1$.

(iii) If $I_d(Body_r)=1$ and $I_d(Head_r)=0$ then r is unreliable w.r.t. I_d because otherwise, from the definition of $\mathbf{T}(I_d)$, $I_d(Head_r)=1$. Since r is unreliable w.r.t. I_d, it follows that $I_d(\neg Head_r)=1$ because otherwise, from the definition of $\mathbf{F}(I_d)$, $I_d(Head_r)=1/2$.

(iv) In all the other cases, $I_d(Head_r)\geq I_d(Body_r)$. ◊

Definition 2.3.3 (reliable semantics): Let P be an *EPP*. The *reliable model* of P, denoted as RM_P, is the interpretation I_d. The *reliable semantics* of P is the "meaning" represented by RM_P.

Example 2.3.1: Consider the *EPP*, $P=\langle R_P, IC_P, <_R \rangle$:

$R_P=\{r_1: q. \quad r_2: p \leftarrow q. \quad r_3: \neg p. \quad r_4: p \leftarrow \sim r.$ with $S_{r_i}=Body_{r_i}, \forall i \leq 4\}$,

$IC_P=BC_P$ and $r_3<r_2$, $r_2<r_1$, $r_3<r_1$.

Computation of $\mathbf{W}_P(\emptyset)$: Rule r_1 is reliable w.r.t. $\emptyset$ because it has higher priority than rules r_2 and r_3 and rules r_1, r_4 do not generate a constraint violation. Similarly, rule r_2 is reliable w.r.t. $\emptyset$. In contrast, rule r_3 is unreliable w.r.t. $\emptyset$ because $p \in \mathrm{Pos}_{r_3,\emptyset}=coh(\{q, p, \neg p\})$ and $\mathrm{Head}_{r_3} \in \mathrm{Dep}_{r_3,\emptyset}(\neg p)$. So, $\mathbf{T}(\emptyset)=\{q, p\}$. The literal $\sim r$ is unreliable w.r.t. $\emptyset$ because $\neg p \in \mathrm{Pos}_{\emptyset}=coh(\{q, p, \neg p, \sim r\})$ and $\sim r \in \mathrm{Dep}_{\emptyset}(p)$. So, $\mathbf{F}(\emptyset)=\{\}$ and $\mathbf{W}_P(\emptyset)=coh(\{p,q\})$.

Computation of $\mathbf{W}_P{}^{\uparrow 2}(\emptyset)$: Rule r_3 is unreliable w.r.t. $\mathbf{W}_P(\emptyset)$. So, $\mathbf{T}(\mathbf{W}_P(\emptyset))=\mathbf{T}(\emptyset)$. However, $\sim r$ is reliable w.r.t. $\mathbf{W}_P(\emptyset)$ because r_3 is blocked w.r.t. $\mathbf{W}_P(\emptyset)$ and consequently, $\neg p \notin \mathrm{Pos}_{\emptyset}$. So, $\mathbf{F}(\mathbf{W}_P(\emptyset))=\{\sim r\}$ and $\mathbf{W}_P{}^{\uparrow 2}(\emptyset)=coh(\{p, q, \sim r\})$.

Computation of $\mathbf{W}_P{}^{\uparrow 3}(\emptyset)$: Because r_3 is unreliable w.r.t. $\mathbf{W}_P{}^{\uparrow 2}(\emptyset)$, it follows that $\mathbf{W}_P{}^{\uparrow 3}(\emptyset)=\mathbf{W}_P{}^{\uparrow 2}(\emptyset)$.

So, $RM_P=\mathbf{W}_P{}^{\uparrow 2}(\emptyset)$.

Example 2.3.2: Let P be the program of Example 2.2.1. Then, the interpretation *coh*({TA(ann), foreign_stud(ann), need_credits(ann,6)}) is the reliable model of P. If P' is as P with $<_R$={} then the reliable model of P' is *coh*({TA(ann), foreign_stud(ann)}) which corresponds to the skeptical meaning of P'. If P' is as P with $S_r = Body_r$ $\forall$ rule r then rules r_3 and r_4 are unreliable w.r.t. $\emptyset$ and thus the reliable model of P' is {}.

Proposition 2.3.3: Let $P=<R_P, IC_P, <_R>$ be an *EPP*. The complexity of computing RM_P is $O(|HB_P|*|R_P|*\max(|IC_P|, |HB_P|*|EC_R|)$, where EC_R is the set of equivalence classes of R_P w.r.t. $\equiv_R$.

Proof: The following algorithm, **RM**(program P), returns the reliable model of P. To compute **F**(I), its complement set is constructed first, as in [76].

```
RM(EPP program P)
{ new_I={};
  repeat
    I=new_I;
    compute Pos_I;   /* Step 1 */
    for each L∈HB_P do compute Dep_I(L); endfor   /* Step 2 */
    for each [r] in P do /* Step 3 */
      compute Pos_{r,I};   /* Step 3.1 */
      for each L∈HB_P do compute Dep_{r,I}(L); endfor   /* Step 3.2 */
    endfor

    repeat          /* Step 4: Compute T(I) */
       for each rule r in P do
          if Body_r ⊆new_I and r is reliable w.r.t. I then add Head_r to new_I; endif
       endfor
    until no change in new_I;

    compl_F={L∈HB_P | ~L is unreliable w.r.t. I};     /* Step 5 */
    repeat          /* Step 6: Compute HB_P –F(I) */
       for each rule r in P do
         if no literal in Body_r is false w.r.t. I and all classical literals in Body_r are in compl_F
         then add H_r to compl_F;
```

```
        endif
      endfor
    until no change in compl_F;
    for each L∈HB_P do /* Step 7*/
      if L∉compl_F then add ~L to new_I; endif
    endfor

    new_I=coh(new_I);  /* Step 8: Compute coh(T(I)∪~F(I)) */
  until I=new_I;
  return(I);
}
```

The complexity of computing Pos_I is the same as that of computing the well-founded model of P when every literal $\neg L$ is replaced by a new atom $\neg\_L$. So, the complexity of Step 1 is $|HB_P|*|R_P|$ [78, 69]. The complexity of Step 2 is $|HB_P|*|R_P|$ because the complexity of computing $\mathrm{Dep}_I(L)$, for a literal L, is $|R_P|$. The complexity of Step 3.1 is $|R_P|$ and that of Step 3.2 is $|HB_P|*|R_P|$. So, the complexity of Step 3 is $|EC_R|*|HB_P|*|R_P|$. The complexity of Step 4 is $|IC_P|*|R_P|$ since $\mathrm{Pos}_{r,I}$ and $\mathrm{Dep}_{r,I}(L)$, $\forall L\in HB_P$, have already been computed. The complexity of Step 5 is $|IC_P|*|HB_P| < |IC_P|*|R_P|$ and that of Step 6 is $|R_P|$ [19]. The complexity of Steps 7 and 8 is $|HB_P|$. Since $\{I_a\}$ is a monotonically increasing sequence w.r.t. $\subseteq$, the total number of iterations until $I=new\_I$, is less than $|HB_P|$. So, the complexity of the algorithm $\mathbf{RM}(P)$ is $O(|HB_P|*|R_P|*\max(|IC_P|, |HP_P|*|EC_R|)$.

◊

2.3.2 Stable *r*-models

The reliable model of an *EPP* corresponds to its skeptical meaning. Credulous meanings can be obtained using the transformation $P/_r I$, where I is an interpretation of P. The transformation P/I for a normal program P is defined in [26, 60]. $P/_r I$ extends P/I to *EPPs*.

Definition 2.3.4 (transformation $P/_r I$): Let P be an *EPP* and I be an interpretation of it. The program $P/_r I$ is obtained as follows:

(i) Remove from P all rules that contain in their body a default literal $\sim L$ s.t. $I(L)=1$.

(ii) Remove from P any rule r with $I(\neg Head_r)=1$.

(iii) If r is a rule in P s.t. $I(Body_r)=1$ and $I(Head_r)=1/2$ then replace r with $Head_r \leftarrow u$.

(iv) Remove from the body of the remaining rules of P any default literal $\sim L$ s.t. $I(L)=0$.

(v) Replace all remaining default literals $\sim L$ with u.

(vi) If $I(L)=1/2$ and $\sim L$ is unreliable w.r.t. I then add the rule $L \leftarrow u$.

(vii) Replace every classically negative literal $\neg A$ with a new atom $\neg\_A$.

The program $P/_r I$ is a non-negative program with a special proposition u. For any interpretation J, $J(u)=1/2$. When P is a normal program and M is a model of P [60], $P/_r M \equiv P/M$ since Steps (ii), (iii), (vi) and (vii) do not have any effect on $P/_r M$.

We say that a model M of P is the *least$_v$ model* of P iff $M(L) \leq M'(L)$ for any model M' and classical literal L of P. The least$_v$ model of a non-negative program can be obtained as the least$_v$ fixpoint of the Ψ_P operator [60] which generalizes the immediate consequence operator of [75].

Definition 2.3.5 (Ψ_P operator) [60]: Let P be a non-negative program, I be an interpretation and A be an atom of P. $\Psi_P(I)$ is defined as follows:

(i) $\Psi_P(I)(A)=1$ if $\exists$ rule $A \leftarrow A_1,...,A_n$ in P s.t. $I(A_i)=1$, $\forall i \leq n$.

(ii) $\Psi_P(I)(A)=1/2$ if $\Psi_P(I)(A) \neq 1$ and $\exists$ rule $A \leftarrow A_1,...,A_n$ in P s.t. $I(A_i) \geq 1/2$, $\forall i \leq n$.

(iii) $\Psi_P(I)(A)=0$, otherwise.

Definition 2.3.6 (stable *r*-model): Let P be an *EPP* and M be an *r*-model of P. M is a *stable r-model* of P iff $least_v(P/_r M)=M$.

Stable *r*-models represent the credulous "meanings" of a program. For example, let P' be as the program P of Example 2.2.1 with $<_R=\{\}$. The stable *r*-models of P' are:

$M_1 = coh(\{$TA(ann), foreign_stud(ann), need_credits(ann,6)$\})$,

$M_2 = coh(\{$TA(ann), foreign_stud(ann), need_credits(ann,12)$\})$, and

$M_3 = RM_{P'} = coh(\{$TA(ann), foreign_stud(ann)$\})$.

The unique stable *r*-model of program P of Example 2.2.1 is:

$RM_P = coh(\{$TA(ann), foreign_stud(ann), need_credits(ann,6)$\})$.

Proposition 2.3.4: Let P be an *EPP*. The reliable model of P is a stable r-model of P.

Proof: Let RM be the reliable model of P. From Proposition 2.3.2, RM is an r-model of P. So, it is enough to show that $RM=least_v(P/_rRM)$. Let $least_v(P/_rRM)=T\cup\sim F$, where T, F are sets of classical literals. Let $I_a=T_a\cup\sim F_a$, where T_a, F_a are sets of classical literals and $RM = I_d$. First, we will prove by induction that $T_b\cup\sim F_b \subseteq T\cup\sim F$, $\forall b\leq d$. It is true that $T_0\subseteq T$ and $F_0\subseteq F$. Suppose that $T_a\subseteq T$ and $F_a\subseteq F$, $\forall a<b$. If b is a limit ordinal then $T_b\subseteq T$ and $F_b\subseteq F$ since $I_b= \cup\{I_a|\ a<b\}$. Assume therefore that $b=a+1$. It is true that $T_{I_a}{}^{\uparrow 0}(\emptyset)\subseteq T$. Assume that $T_{I_a}{}^{\uparrow a'}(\emptyset)\subseteq T$, we will show that $T_{I_a}{}^{\uparrow a'+1}(\emptyset)\subseteq T$. Let $L\in T_{I_a}{}^{\uparrow a'+1}(\emptyset)$. Then, $\exists\ r: L\leftarrow L_1,...,L_n$ in P s.t. r is reliable w.r.t. I_a and $\forall i\leq n$ either (i) $L_i\in I_a$ or (ii) L_i is a classical literal and $L_i\in T_{I_a}{}^{\uparrow a'}(\emptyset)$. Since $I_a\subseteq T\cup\sim F$ and $L\in RM$, there is a rule $L\leftarrow L'_1,...,L'_m$ in $P/_rRM$ where $L'_1,...,L'_m$ are all the classical literals in $\{L_1,...,L_n\}$. From the facts $T_{I_a}{}^{\uparrow a'}(\emptyset)\subseteq T$, $I_a\subseteq T\cup\sim F$ and the definition of $least_v(P/_r RM)$, it follows that $L\in T$. This implies that $T(I_a)=T_b\subseteq T$.

Now, we will show that $F_b\subseteq F$. Since $F_b= \neg T_b\cup F(I_a)$, it is enough to show that $\neg T_b\subseteq F$ and $F(I_a)\subseteq F$. If $L\in\neg T_b$ then $\neg L\in RM$ and from Step (ii) of Def. 2.3.4, $L\in F$. Consequently, $\neg T_b\subseteq F$. For all rules $H\leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in P (L_i, L'_i are classical literals) with $H\in F(I_a)$ either $\exists i\leq m$, $L'_i\in F(I_a)\cup F_a$ or $\exists j\leq n$, $L_j\in T_a$. This implies that, for each rule $H\leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in P with $H\in F(I_a)$ either there is a corresponding rule $H\leftarrow A_1,...,A_k$ in $P/_rRM$ (from Steps (iv) and (v) of Def. 2.3.4) with $A_i\in F(I_a)\cup F$ for an $i\leq k$ or there is no corresponding rule in $P/_rRM$ (from the Steps (i) and (ii) of Def. 2.3.4). Note that, no rule $H\leftarrow u$ is added to $P/_rRM$ (from Steps (iii) and (vi) of Def. 2.3.4) because H is false w.r.t. RM. So, for each rule $H\leftarrow A_1,...,A_k$ in $P/_rRM$ with $H\in F(I_a)\cup F$, $\exists i\leq k$ such that $A_i\in F(I_a)\cup F$. From the definition of $least_v(P/_rRM)$, it follows that $F(I_a)\subseteq F$. Consequently, $F_b\subseteq F$.

So, we proved that $T_d\subseteq T$ and $F_d\subseteq F$.

We will show that $T\subseteq T_d$. Let a be the first ordinal s.t. there is a literal $L\notin T_d$ and $\Psi_{P'}{}^{\uparrow a+1}(\emptyset)(L)=1$, where $P'=P/_rRM$. Then, there is a rule r: $L\leftarrow A_1,...,A_k$ in $P/_rRM$ with

$\Psi_{P'}{}^{\uparrow a}(\emptyset)(A_i)=1$, $\forall i \leq k$. This implies that there is a rule in P whose body literals are true w.r.t. RM. Since $L \notin T_d$, it follows that $\neg L \in T_d$ or L is unknown w.r.t. RM. If $\neg L \in T_d$ then from Step (ii) of Def. 2.3.4, $L \notin T$ which is a contradiction. If L is unknown w.r.t.RM, the rule r should not exist in $P/_r RM$ because of the Step (iii) of Def. 2.3.4 and the fact that all of the body literals of r are true w.r.t. $\Psi_{P'}{}^{\uparrow a}(\emptyset)$ and thus w.r.t. RM. So, $L \in T_d$ and consequently $T \subseteq T_d$.

We will show that $F \subseteq F_d$. Let $F_{coh}=\{H \mid \neg H \in T_d\}$. $F_{coh} \subseteq F_d$ because RM is a coherent interpretation. For all rules $H \leftarrow A_1,...,A_k$ in $P/_r RM$ with $H \in F-F_{coh}$, there is $i \leq k$ such that $A_i \in F$. This implies that for each rule $H \leftarrow L'_1,...,L'_m, \sim L_1,...,\sim L_n$ in P (L_i, L'_i are classical literals) with $H \in F-F_{coh}$ either (i) $\exists i \leq m$, $L'_i \in F$ (from Steps (iv) and (v) of Def. 2.3.4) or (ii) $\exists j \leq n$, $L_j \in T_d$ (from Step (i) of Def. 2.3.4). We will show that $\forall H \in F-F_{coh}$, $\sim H$ is reliable w.r.t. RM. If $\sim H$ is unreliable w.r.t RM then $H \notin \mathbf{F}(I_d)$ and consequently, $RM(H) \geq 1/2$. However, if $H \in F$ then $H \notin T$ and consequently $RM(H) \neq 1$. So, $RM(H)=1/2$ and the rule $H \leftarrow u$ should be added to $P/_r RM$ (from Step (vi) of Def. 2.3.4). This implies that $H \notin F$, which is a contradiction. So, $\forall H \in F-F_{coh}$, $\sim H$ is reliable w.r.t. RM. Since $\mathbf{F}(I_d)$ is the maximum set that satisfies the property satisfied by $F-F_{coh}$, $F-F_{coh} \subseteq \mathbf{F}(I_d)$. So, $F \subseteq F_d$.

Consequently, $RM=T_d \cup \sim F_d = T \cup \sim F = least_v(P/_r RM)$. ◊

Proposition 2.3.5: Let P be an *EPP*. The reliable model of P is the least stable r-model of P.

Proof: Let RM be the reliable model of P. From Proposition 3.4, RM is a stable r-model of P. So, it is enough to show that if M is a stable r-model of P then $RM \subseteq M = least_v(P/_r M)$. Let $M=T \cup \sim F$, where T, F are sets of classical literals. Let $I_a=T_a \cup \sim F_a$, where T_a, F_a are sets of classical literals and $RM = I_d$. We will show by induction that $I_b \subseteq T \cup \sim F$, $\forall b \leq d$. It is true that $T_0 \subseteq T$ and $F_0 \subseteq F$. Suppose that $T_a \subseteq T$ and $F_a \subseteq F$, $\forall a<b$. If b is a limit ordinal then $T_b \subseteq T$ and $F_b \subseteq F$ since $I_b = \cup\{I_a \mid a<b\}$. Assume therefore that $b=a+1$. It is true that $\mathrm{T}_{I_a}{}^{\uparrow 0}(\emptyset) \subseteq T$. Assume that $\mathrm{T}_{I_a}{}^{\uparrow a'}(\emptyset) \subseteq T$, we will show that $\mathrm{T}_{I_a}{}^{\uparrow a'+1}(\emptyset) \subseteq T$. Let $L \in \mathrm{T}_{I_a}{}^{\uparrow a'+1}(\emptyset)$. Then, $\exists\, r: L \leftarrow L_1,...,L_n$ in P s.t. r is reliable w.r.t. I_a and $\forall i \leq n$ either (i) $L_i \in I_a$ or (ii) L_i is a classical literal and $L_i \in \mathrm{T}_{I_a}{}^{\uparrow a'}(\emptyset)$. Since $I_a \subseteq M$, it follows

that r is reliable w.r.t. M. From the facts that M is an r-model of P, $I_a \subseteq M$, $T_{I_a}{}^{\uparrow a'}(\emptyset) \subseteq T$ and r is reliable w.r.t. M, it follows that $L \in T$. So, $T(I_a)=T_b \subseteq T$.

Now, we will show that $F_b \subseteq F$. Since $F_b = \neg T_b \cup F(I_a)$, it is enough to show that $\neg T_b \subseteq F$ and $F(I_a) \subseteq F$. If $L \in \neg T_b$ then $\neg L \in M$ and from Step (ii) of Def. 2.3.4, $L \in F$. Consequently, $\neg T_b \subseteq F$. For all rules $H \leftarrow L'_1,...,L'_m,\sim L_1,...,\sim L_n$ in P (L_i, L'_i are classical literals) with $H \in F(I_a)$ either $\exists i \leq m$, $L'_i \in F(I_a) \cup F_a$ or $\exists j \leq n$, $L_j \in T_a$. This implies that, for each rule $r: H \leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in P with $H \in F(I_a)$ either there is a corresponding rule $H \leftarrow A_1,...,A_k$ in $P|_r M$ (from Steps (iv) and (v) of Def. 2.3.4) with $A_i \in F(I_a) \cup F$ for an $i \leq k$ or there is no corresponding rule in $P|_r M$ (from Steps (i) and (ii) of Def. 2.3.4). Note that, r is not transformed into $H \leftarrow u$ in $P|_r M$ in Step (iii) of Def. 2.3.4 because the facts $I_a \subseteq M$ and $least_v(P|_r M)=M$ imply that $\exists i \leq m$, $L'_i \notin T$ or $\exists j \leq n$, $L_j \notin F$. Moreover, no rule $H \leftarrow u$ with $H \in F(I_a)$ is added to $P|_r M$ in Step (vi) of Def. 2.3.4 because the facts $\sim H$ is reliable w.r.t. I_a and $I_a \subseteq M$ imply that $\sim H$ is reliable w.r.t. M. So, for each rule $H \leftarrow A_1,...,A_k$ in $P|_r M$ with $H \in F(I_a) \cup F$, $\exists i \leq k$ s.t. $A_i \in F(I_a) \cup F$. From the definition of $least_v(P|_r M)$, it follows that $L \in F$. So, $F_b \subseteq F$ and thus $T_d \subseteq T$ and $F_d \subseteq F$. Consequently, $RM=T_d \cup \sim F_d \subseteq T \cup \sim F=M$. ◊

Proposition 2.3.6: Let $P=<R_P, IC_P, <_R>$ be an *EPP* and $<_R'$ be an extension of $<_R$. Every stable r-model of $P'=<R_P, IC_P, <_R'>$ is a stable r-model of P and $RM_P \subseteq RM_{P'}$.

Proof: Let M be a stable r-model of P'. We will show that every default literal which is unreliable w.r.t. M in P' is also unreliable w.r.t. M in P. Assume that $\sim L$ is unreliable w.r.t. M in P'. Since $<_R'$ is an extension of $<_R$, the set of rules with priority no lower than r in P' is a subset of that in P. So, Pos_M and $Dep_M(K)$, for a literal K, in P' are subsets of the corresponding sets in P and consequently $\sim L$ is unreliable w.r.t. M in P.

Let $S=\{L \mid M(L)=1/2$ and $\sim L$ is unreliable w.r.t. $M\}$. For all $L \in S$, $least_v(P'/M)(L)=1/2$ because $least_v(P'/M)=M$. This and the fact $P|_r M = P'|_r M \cup \{L \leftarrow u \mid L \in S\}$ imply that $least_v(P/M)= least_v(P'/M)=M$. From Proposition 2.2.2, M is an r-model of P. So, M is a stable r-model of P. Since RM_P is the least stable r-model of P, $RM_P \subseteq RM_{P'}$. ◊

2.3.3 Diagnosis Example

We will consider an application of *RS* to diagnosis and we will show how prioritized defaults can be used to express the relative reliability of circuit components.

Example 2.3.3:

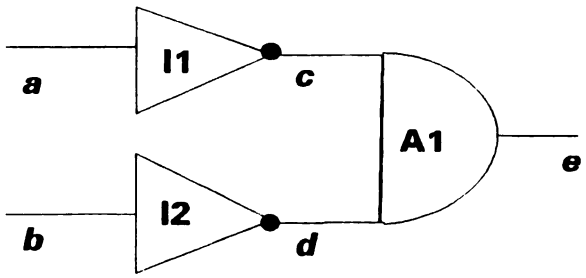

Figure 2.1: A digital circuit

The circuit of Figure 2.1 consists of two inverters and one AND gate. To reason about its behavior, we give a simple formulation with an *EPP*, $P=<R_P,IC_P,<_R>$:

R_P={ /* description of I1 gate */

r_1: $\neg c \leftarrow a$, OK_I1. r_2: $c \leftarrow \neg a$, OK_I1.

/* description of I2 gate */

r_3: $\neg d \leftarrow b$, OK_I2. r_4: $d \leftarrow \neg b$, OK_I2.

/* description of A1 gate */

r_5: $e \leftarrow c,d$, OK_A1. r_6: $\neg e \leftarrow \neg c$, OK_A1. r_7: $\neg e \leftarrow \neg d$, OK_A1.

r_8: a. /* a input has value 1 */ r_9: $\neg b$. /* b input has value 0 */

r_{10}: e. /* e output has value 1 */

/* assumptions that gates are working correctly*/

r_{11}: OK_I1. r_{12}: OK_I2. r_{13}: OK_A1.

$S_{ri}=Body_{ri}$, $\forall i \leq 13$},

$IC_P=BC_P$, and $<_R$ indicates that any rule r_i, i=1,...,10 has higher priority than any rule r_j, j=11,12,13.

Note that every classical model of R_P violates the constraint $\bot \leftarrow e, \neg e$. Apparently, the above circuit is faulty. Though there is no evidence that I2 does not work correctly, one of the gates I1 and A1 should be faulty. All rules r_i, $i \leq 10$, are reliable w.r.t. Ø because they have higher priority than rules r_{11}, r_{12} and r_{13} and do not generate any constraint violation. Rule r_{12} also is reliable w.r.t. Ø because OK_I2 does not belong neither to $\text{Dep}_{r12,\emptyset}(e)$ nor to $\text{Dep}_{r12,\emptyset}(\neg e)$. Rule r_{11} is unreliable w.r.t. Ø because $e \in \text{Pos}_{r11,\emptyset}$ and $\text{OK\_I1} \in \text{Dep}_{r11,\emptyset}(\neg e)$. Similarly, rule r_{13} is unreliable w.r.t. Ø. The reliable model of P is $RM_P = coh(\{a, \neg b, d, e, \text{OK\_I2}\})$. The truth values of OK_I1, OK_A1 are unknown because rules r_{11} and r_{13} are unreliable. The truth values of c and $\neg c$ are unknown because the truth value of OK_I1 is unknown. The stable r-models of P are:
$M_1 = coh(\{a, \neg b, d, e, \text{OK\_I2}, \text{OK\_A1}\})$ and $M_2 = coh(\{a, \neg b, \neg c, d, e, \text{OK\_I1}, \text{OK\_I2}\})$.
If we extend $<_R$ with $r_{11} < r_{13}$, indicating that gate A1 is more reliable than gate I1 then the unique stable r-model of the new program equals M_1.

Let P' be as P with $S_r = \{\}$ $\forall$ rule r. All rules in P' except r_7 and r_{10} are reliable w.r.t. Ø. Consequently, $RM_{P'} = \{a, \neg b, \neg c, d, \text{OK\_I1}, \text{OK\_I2}, \text{OK\_A1}\}$ indicating that all gates are working correctly but the truth value of output e is unknown. The stable r-models of P' are:
$M'_1 = coh(\{a, \neg b, \neg c, d, e, \text{OK\_I1}, \text{OK\_I2}, \text{OK\_A1}\})$ and
$M'_2 = coh(\{a, \neg b, \neg c, d, \neg e, \text{OK\_I1}, \text{OK\_I2}, \text{OK\_A1}\})$.
Model M'_1 indicates that output e has value 1 and that all gates are working correctly. This is an unintuitive result because if all gates are working correctly then output e should have value 0. Model M'_2 indicates that output e has value 0. This is also an unintuitive result because rule r_{10}, which expresses that the observed value of e is 1, has higher priority than rules r_{11}, r_{12} and r_{13}. The same reliable model and stable r-models are derived when P' is extended with $r_{11} < r_{13}$. The reason for these unintuitive results is that $S_r = \{\}$ $\forall$ rule r in P', even though the rules r_i are complete, for all $i \leq 10$. When a rule r_i, for $i \leq 10$, is in conflict with an observed output, the truth value of any literal in $Body_{r_i}$ may be mistaken. For this, S_{r_i} should be equal to $Body_{r_i}$, for all $i \leq 10$.

2.4 Related Work

In this Section, we review earlier work on semantics of subclasses of *EPPs* and give the relationship of these semantics with the reliable semantics.

2.4.1 Three-Valued Stable Model Semantics

Przymusinski [60] defines the 3-*valued stable model semantics* for normal programs by extending the *P/I* transformation [26] to 3-valued interpretations *I*. Specifically, if *P* is a normal program and *I* is a 3-valued interpretation of it then the non-negative program *P/I* is defined as follows:

(i) Remove from *P* all rules that contain in their body a default literal $\sim L$ s.t. $I(L)=1$.

(ii) Remove from the body of the remaining rules of *P* any default literal $\sim L$ s.t. $I(L)=0$.

(iii) Replace all remaining default literals $\sim L$ with $\boldsymbol{u}$.

A 3-*valued stable model* of *P* is a 3-valued interpretation *M* of *P* that satisfies $least_{\mathbf{v}}(P/M)=M$. The 3-*valued stable model semantics* of *P* is defined as the intersection of all the 3-valued stable models of *P*. As the next proposition shows, the reliable semantics is a generalization of the 3-valued stable model semantics.

Proposition 2.4.1: Let *P* be a normal program. Then, *M* is a 3-valued stable model of *P* iff $M\cup\{\sim\neg A \mid A$ is an atom of $P\}$ is a stable *r*-model of *P*.

Proof: When *P* is a normal program, *M* is a 3-valued stable model of *P* iff $M\cup\{\sim\neg A \mid A$ is an atom of $P\}$ is an extended stable model of *P* [54]. Proposition 2.4.1 now follows from Proposition 2.4.3. ◊

Przymusinski has shown that the intersection of all 3-valued stable models of a normal program *P* coincides with the well-founded model of *P* [76]. Consequently, Proposition 2.4.1 implies that the reliable model of a normal program coincides with its well-founded model.

2.4.2 Answer Set Semantics

Gelfond and Lifschitz [27] define the *answer set semantics* for extended programs by extending the *P/I* transformation [26] to extended programs *P*. A 2-valued interpretation is a set of classical literals. A classical literal *L* is true (resp. false) w.r.t. a 2-valued interpretation *I* iff $L \in I$ (resp. $L \notin I$). Specifically, if *P* is an extended program and *I* is a 2-valued interpretation then the program *P/I* is defined as follows:

(i) Remove from *P* all rules that contain in their body a default literal $\sim L$ s.t. $I(L)=1$.

(ii) Remove from the body of the remaining rules of *P* any default literal $\sim L$ s.t. $I(L)=0$.

Let *P* be a positive program. If $least_v(P)$ contains a pair of complementary literals then $a(P)=_{def} HB_P$. Otherwise, $a(P)=_{def} least_v(P)$. An *answer set* of an extended program *P* is a 2-valued interpretation *M* that satisfies $a(P/M)=M$. The *answer-set semantics* of an extended program is defined as the intersection of its answer-sets. When *P* is a normal program, the answer set semantics of *P* coincides with the stable model semantics of *P* [26]. However, similarly to the stable model semantics, the answer set semantics is not defined for all programs. There are several arguments for and against this. Some researchers would like to dismiss logic programs without any answer sets as not good (analogous to "inconsistent theories" in case of first order theories) theories while others would like to have a semantics that characterizes all logic programs. We believe that the second is the correct approach since the knowledge base may contain information that should be salvaged. For example, consider the program $P=\{p \leftarrow \sim p.\ \ a.\}$. *P* has no answer set even though *a* should be true. The reliable semantics of *P* is $\{a, \sim\neg a, \sim\neg p\}$.

An extended program *P* can have an answer set HB_P. In this case, *P* is called *contradictory* [27] and HB_P is the unique answer set of *P*. For example, when WFM_P is contradictory, the answer set semantics of *P* is HB_P. Thus, when *P* is contradictory, all information in *P* is lost. For example, the unique answer set of $P=\{p.\ \ \neg p.\ \ a.\}$ is HB_P. The problem of finding whether an extended program has an answer set is *NP*-complete [22].

The following relationship between the answer-set semantics and *RS* can be shown.

Proposition 2.4.2: Let P be an extended program. If $M \neq HB_P$ is an answer-set of P then $M \cup \{\sim A \mid A \notin M\}$ is a stable r-model of P.

Proof: P is non-contradictory since $M \neq HB_P$ is an answer-set of P. So, if M is an answer-set of P then $M \cup \{\sim A \mid A \notin M\}$ is an extended stable model of P [54]. Proposition 2.4.2 now follows from Proposition 2.4.3. ◊

The well-founded semantics and the reliable semantics are sometimes over-skeptical but in return they offer a constructive definition and efficiency. This will be clearer with the following example. Let $P=\{p \leftarrow \sim q. \quad q \leftarrow \sim p. \quad a \leftarrow q. \quad a \leftarrow p.\}$. Then, the answer sets of P are $\{p,a\}$, $\{q,a\}$ whereas the reliable semantics of P is $\{\sim\neg a, \sim\neg p, \sim\neg q\}$. Thus, according to reliable semantics of P, a is undefined even though a is true in all answer sets of P. So, the answer set semantics based on the intersection of the answer sets infers a even though it remains undecided about p and q.

2.4.3 Extended Well-Founded Semantics

Let P be an extended program and I an interpretation of it. In [54], the operator Φ_P is defined as $\Phi_P(I)=coh(least_v(P/I))$ if $least_v(P/I)$ does not contain a pair of complementary literals. Otherwise, $\Phi_P(I)$ is not defined. The *extended well-founded model* ($XWFM_P$) of P is defined in [54] as the least fixpoint of Φ_P. An *extended stable model* of P is a fixpoint of Φ_P. Let $I_0=\{\}$, $I_{a+1}=\Phi_P(I_a)$ and $I_a= \cup\{I_b \mid b<a\}$ if a is a limit ordinal. Then, $XWFM_P=I_d$ where d is the smallest ordinal s.t. $I_{d+1}=I_d$. When there exists an a s.t. $\Phi_P(I_a)$ is not defined, P is called *contradictory*.

Proposition 2.4.3: Let P be a non-contradictory extended program. Then, M is an extended stable model of P iff M is a stable r-model of P.

Proof: Let M be an extended stable model of P. From the definition of extended stable model [54], M is an r-model of P and $least_v(P/M)=M$. So, M is a stable r-model of P.

Let M be a stable r-model of P. Since P is a non-contradictory extended program, there is no L s.t. $L \in Pos_\emptyset$ and $\neg L \in Pos_\emptyset$ and $XWFM_P=Pos_\emptyset$. So, all default literals and rules in P are reliable

w.r.t. $I=\emptyset$. This implies that default literals and rules in P are reliable w.r.t. M. Consequently, Steps (iii) and (vi) of Definition 2.3.4 have no effect on $P/_rM$. So, from the definition of extended stable model, M is an extended stable model of P. ◊

Proposition 2.4.3 implies that if P is a non-contradictory extended program then the reliable model of P coincides with the extended well-founded model of P. When P is contradictory, the extended well-founded semantics of P is not defined in contrast to the reliable semantics. For example, consider the extended program $P=\{\neg p.\quad p.\quad b.\}$. P has three stable r-models $\{b, \sim\neg b\}$, $\{\neg p, b, \sim p, \sim\neg b\}$ and $\{p, b, \sim\neg p, \sim\neg b\}$ but no extended well-founded semantics.

2.4.4 Relevant Expansion

In [79], a program P with constraints is called *revisable* if it has a Δ-model, that is, if there is a consistent interpretation M s.t. for every rule r in P, $M(Head_r)\geq M(Body_r)$. Witteveen [79] shows that every revisable program P whose well-founded model [76] is inconsistent, can be expanded into a new program P' (*relevant expansion*) that has consistent well-founded model. The *foundation* of a literal L, $F(L)$, is defined as follows:

A classical literal $L'\in F(L)$ iff ∃ rule r s.t. all literals in $Body_r$ are true w.r.t WFM_P and

(i) $\sim L'\in Body_r$ or (ii) $\sim L'\in F(K)$ for a classical literal $K\in Body_r$.

The *relevant expansion* of P is the program $P'=P\cup\{L\leftarrow \sim L \mid L\in F(\bot)\}$ which has consistent well-founded model. The semantics of P is defined as the well-founded model of P'. However, the well-founded model of P' may not be a coherent interpretation of P. Moreover, when P is not revisable, P is not given any semantics in [79]. For example, the program P of Example 2.2.1 with $<_R=\{\}$ is not revisable but it has three stable r-models.

2.4.5 Conservative Vivid Logic

The *conservative reasoning* for extended programs, presented by Wagner [77], is as follows:

- A literal L is true iff (i) $\exists$ rule r s.t. $Head_r=L$ and $Body_r$ is true and (ii) $\forall$ rule r' s.t. $Head_{r'}=\neg L$, $Body_{r'}$ is false.
- A literal L is false iff (i) $\forall$ rule r with $Head_r=L$, $Body_r$ is false or (ii) $\exists$ rule r s.t. $Head_{r'}=\neg L$ and $Body_r$ is true.

In [72], it is shown that an extended program P can be translated into a normal program P_{cr} s.t. the three-valued completion of P_{cr} is sound and complete w.r.t. the conservative reasoning. The program P_{cr} is obtained as follows:

- For every rule $L \leftarrow Body_r$ (resp. $\neg L \leftarrow Body_r$) in P, where L is a positive classical literal, P_{cr} contains the clause $L^+ \leftarrow Body_r$ (resp. $L^- \leftarrow Body_r$).
- For every classical literal L, P_{cr} contains the clauses $L \leftarrow L^+, \sim L^-$ and $\neg L \leftarrow L^-, \sim L^+$.

For example, if $P=\{p.\ \neg p.\ a \leftarrow \sim p.\}$ then

$P_{cr}=\{p^+.\quad p^-.\quad a^+ \leftarrow \sim p.\quad a \leftarrow a^+, \sim a^-.\quad \neg a \leftarrow a^-, \sim a^+.\quad p \leftarrow p^+, \sim p^-.\quad \neg p \leftarrow p^-, \sim p^+.\}$.

and the *CVL* semantics is $M=\{a, \sim\neg a\}$. Note that neither p nor $\neg p$ is evaluated as true because of the rules $\neg p \leftarrow$ and $p \leftarrow$, respectively. However, $\sim p$ is evaluated as true because of the rule $\neg p \leftarrow$ even though there is the rule $p \leftarrow$ in P. Consequently, a^+ and a are evaluated as true. In contrast, in the reliable semantics of P, the truth values of $\sim p$ and a are unknown. We think that this is a more intuitive result because of the rule $p \leftarrow$ in P.

The *CVL* semantics of an extended program P can be a strict subset of the WFM_P even when WFM_P is consistent and the intuitively correct semantics. For example, if $P = \{\neg p \leftarrow \sim p.\quad p.\}$ then $P_{cr}=\{p^- \leftarrow \sim p.\quad p^+.\quad p \leftarrow p^+, \sim p^-.\quad \neg p \leftarrow p^-, \sim p^+.\}$.

Note that P is consistent and $WFM_P=\{p, \sim\neg p\}$ which is the intuitively correct semantics. Yet, the three-valued completion of P_{cr} and thus, the *CVL* semantics of P does not imply neither p nor $\sim\neg p$.

2.4.6 Predicate Logic Extensions

Prioritization of defaults is supported in [11, 12] and in [64]. In all of these works, a default is a formula containing only the classical connectives $\neg$ and $\leftarrow$. When P has a classical model, the semantics of P in [11, 12, 64] coincide with the predicate logic semantics of P. In [11, 12, 64],

rules are considered to be clauses, i.e., there is no distinction between the head and the body of a rule. For example, the program $P = \{p \leftarrow \neg p.\}$ is considered equivalent with $\{p.\}$ and the semantics of P is $\{p\}$. In contrast, the *WFM* of $P' = \{p \leftarrow \sim p.\}$ is $\{\}$. In all of these works the semantics of P is defined as follows:

- Let M, N be two classical interpretations and $D(M)$, $D(N)$ the set of defaults in P that are classically satisfied by M, N, respectively. Then, $M < N$ iff $D(M) \neq D(N)$ and $\forall r \in (D(N)-D(M))$, $\exists r' \in (D(M)-D(N))$ s.t. $r < r'$.
- An interpretation M is called *intended* iff there is no other interpretation N s.t. $N < M$.
- The semantics of P is defined as the intersection of all intended models of P.

The *most reliable consistent set of premisses* D when rules are totally ordered $r_1 < ... < r_n$ is defined as follows: $D_0 = \emptyset$ and $\forall 0 < i \leq n$, if $D_i \cup \{r_i\}$ is consistent then $D_{i+1} = D_i \cup \{r_i\}$ else $D_{i+1} = D_i$. Roos [64] shows that the proposed semantics coincides with the intersection of all classical models of the most reliable consistent set of premisses for all linear extensions of $<_R$. However, the number of linear extensions of $<_R$ can be exponentially large. For example, the number of linear extensions of $<_R = \{\}$ is $n!$, where n is the number of defaults.

The intended models of program P in Example 2.3.3 are: $M_1 = coh(\{a, \neg b, c, d, e, \neg$OK_I1, OK_I2, OK_A1$\})$ and $M_2 = coh(\{a, \neg b, \neg c, d, e,$ OK_I1, OK_I2, $\neg$OK_A1$\})$. Thus, the semantics of P according to [11, 12] and [64] agree with the *RS* of P.

2.4.7 Ordered Logic

Prioritization of rules is also investigated in [24, 43]. An *ordered logic program* is a partially-ordered set of rules without negation by default. Even though the *c-assumption-free semantics* [24] and *assumption-free semantics* [43] are defined for all ordered logic programs, negation by default is not supported and only the basic constraints are considered. The *skeptical c-partial model* of an ordered logic program P is defined in [24] as follows:

• A

the

Ø

• A

∩

• Th

r i

(a.

rules

outp

repr

that

the c

The

with

Prop

rule

class

Proo

rule

equiv

classi

true fo

$\hat{S}^{\alpha+1}$(

enough

is not c

- A literal set S is *c-unfounded* w.r.t. an interpretation I iff $\forall L \in S$, if r is a rule in P with $Head_r = L$ then either (i) $\exists$ rule r' s.t. $r' \not< r$, $Head_{r'} = \neg Head_r$ and $Body_{r'} \cup Head_{r'} \subseteq I$ or (ii) $Body_r \cap S \neq \emptyset$.
- A rule r is *c-defeasible* w.r.t. I iff $\exists$ rule r' s.t. $r' \not< r$, $Head_{r'} = \neg Head_r$ and $(Body_{r'} \cup Head_{r'}) \cap U^c(I) = \emptyset$, where $U^c(I)$ is the greatest c-unfounded set w.r.t. I.
- The *sceptical c-partial model* of P is the least fixpoint of the monotonic operator $S(I)=\{L \mid \exists$ rule r in P s.t. $Head_r=L$, $Body_r \subseteq I$ and r is not c-defeasible w.r.t. $I\}$.

The *skeptical c-partial model* of the program P in Example 2.3.3 is $\{a, \neg b, \neg c, d,$ OK_I1, OK_I2, OK_A1$\}$. This is because rules r_7 and r_{10} are the only c-defeasible rules w.r.t. $\emptyset$ in P. According this model, all gates are working correctly but the truth value of the output e is unknown. This unintuitive result is derived because in [24], the rule ordering $r' < r$ represents that rule r is an *exception* of rule r'. This corresponds in our framework with the case that $S_r=\{\}$ $\forall$ rule r. In Example 2.3.3, we showed that if $S_r=\{\}$ $\forall$ rule r then rules r_7 and r_{10} are the only unreliable rules w.r.t. $\emptyset$ and the reliable model equals the skeptical c-partial model of P. The next proposition shows that the reliable model of every ordered logic program, P, coincides with the skeptical c-partial model of P.

Proposition 2.4.4: Let $P=\langle R_P, IC_P, <_R \rangle$ be an *EPP* which is free from default literals, $S_r=\{\}$ $\forall$ rule r, and $IC_P = BC_P$. Then, M is the skeptical c-partial model of P [24] iff M is the set of classical literals in RM_P.

Proof: To simplify the proof, we redefine the operator $T(J)$ of Def. 2.3.2 as follows: $T(J)=\{L \mid \exists$ rule r in P s.t. $Head_r=L$, $Body_r \subseteq J$ and r is reliable w.r.t. $J\}$. Note that both definitions give equivalent reliable semantics. Let $I_a = W_P{}^{\uparrow a}(\emptyset)$, for all a. We will show by induction that the set of classical literals in I_a coincides with $S^{\uparrow a}(\emptyset)$, for all a. This is true when $a=0$. Suppose that it is true for all ordinals $\leq a$. We will show that the set of classical literals in I_{a+1} coincides with $S^{\uparrow a+1}(\emptyset)$. Since $S(I)=\{L \mid \exists$ rule r s.t. $Head_r=L$, $Body_r \subseteq I$ and r is not c-defeasible w.r.t. $I\}$, it is enough to show that for each rule r, $Body_r \subseteq I_a$ and r is reliable w.r.t. I_a iff $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and r is not c-defeasible w.r.t. $S^{\uparrow a}(\emptyset)$.

$Body_r \subseteq I_a$ and r is reliable w.r.t. I_a

(From the inductive hypothesis and the fact that $Body_r$ is free of default literals, it follows that $Body_r \subseteq S^{\uparrow a}(\emptyset)$.)

iff $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and rule r is reliable w.r.t. I_a

(From the fact $S_r = \{\}$ $\forall$ rule r and the reliable rule definition, it follows that r is reliable w.r.t. I_a iff (i) $\exists$ no rule $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $Body_{r'} \subseteq Pos_{I_a}$ or (ii) $Body_r$ is not a subset of Pos_{I_a}. Note that $Body_r \subseteq S^{\uparrow a}(\emptyset) \subseteq I_a \subseteq Pos_{I_a}$.)

iff $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and $\exists$ no rule $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $Body_{r'} \cup Head_{r'} \subseteq Pos_{I_a}$

iff $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and $\exists$ no $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $(Body_{r'} \cup Head_{r'}) \cap (HB_P - Pos_{I_a}) = \emptyset$

(Let r' be a rule in P with $Body_{r'} \subseteq Pos_{I_a}$. Then, r' is blocked w.r.t. I_a iff $\neg Head_{r'} \in I_a$ iff $\exists$ $r'' \not< r'$ with $Head_{r''} = \neg Head_{r'}$ and $Body_{r''} \cup Head_{r''} \subseteq S^{\uparrow a}(\emptyset)$. Consequently, $U^c(S^{\uparrow a}(\emptyset)) = HB_P - Pos_{I_a}$.)

iff $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and $\exists$ no $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $(Body_{r'} \cup Head_{r'}) \cap U^c(S^{\uparrow a}(\emptyset)) = \emptyset$

iff $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and r is not c-defeasible w.r.t. $S^{\uparrow a}(\emptyset)$. $\Diamond$

In [43], the *well-founded partial model* of an ordered logic program P is defined as follows:

- A literal set S is *unfounded* w.r.t. an interpretation I iff $\forall L \in S$, if r is a rule in P with $Head_r = L$ then either (i) $\exists$ rule r' s.t. $r' \not< r$, $Head_{r'} = \neg Head_r$ and $Body_{r'} \subseteq I$ or (ii) $Body_r \cap S \neq \emptyset$.
- A rule r is *defeasible* w.r.t. I iff $\exists$ rule r' s.t. $r' \not< r$, $Head_{r'} = \neg Head_r$ and $Body_{r'} \cap U(I) = \emptyset$, where $U(I)$ is the greatest unfounded set w.r.t. I.
- The *well-founded partial model* of P is the least fixpoint of the monotonic operator $S(I) = \{L \mid \exists$ rule r in P s.t. $Head_r = L$, $Body_r \subseteq I$ and r is not defeasible w.r.t. $I\}$.

Similarly to [24], rule ordering in [43] represents *exceptions* and not *reliability*. This corresponds in our framework with the case that $S_r = \{\}$ $\forall$ rule r. Indeed, the reliable model of the program P in Example 2.3.3 with $S_r = \{\}$ $\forall$ rule r, is the same as the well-founded partial model of P. Another difference between the reliable model and the well-founded partial model is

demonstrated by the following example: The well-founded partial model of $P=\{p.\ \neg p\leftarrow q.\ \neg q.\ q.\}$ with $<_R=\{\}$ is $\{p\}$. According to this model, p is true even though $\neg p$ can also be derived from P. This is because rule $p\leftarrow$ is not considered defeasible. According to [43], the literal q is ambiguous and thus the derivation of q and $\neg p$ is blocked. In [70], a similar ambiguity blocking approach applied to inheritance networks was severely questioned. In our approach, ambiguities are propagated and thus, rule $p\leftarrow$ is considered unreliable. Note that $RM_P=\{\}$, independently of the values of S_r.

Proposition 2.4.5 shows that the reliable semantics is more skeptical than the assumption-free semantics of [43]. The proposition follows immediately from Proposition 2.4.4 and the fact that the skeptical c-partial model of an ordered logic program P is a subset of the well-founded partial model of P [Theorem 8, 24].

Proposition 2.4.5: Let $P=<R_P, IC_P, <_R>$ be an *EPP* which is free from default literals, $S_r=\{\}\ \forall$ rule r, and $IC_P=BC_P$. Then, the set of classical literals in RM_P is a subset of the well-founded partial.model of P [43].

The rule ordering $<_R$ in *RS* expresses that in case of conflict, one rule is considered more reliable than another. Saying that *r is more reliable than r'* is different than saying that *r is an exception to r'*. Let r: $L\leftarrow L_1,...,L_n$ and r': $L'\leftarrow L'_1,...,L'_m$ be two rules. The fact that r is an exception of r' can be expressed by replacing the old rule r' with r': $L'\leftarrow L'_1,...,L'_n,\sim name_r$ and by adding the rule: $name_r\leftarrow L_1,...,L_m$ [53, 56]. For example, let r: $\neg flies(X)\leftarrow penguin(X)$ and r': $flies(X)\leftarrow bird(X)$. The fact that r is an exception of r' is represented by replacing the old rule r' with r': $flies(X)\leftarrow bird(X),\sim nf(X)$ and adding the rule $nf(X)\leftarrow penguin(X)$. The relation $<_R$ is extended as follows: The added rule has lower (resp. higher) priority than a rule r'' iff $r<r''$ (resp. $r''<r$).

2.4.8 Default Logic

According to default logic, when only partial information is available, the program can be augmented with a set of *default* rules. Conclusions drawn with the aid of these defaults are not certain and they can be defeated by the acquisition of more information.

In Reiter's *default logic* [63], defaults are represented as inference rules which have a consistency-check side condition. More specifically, a default rule is an expression of the form:

$$\frac{a : b1,...bn}{c}$$

where $a, b1,...,bn$ and c are *well-formed formulas* (*wffs*). The formula a is called *precondition*, the formulas $b1,..., bn$ are called *justifications* and they are checked for consistency with the database and the formula c is called *consequent* of the default rule. For example, one would encode the default "Birds can fly" in default logic as:

$$\frac{b(x):f(x)}{f(x)}$$

which is read as: if x is a bird and it is consistent to conclude that x can fly, then x can fly.

A default theory in Reiter's formalism is a set of sentences W with a set of default rules D. An *extension* of this default theory is a logical theory T such that:

1. None of the rules can be consistently applied to obtain a conclusion not already in the extension.
2. Subject to this condition, the extension is minimal.

Consider Γ as an operator on a logical theory T, returning a new logical theory $\Gamma(T)$ which is the result of applying a default rule in D to T. Then, $T \subseteq \Gamma(T)$. An extension E is a least fixed point of the operator Γ. A default theory may have more than one extensions. In default logic, exceptions should be coded up in the rules. For example, consider the well-known *inheritance with exceptions* example: the class of penguins is a subclass of the class of birds but the property of "being able to fly," which holds by default for birds, is not inherited by the penguins. The knowledge base

contains (i) the premises: a is a penguin, penguins are birds, and (ii) the defaults: birds usually can fly, penguins usually cannot fly. The problem is: "Does a fly?"

One may consider coding this in default logic, as follows:

(X is a variable, p stands for penguin, b for bird, f for fly)

$p(a)$. $\quad b(X) \leftarrow p(X)$. $\quad \dfrac{b(X):f(X)}{f(X)} \quad \dfrac{p(X):\neg f(X)}{\neg f(X)}$

However, even though we expect that a does not fly, the above default theory has two extensions, one containing $f(a)$ (a flies) and another containing $\neg f(a)$. To obtain only one extension that contains $f(a)$, the first default should be stated as follows:

$$\frac{b(X):\neg p(x),f(X)}{f(X)}$$

Thus, the fact that penguins are exceptions to the default about birds should be explicitly indicated in the rule. It seems easier to represent exceptions with a hierarchy of defaults, where more specific defaults have higher priority. The previous example can be stated with an *EPP* as follows:

$P = \{ r_1: f(X) \leftarrow b(X). \quad r_2: \neg f(X) \leftarrow p(X). \quad r_3: b(X) \leftarrow p(X). \quad r_4: p(a).\}$

with $S_{r1} = S_{r2} = \{\}$, $S_{r3} = \{p(X)\}$ and $r_1 < r_2$, $r_2 < r_3$, $r_2 < r_4$. The reliable model of P contains $\neg f(a)$, as it was expected. Assume that later we find that an emu is a bird that generally does not fly. We do not need to modify any rule in P but only add the rules $\{r_5: \neg f(X) \leftarrow e(X). \quad r_6: b(X) \leftarrow e(X).\}$ with $S_{r5} = \{\}$, $S_{r6} = \{e(X)\}$ and the priorities $\{r_1 < r_5 < r_6\}$ to it.

A default d in default logic can have a much more powerful representation than a rule in an extended logic program, since the prerequisite, justifications and consequent of d can be general wffs. However, a relationship between the *RS* and the extensions of a special subclass of default theories can be derived using a result in [27]. Let P be an extended program and T_P be the default theory that contains the default:

$$\frac{L_1,\ldots,L_m:\neg L_{m+1},\ldots,\neg L_n}{L_0}$$

for any rule $L_0 \leftarrow L_1,...,L_m,\sim L_{m+1},...,\sim L_n$ in P. Gelfond and Lifschitz [27] have shown that the deductive closure of an answer set of an extended program P is an extension of the default theory T_P and vice versa. The next proposition follows from this result and proposition 2.4.2.

Proposition 2.4.6: Let P be an extended logic program. Then, any extension of the default theory T_P is the deductive closure of a stable r-model of P.

Similarly to the answer set semantics [27], there are default theories which do not have extensions. For example, consider the default theory T_P:

$$\frac{:}{p} \qquad \frac{:\neg q}{q}$$

The corresponding extended logic program is: $P=\{r_1: p\leftarrow. \quad r_2: q\leftarrow\sim q.\}$. The reliable model of P is $RM_P=\{p, \sim\neg p, \sim\neg q\}$. The reverse of Proposition 2.4.6 does not hold since RM_P is a stable model of P but T_P has no extension.

A default theory T may have an inconsistent extension in which case it is the only extension of T. For example, the default theory

$$\frac{:}{p} \qquad \frac{:}{\neg p} \qquad \frac{:}{q},$$

has an inconsistent extension.

The circuit of Example 2.3.3 can be formulated by a default theory (D, W), where W contains the rules r_i, $i=1,...,19$ and D contains the default rules:

$$\frac{:OK\_I1}{OK\_I1} \qquad \frac{:OK\_I2}{OK\_I2} \qquad \frac{:OK\_A1}{OK\_A1}$$

The extensions of the above default theory are the deductive closures of the models: $M_1=\{a,\neg b,c,d,e,\neg$OK_I1, OK_I2,OK_A1$\}$ and $M_2=\{a,\neg b,\neg c,d,e,$OK_I1,OK_I2,$\neg$OK_A1$\}$.

Thus, the semantics default logic gives to P coincides with the deductive closure of RS of P.

Since we have only one-level assumptions, both RS and default logic work equally well. However, a hierarchy of assumptions can be easily handled by RS but not by default logic.

2.5 Conclusions

In this chapter, we presented the *reliable semantics* (*RS*) for extended programs with rule prioritization and integrity constraints. We gave both a fixpoint and model theoretic characterization of *RS* and proved that they are equivalent. *RS* is always defined and non-contradictory. The *RS* fixpoint operator avoids contradictions by taking the first of the approaches presented in Section 1.3. The operator takes into account the priorities of the rules. Rules in P are implicitly given higher priority than *CWAs*.

Every rule r has a corresponding set $S_r \subseteq Body_r$, called the *preliminary suspect set* of r. When a constraint $\bot \leftarrow L_1,...,L_k$ is violated, the rules used in the *last* step of the derivation of L_i are considered "suspect." If a rule r is "suspect" for a constraint violation then the rules and *CWAs* used in the last step of the derivation of literals in S_r are also "suspect." The values of S_r depend on the reasons for a constraint violation. The motivation behind the idea of the preliminary suspect sets is given in section 1.3. Criteria for defining the values of the preliminary suspect sets are given in section 2.2. Intuitively, a rule r is considered *reliable* if it is not "suspect" for any constraint violation caused by rules with priority no lower than r. A *CWA* is considered *reliable* if it is not "suspect" for any constraint violation caused by the well-founded model of P. Only reliable *CWAs* and literals derived from reliable rules are consequences of the fixpoint operator of *RS*.

The model theoretic characterization of the *RS* of P is given by defining the *stable r-models* of P. In subsection 2.3.2, we proved that *RS* is the least stable r-model of P. *RS* represents the skeptical "meaning" of P and thus none of its conclusions is based on unreliable rules or *CWAs*. The degree of "skepticism" in *RS* depends on the preliminary suspect sets of its rules. Credulous conclusions are obtained by isolating the conflicting results in the multiple stable r-models of P. Stable r-models of P represent possible "meanings" of P. For example, if $P=\{\neg p \leftarrow. \quad p \leftarrow.\}$ then P has two stable r-models $\{\neg p, \sim p\}$ and $\{p, \sim \neg p\}$ which correspond to the two possible meanings of P. The *RS* of P is $\{\}$ which is the intersection of the two stable r-models. In subsection 2.3.1, we proved that when the Herbrand base of an *EPP* is finite, the complexity of computing *RS* is polynomial w.r.t. the size of the program.

In section 2.4, we proved that *RS* extends the well-founded semantics for normal programs [76] and extended well-founded semantics [54] for non-contradictory extended programs [54]. *RS* generalizes the approaches taken in *ordered logic* [24, 43] and *conservative vivid logic* [77], in the sense that the latter approaches correspond to the case that $S_r=\{\}$ $\forall$ rule r. *RS* also generalizes the approach taken in *relevant expansion* [79], in the sense that the latter corresponds to the case that $S_r=Body_r$ $\forall$ rule r. In subsection 2.4.7, we proved that if P is an ordered logic program then the *RS* of P coincides with the *skeptical c-partial model* of P [24] and is a subset of the *well-founded partial model* of P [43]. We also showed that rule prioritization in *RS* expresses reliability whereas in ordered logic it represents exceptions. Ordered logic does not support negation by default and vivid logic does not support rule prioritization. The relevant expansion approach does not support rule prioritization and fails to give semantics to every contradictory extended program. Thus, *RS* provides a broader framework for unifying these semantics.

CHAPTER 3

MODULAR RELIABLE SEMANTICS FOR MODULAR LOGIC PROGRAMS WITH RESULT PRIORITIZATION

3.1 Introduction

Modules in a reasoning system arise as a result of a functional decomposition of a complex reasoning task into a number of simpler subtasks. Each module is an interactive reasoning subsystem that is used for the (often partial) *definition* of its exported predicates. In our framework, a knowledge base consists of a set of modules. Each module contains a set of rules viewed as an open logic theory [13] with a set of input literals. A module represents an incomplete specification of some domain because its input literals are defined in other modules. However, a module becomes a standard extended logic program (closed module) when the truth values of its input literals are known.

The prospect of contradiction is even stronger when information is distributed in a set of modules. The code of a module is usually hidden from other modules. Thus, modules export their results while they hide the way these results are computed. *Independent modules* export results obtained only by local information. Independent modules may represent independent data and knowledge bases. *Supervisory modules* may aggregate results of lower-level modules or resolve conflicts between lower-level modules. *Cooperating modules* exchange intermediate results towards a common goal. When exported results are in conflict, prioritization of results can express higher confidence in some results over others.

Several different proposals assign declarative semantics to modular logic programs including [48, 49, 14, 44, 68, 50, 9]. Yet, the problem of conflicting predicate definitions by the different modules is not investigated in the above works. In [68], the syntactical side of modularity is

modules is not investigated in the above works. In [68], the syntactical side of modularity is handled. In [14], an algebra of modules is proposed. In [48, 49, 44, 50, 9] dynamic compositions of modules are considered where the nesting of the modules indicates the direction in which information is exported/imported. For example, to prove goal G in query $[M_1][M_2]G$, module M_2 imports information exported by M_1 but not vice versa. In this paper, only statically configured modules are considered, that is, each module M imports information from other modules as specified in the definition of M.

A *prioritized modular logic program* (*PMP*) consists of a set of modules and a partial order $<_{def}$ on the predicate definitions (M, p), where M is a module and p is a predicate exported by M. We assume that modules are internally consistent. However, a *PMP* is possibly not globally consistent. When a conflict occurs, $<_{def}$ expresses our relative confidence in the predicate definitions contributing to the conflict. Each module has a set of *local* literals that are inaccessible to other modules. Literals that can be accessed (imported) by any module have the form $\{M_1,...,M_n\}$:A, $\{M_1,...,M_n\}$: $\neg A$ or $\{M_1,...,M_n\}$: $\sim A$, where M_i are module names and A is a conventional atom whose predicate is exported by all M_i. Intuitively, a literal $\{M_1,...,M_n\}$:A is evaluated as true if (i) A is derived from a module M_i and (ii) if $\neg A$ is derived from a module $M_j \neq M_i$ then result A has higher priority than result $\neg A$. A literal $\{M_1,...,M_n\}$: $\sim A$ is evaluated as true if $\{M_1,...,M_n\}$: $\neg A$ is true or $\sim A$ is true in all modules M_i.

We present a semantics for *PMPs*, called *modular reliable semantics* (*MRS*), which assigns a truth value *true*, *false* or *unknown* to every literal. Every *PMP* has at least one *stable m-model*. The *MRS* of a program P is the least fixpoint of a monotonic operator and the least stable *m*-model of P. When a *PMP* is contradictory, exported results (represented by indexed literals) are considered *unreliable* if: (i) they contribute to a contradiction, and (ii) they do not have higher priority than the other contributing results. The *MRS* of a *PMP*, P, represents the skeptical meaning of P and thus none of its conclusions is based on unreliable exported results. Credulous

conclusions are obtained by isolating the conflicting results in the multiple stable *m*-models of *P*. The complexity of computing *MRS* is polynomial w.r.t. the size of the program (when the Herbrand Base of *P* is finite).

A *prioritized extended logic program* (*PEP*) consists of a set of partially ordered rules. A *PEP* can be translated into a *PMP* by considering each rule as a module that imports all predicates appearing in its body and exports its head predicate. Consequently, the *MRS* for *PEPs* is also defined. *MRS* is defined for all *PEPs* and extends the well-founded semantics [76] and the extended well-founded semantics [54]. Under certain conditions, a *PMP* can be translated into a *PEP* with equivalent modular reliable semantics.

3.2 Informal Presentation and Intuitions

Our framework can be used for the representation of result-sharing cooperating agents [46]. A complex task is statically decomposed into a set of simpler subtasks, each assigned to an agent. Each of the subtasks may be decomposed into even simpler subtasks, each assigned to a lower-level agent. An agent consists of a theory and, implicitly, by an inference mechanism. The theory of an agent is encapsulated in a module whose name is used for the identification of the agent. Typically, the subtask of an agent can not be solved only with local information and agents import results exported from other agents. Agents may have overlapping or even identical capabilities. Therefore, it is possible that they export agreeing or contradictory results. When agents M_1, M_2 export contradictory conclusions about a literal L, the truth value of L w.r.t. M_1, M_2 (expressed by the truth value of the literal $\{M_1, M_2\}$:L) is unknown. Yet, agents M_1, M_2 maintain their individual beliefs about L which is expressed by the truth value of the literals $\{M_1\}$:L, $\{M_2\}$:L, respectively.

Example 3.2.1: Sensors S_1, S_2 are gathering information from two different angles about the persons entering a building. Modules M_1, M_2 are assigned with the identification of terrorists based on the information collected from sensors S_1, S_2, respectively. Each module M_1, M_2 exports the

result *entered*(*terrorist*) (resp. ¬*entered*(*terrorist*)) iff it reaches the conclusion that an (resp. no) terrorist has entered the building. It is possible that M_1, M_2 disagree, i.e., M_1 exports *entered*(*terrorist*) whereas M_2 exports ¬*entered*(*terrorist*). The results exported by M_1, M_2 can be queried by other modules in various ways. For example,

- $Query_1 \leftarrow \{M_1\}$:*entered*(*terrorist*), $\{M_2\}$:*entered*(*terrorist*).

  $Query_1$ is true if *entered*(*terrorist*) is true in both M_1, M_2. $Query_1$ is false by default if *entered*(*terrorist*) is false (by default or classically) in at least one of M_1, M_2.
- $Query_2 \leftarrow \{M_1\}$: ¬*entered*(*terrorist*).

  $Query_2$ is true if ¬*entered*(*terrorist*) is true in M_1 (even if *entered*(*terrorist*) is true in M_2).
- $Query_3 \leftarrow \{M_1, M_2\}$: *entered*(*terrorist*).

  $Query_3$ is true if *entered*(*terrorist*) is true in at least one of M_1, M_2 and M_1, M_2 do not disagree. $Query_3$ is false by default if *entered*(*terrorist*) is false in both M_1, M_2.
- $Query_4 \leftarrow \{M_1\}$: ~*entered*(*terrorist*).

  $Query_4$ is true if *entered*(*terrorist*) is false in M_1 (even if *entered*(*terrorist*) is true in M_2).
- $Query_5 \leftarrow \{M_1, M_2\}$: ~*entered*(*terrorist*).

  $Query_5$ is true if *entered*(*terrorist*) is false in both M_1, M_2.

Individual agent theories are assumed to be consistent. Yet, the consistency of the union of agent theories is not assured. As we saw in Example 3.2.1, one case of contradiction is when independent modules export contradictory results. In this case, the contradiction depends only on the independent modules and it is relatively easy to resolve. Yet, generally, contradictions may involve several module interactions. For example, an agent exports a faulty result, this result is imported by another agent which exports a faulty result based on the imported faulty result. After a few module interactions, contradiction may arise in two ways :

(i) Complementary literals are derived inside an module. For example, both q, $\neg q$ are derived inside module M_3 of program P. Note that the derivation of q, $\neg q$ is based on the imported results $\{M_1\}:p$, $\{M_2\}:r$, respectively.

program P

module M_1 **exports** p	**module** M_2 **exports** r	**module** M_3 **imports** p, r
rules p.	**rules** r.	**rules** $q \leftarrow \{M_1\}:p$. $\neg q \leftarrow \{M_2\}:r$.

(ii) Complementary literals are exported from two different modules. For example, in program P_2, module M_2 exports the result $\neg q$. Module M_3 exports q whose derivation is based on $\{M_1\}:r$.

program P_2

module M_1 **exports** r	**module** M_2 **exports** q	**module** M_3 **imports** r **exports** q
rules r.	**rules** $\neg q$.	**rules** $q \leftarrow \{M_1\}:r$.

When contradiction appears, the sources of the contradiction are traced back to the contributing exported results. Domain specific information might indicate that some exported results are more reliable (have higher priority) than others. Let res_1 and res_2 be two exported results contributing to the contradiction. If res_1 has higher priority than res_2 and no contradiction arises without res_2 then only res_1 is taken into account. If the priority of res_1, res_2 cannot be compared then both are eliminated from *MRS* (skeptical approach). For example, in the program P above, $\{M_1\}:p$ and $\{M_2\}:r$ are two exported results contributing to the derivation of q, $\neg q$ in module M_3. If the definition of p in M_1 has higher priority than that of r in M_2, i.e., $(M_2, r) <_{def} (M_1, p)$, then result $\{M_1\}:p$ is considered more reliable than $\{M_2\}:r$. In this case, $\{M_1\}:p$ is true and $\{M_2\}:r$ is unknown in all stable m-models of P. If none of $\{M_1\}:p$, $\{M_2\}:r$ is more reliable than the other then their truth value in *MRS* is unknown. Results $\{M_1\}:p$, $\{M_2\}:r$ are isolated in the stable m-models of P. In other words, there is a stable m-model which evaluates only $\{M_1\}:p$ as true and another which evaluates only $\{M_2\}:r$ as true.

3.3 *m*-models for Prioritized Modular Logic programs

Our alphabet contains a finite set of constant, predicate and variable symbols from which ordinary atoms are constructed in the usual way. It also contains a finite set of module names. An *indexed atom* has the form $\{M_1,...,M_n\}:A$, where A is an ordinary atom and M_i are module names. A *classical literal* is either an atom A or its classical negation $\neg A$. The classical negation of a literal L is denoted by $\neg L$, where $\neg(\neg L)=L$. A *default literal* is the default negation $\sim L$ of a classical literal L, where $\sim(\sim L)=L$. A literal is called *indexed* when its corresponding atom is indexed. We define $\{M_1,...,M_n\}:\neg A = \neg\{M_1,...,M_n\}:A$ and $\{M_1,...,M_n\}:\sim A = \sim\{M_1,...,M_n\}:A$, where M_i are module names and A is an ordinary atom. The classical literals L, $\neg L$ are called *complementary* to each other. The predicate of a literal L is denoted by $pred_L$.

A module with name M is a tuple $<Exp_M, Imp_M, R_M>$. The set Exp_M contains the predicates that are exported (defined) by M. The set Imp_M contains the predicates imported by M. The set R_M contains rules of the form: $L \leftarrow L_1,...,L_m,\sim L_{m+1},...,\sim L_n$, where L is a non-indexed classical literal and L_i are classical literals. If an indexed literal $\{M_1,...,M_n\}:L$ is in the body of a rule then M imports L from the modules $M_1,...,M_n$. If a non-indexed literal L is in the body of a rule and $pred_L \in Imp_M$ then M imports L from all modules that export $pred_L$.

A *prioritized modular logic program* (*PMP*), P, is a pair $<Mod_P, <_{def}>$. Mod_P is a set of modules and $<_{def}$ is a partial order on Def_P, where $Def_P = \{(M,p) \mid M \in Mod_P \text{ and } p \in Exp_M\}$. Each $(M,p) \in Def_P$ represents the *definition* of predicate p in module M. If $\{M_1,...,M_n\}:L$ is an indexed literal appearing in P then $(M_i,pred_L) \in Def_P$, $\forall i \leq n$.

Individual modules are assumed to be consistent but their union may be inconsistent. Thus, when complementary literals are derived within a module M, it is because of unreliable information imported by M. When literals L, $\neg L$ are derived from different modules M, M', it is because the definition of $pred_L$ in M, M' is unreliable or the information imported by M, M' is unreliable. When

conflict occurs, $<_{def}$ expresses our relative confidence in the predicate definitions contributing to the conflict. Let (M,p) and (M',p') be in Def_P. The notation $(M,p) <_{def} (M',p')$ (resp. $(M,p) \not<_{def} (M',p')$) means that the definition of p in M is less (resp. not less) trusted than that of p' in M'. Intuitively, a literal L exported by a module M is reliable if it does not contribute to any derivation of complementary literals caused by definitions $(M',p') \not<_{def} (M,pred_L)$. Note that, the reliability of L is not affected by predicate definitions less trusted than the definition of $pred_L$ in M.

We define $S_L=\{M \in Mod_P \mid pred_L \in Exp_M\}$, where L is a literal. The indexed literals $S_L:L$ are called *globally indexed*. To simplify the presentation of the semantics a *renaming* mechanism is employed. Let r be a rule in a module M. Then, the head L of r is replaced by a new literal $M\#L$. Every non-indexed literal L in the body of r with $pred_L \notin Imp_M$ is replaced by $M\#L$. Every non-indexed literal L in the body of r with $pred_L \in Imp_M$ is replaced by the globally indexed literal $S_L:L$. Literals $M\#L$ are called *local* to M and they are not accessed by other modules. In contrast to local literals, *indexed* literals are accessible to all modules. When we refer to a *PMP*, we assume that the above renaming has already been done. Note that after renaming, only local and indexed literals appear in a module M. We define $M\#\neg A = \neg M\#A$ and $M\#\sim A = \sim M\#A$, where M is a module name and A is an ordinary atom.

The *Herbrand Base* (HB_P) of a *PMP*, P, consists of (i) the instantiations of the classical literals appearing in P, (ii) the globally indexed classical literals $S_L:L$, where L is the instantiation of a literal in P, and (iii) all literals $M\#L$, where L is an instantiated literal and $(M,pred_L) \in Def_P$. The instantiation of a *PMP* is obtained by replacing each module with its instantiation which is defined in the usual way. In the rest of the paper, we assume that programs have been instantiated and thus all rules are propositional.

Let M be a module. We define $close_u(M)$ as the extended logic program that results if we replace every indexed literal in M with the special proposition u (meaning *undefined*). We say that M is ***internally consistent*** iff the *extended well-founded semantics* [54] of $close_u(M)$ is defined

(the *WFM* of $close_{tt}(M)$ is not contradictory). All modules of a *PMP* are assumed to be internally consistent.

Example 3.3.1: A *contig* is a set of overlapping DNA fragments that span some region of a genome [80]. One method to detect overlaps uses common features (restriction digest patterns, hybridization signals, STS hits). The idea is that if the same feature is found in two DNA fragments then they probably overlap. The overlap is not certain because of unreliable experimental results and feature repetition in the genome. Consider the following *PMP* (before renaming) with modules *OD* (*OverlapData*), *OF* (*OverlapFeature*), *F* (*Feature*): (terms that start with capital are variables)

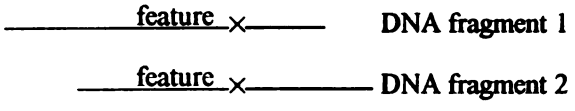

Figure 3.1: DNA fragments

module *OD* **exports** *overlap* /* Very reliable *Overlap* data */

rules ¬*overlap*(*frag*1,*frag*2).

module *OF* **exports** *overlap* **imports** *feat*

/* If the same feature *Feat* is found in fragments *Frag*1 and *Frag*2 then the fragments overlap */

rules *overlap*(*Frag*1,*Frag*2)← *feat*(*Frag*1,*Feat*), *feat*(*Frag*2,*Feat*).

module *F* **exports** *feat* /* A feature *x* is found in fragments *frag*1 and *frag*2 */

rules *feat*(*frag*1, *x*). *feat*(*frag*2, *x*).

$(F,feat) <_{def} (OD,overlap)$ /* *overlap* data in *OD* are more reliable than the experimental data in *F* */

Even though the modules *OD*, *OF* and *F* are internally consistent, their union is inconsistent because both *overlap*(*frag*1,*frag*2) and ¬*overlap*(*frag*1,*frag*2) can be derived from the above rules. Note that ¬*overlap*(*frag*1,*frag*2) is derived from module *OD* and that the derivation of *overlap*(*frag*1,*frag*2) from module *OF* depends on *feat*(*frag*1,*x*) and *feat*(*frag*2,*x*), exported by module *F*. Since $(F,feat) <_{def} (OD,overlap)$, i.e., the definition of *feat* in *F* is less reliable that of *overlap* in *OD*, the results *feat*(*frag*1,*x*) and *feat*(*frag*2,*x*) are considered unreliable. Thus, the truth value of the literals *feat*(*frag*1,*x*) and *feat*(*frag*2,*x*) is considered unknown and the literal ¬*overlap*(*frag*1,*frag*2) is evaluated as true. After the renaming mechanism is employed, modules become:

module *OD* **exports** *overlap*

rules ¬*OD*#*overlap*(*frag*1,*frag*2).

module *OF* **exports** *overlap* **imports** *feat*

rules *OF*#*overlap*(*Frag*1,*Frag*2)← {*F*}:*feat*(*Frag*1, *Feat*), {*F*}:*feat*(*Frag*2, *Feat*).

module *F* **exports** *feat*

rules *F*#*feat*(*frag*1, *x*). *F*#*feat*(*frag*2, *x*).

Definition 3.3.1 (interpretation): Let *P* be a *PMP*. An interpretation *I* is a set of literals $T \cup \sim F$, where *T* and *F* are disjoint sets of classical literals. *I* is *consistent* iff $T \cap \neg T = \emptyset$. *I* is *coherent* iff it satisfies the *coherence property*: $\neg T \subseteq F$.

An interpretation *I* can also be seen as a pair $<I_{ind}, I_{loc}>$, where I_{ind} is the set of *indexed* literals contained in *I* and $I_{loc}[M]$, for any module *M*, is the set of *local* literals of *M* contained in *I*.

Definition 3.3.2 (truth valuation of a literal): A literal L is true (resp. false) w.r.t. an interpretation I iff $L \in I$ (resp. $\sim L \in I$). A literal that is neither true nor false w.r.t. I, it is undefined w.r.t. I.

The truth value of a literal $M\#L$ represents the truth value of L in module M. A classical literal $\{M_1,...,M_n\}:L$ is true iff $M_i\#L$ is true for an $i \leq n$ and $\{M_1,...,M_n\}:L$ is reliable. A default literal $\sim\{M_1,...,M_n\}:L$ is true iff $\sim M_i\#L$ is true for all $i \leq n$ and $\sim\{M_1,...,M_n\}:L$ is reliable. Let I be a set of literals known to be true. In Definition 3.3.5, the concept of reliable indexed literal is defined which is used in defining the m-models and in the fixpoint computation of the modular reliable semantics. To decide if an indexed literal $S:L$ is reliable w.r.t. a definition (M,p), all possible derivations of complementary literals caused by definitions $(M',p') \not<_{def} (M,p)$ should be considered. $S:L$ is reliable if it does not contribute to any such derivation of complementary literals. Intuitively, $\mathbf{Pos}_{[M,p],I}$ contains the literals possible to derive when results exported by modules M' with $(M',pred_L) <_{def} (M,p)$ are ignored. $\mathbf{Pos}_{[M,p],I}$ is defined as the least fixpoint of a monotonic operator which resembles the well-founded semantics operator [76].

Definition 3.3.3 (possible literal set): Let P be a *PMP*, I,J sets of literals and $(M,p) \in Def_P$. The *possible literal set* w.r.t. (M,p) and I, denoted by $\mathbf{Pos}_{[M,p],I}$, is defined as follows:

- $\mathbf{PT}_{[M,p],I}(J) = \{M'\#L \mid \exists\, M'\#L \leftarrow L_1,...,L_n$ in module M' s.t. $L_i \in J,\ \forall i \leq n\} \cup$

$$\{S:L \in HB_P \mid \neg S:L \notin I \text{ and } \exists\, M' \in S \text{ s.t. } M'\#L \in J \text{ and } (M',pred_L) \not<_{def} (M,p)\}$$

- $\mathbf{PF}_{[M,p],I}(J)$ is the greatest set $U \subseteq HB_P$ s.t.

  (i) if $M'\#L \in U$ and r is a rule s.t. $Head_r = M'\#L$ then $\exists K \in Body_r$ s.t. $K \in U$ or $\sim K \in J$,

  (ii) if $S:L \in U$ then $\forall M' \in S$, $M'\#L \in U$ and $(M',pred_L) \not<_{def} (M,p)$.

- $\mathbf{PW}_{[M,p],I}(J) = coh(\mathbf{PT}_{[M,p],I}(J) \cup \sim\mathbf{PF}_{[M,p],I}(J))$.

- $\mathbf{Pos}_{[M,p],I}$ is the least fixpoint of the operator $\mathbf{PW}_{[M,p],I}$.

A local literal $M'\#L \in \mathbf{PT}_{[M,p],I}(J)$ iff there is a rule r with $Head_r = M'\#L$ and $Body_r \subseteq J$. Since modules are assumed to be internally consistent, we do not need to verify that $\neg M'\#L \notin I$. An indexed literal $\{M_1,...,M_n\}:L \in \mathbf{PT}_{[M,p],I}(J)$ iff $\neg\{M_1,...,M_n\}:L$ is not true w.r.t. I and for an $i \leq n$, L is true in M_i w.r.t. J and $(M_i, pred_L) \not<_{def} (M,p)$. Conditions (i) and (ii) in the definition of $\mathbf{PF}_{[M,p],I}(J)$, generalize the concept of unfounded set, as defined in [76]. The additional condition (ii) expresses that $\{M_1,...,M_n\}:L$ is false by default if for all $i \leq n$, L is false by default in M_i and $(M_i, pred_L) \not<_{def} (M,p)$. Coherence is enforced by the *coh* operator in the definition of $\mathbf{PW}_{[M,p],I}(J)$.

Intuitively, a literal $S:L$ is reliable w.r.t. (M,p) and I iff there are no K, $\neg K$ in $\mathrm{Pos}_{[M,p],I}$ s.t. the derivation of K depends on $S:L$. The *dependency set* of K w.r.t. (M,p) and I is the set of literals in $\mathrm{Pos}_{[M,p],I}$ that the derivation of K depends on. Since I is a set of literals known to be true, the *dependency set* of a literal $K \in I$ equals $\{\}$.

Definition 3.3.4 (dependency set): Let P be a *PMP*, I a set of literals and $(M,p) \in Def_P$. The *dependency set* of a literal K w.r.t. (M,p) and I, denoted by $\mathrm{Dep}_{[M,p],I}(K)$, is the least set $\mathrm{D}(K)$ such that if $K \in I$ then $\mathrm{D}(K)=\{\}$ else

(i) If $K = \sim M'\#L$ is a default literal then

(a) $\mathrm{D}(\neg M'\#L) \subseteq \mathrm{D}(\sim M'\#L)$ and

(b) $\forall\, M'\#L \leftarrow L_1,...,L_n$ in M', if $\sim L_i \in \mathrm{Pos}_{[M,p],I}$ for $i \leq n$ then $\mathrm{D}(\sim L_i) \subseteq \mathrm{D}(\sim M'\#L)$.

(ii) If $K = M'\#L$ is a classical literal then

if $M'\#L \leftarrow L_1,...,L_n$ in M' s.t. $\{L_1,...,L_n\} \subseteq \mathrm{Pos}_{[M,p],I}$ then $\mathrm{D}(L_i) \subseteq \mathrm{D}(M'\#L)$, $\forall i \leq n$.

(iii) If $K = \sim S:L$ is a default literal then

(a) $\mathrm{D}(\neg S:L) \subseteq \mathrm{D}(\sim S:L)$ and

(b) if $\forall M' \in S$, $(M', pred_L) \not<_{def} (M,p)$ and $\sim M'\#L \in \mathrm{Pos}_{[M,p],I}$ then $\mathrm{D}(\sim M'\#L) \subseteq \mathrm{D}(\sim S:L)$, $\forall M' \in S$ and $\sim S':L \in \mathrm{D}(\sim S:L)$, $\forall S' \subseteq S$.

(iv) If $K = S:L$ is a classical literal then

(a) $\mathrm{D}(M'\#L) \subseteq \mathrm{D}(S:L)$, $\forall M' \in S$ s.t. $(M', pred_L) \not<_{def} (M,p)$ and $S':L \in \mathrm{D}(S:L)$, $\forall S' \subseteq S$.

Case (i) expresses that $\sim M'\#L$ can be true because of the coherence inference rule or the default rule for negation. Case (ii) expresses that if $M'\#L \leftarrow L_1,...,L_n$ is a rule and all L_i are true then $M'\#L$ is true. Case (iii) expresses that if $\neg S:L$ is true or $\sim M'\#L$ is true for all $M' \in S$ then $\sim S:L$ is true. Case (iv) expresses that if $M'\#L$ is true for an $M' \in S$ then $S:L$ is true and if $S':L$ is true for an $S' \subseteq S$ then $S:L$ is true.

Definition 3.3.5 (reliable indexed literal): Let P be a *PMP*, I a set of literals, $S:L$ a literal, $(M,pred_L) \in Def_P$, $M \in S$ and p be equal to $pred_L$.

- The literal $S:L$ is *unreliable* w.r.t. (M,p) and I iff $\exists\ \neg K \in \text{Pos}_{[M,p],I}$ s.t.

  if $K = S':L$ with $S' \supset S$ then $S:L \in \text{Dep}_{[M,p],I}(M'\#L)$ for an $M' \in S'$ s.t. $(M',p) \not<_{def} (M,p)$

  else $S:L \in \text{Dep}_{[M,p],I}(K)$.

- The literal $S:L$ is *reliable* w.r.t. (M,p) and I iff it is not *unreliable* w.r.t. (M,p) and I.

Assume that $S:L$ is *unreliable* w.r.t. (M,p) and I. If K of Definition 3.3.5 is a local literal $M'\#L'$ then (i) the literals K, $\neg K$ are derived inside module M' and (ii) $S:L$ contributes to the derivation of K. If $K = S':L' \neq S:L$ is an indexed literal then: (i) there are literals $M'\#L'$, $\neg M''\#L'$ derived in modules $M' \in S'$ and $M'' \in S'$ s.t. $(M',p) \not<_{def} (M,p)$ and $(M'',p) \not<_{def} (M,p)$, and (ii) $S:L$ contributes to the derivation of $M'\#L'$. If $K = S:L$ then there are literals $M'\#L$, $\neg M''\#L$ derived in modules $M' \in S$ and $M'' \in S$ with $(M',p) \not<_{def} (M,p)$ and $(M'',p) \not<_{def} (M,p)$.

Example 3.3.2: Let P be the *PMP* of Example 3.3.1 and $I=\emptyset$.

$\text{Pos}_{[OD,overlap],I} = coh(\{\neg OD\#overlap(frag1,frag2),\ \neg\{OD\}:overlap(frag1,frag2),$

$\neg\{OD,OF\}:overlap(frag1,frag2)\})$

The literal $\neg\{OD\}:overlap(frag1,frag2)$ is reliable w.r.t. $(OD,overlap)$ and I because

$\neg\{OD\}:overlap(frag1,frag2) \notin \text{Dep}_{[OD,overlap],I}(\neg K),\ \forall K \in \text{Pos}_{[OD,overlap],I}$.

Similarly, $\neg\{OD,OF\}:overlap(frag1,frag2)$ is reliable w.r.t. $(OD,overlap)$ and I.

$\mathbf{Pos}_{[F,feat],I} = coh(\{F\#feat(frag1,x),\ \{F\}:feat(frag1,x),\ F\#feat(frag2,x),\ \{F\}:feat(frag2,x),$

$OF\#overlap(frag1,frag2),\ \{OF\}:overlap(frag1,frag2),\ \{OD,OF\}:overlap(frag1,frag2),$

$\neg OD\#overlap(frag1,frag2),\ \neg\{OD\}:overlap(frag1,frag2),\ \neg\{OD,OF\}:overlap(frag1,frag2)\})$

The literal $\{F\}:feat(frag1,x)$ is unreliable w.r.t. $(F,feat)$ and I because

(i) $\neg\{OD,OF\}:overlap(frag1,frag2) \in \mathbf{Pos}_{[F,feat],I}$ and

(ii) $\{F\}:feat(frag1,x) \in \mathbf{Dep}_{[F,feat],I}(\{OD,OF\}:overlap(frag1,frag2))$.

Similarly, $\{F\}:feat(frag2,x)$ is unreliable w.r.t. $(F,feat)$ and I.

$\mathbf{Pos}_{[OF,overlap],I} = \mathbf{Pos}_{[F,feat],I}$. The literal $\{OF\}:overlap(frag1,frag2)$ is reliable w.r.t. $(OF,overlap)$ and I whereas the literal $\{OD,OF\}:overlap(frag1,frag2)$ is not.

Definition 3.3.6 (*m*-model): Let P be a *PMP*. A consistent, coherent interpretation I is an *m-model* of P iff (i) $\forall$ rule r, $I(Head_r) \geq I(Body_r)$ and (ii) $\forall$ classical literal $S{:}L$, both of the following are true:

– if $I(S{:}L) \neq 1$ then $\forall M \in S$ s.t. $I(M\#L)=1$, $S{:}L$ is unreliable w.r.t. $(M,pred_L)$ and I

– if $I(S{:}L)=0$ then $I(\neg S{:}L)=1$ or $\forall M \in S$, $I(M\#L)=0$.

Since condition (i) defines 3-valued models [60], an *m*-model of P is a 3-valued model of every module of P. In condition (ii), the first subcondition expresses that if $S{:}L$ is a classical literal, $M \in S$ and $I(M\#L)=1$ then $I(S{:}L)$ can be $\neq 1$ only if $S{:}L$ is unreliable w.r.t. $(M,pred_L)$ and I. The purpose of the second subcondition in condition (ii) is to allow $S{:}L$ to be false when $\neg S{:}L$ holds, even if $\exists M \in S$ s.t. $I(M\#L)>0$ (L is not false in M).

Example 3.3.3: Let P be as in Example 3.3.1. Then, I is an *m*-model of P:

I=*coh*({F#*feat*(*frag*1,x), F#*feat*(*frag*2,x), ¬OD#*overlap*(*frag*1,*frag*2), ¬{OD}:*overlap*(*frag*1,*frag*2), ¬{OD,OF}:*overlap*(*frag*1,*frag*2)}).

3.4 Modular Reliable Semantics for Prioritized Modular Logic Programs

In this Section, we define the *reliable m-model*, *stable m-models* and *modular reliable semantics* of a *PMP*, P. We show that the reliable m-model of P is the least stable m-model of P.

3.4.1 Reliable *m*-model

An m-unfounded set U w.r.t. J is also an unfounded set w.r.t. J according to [76] with an additional constraint: if S:$L \in U$ then ~S:L should be reliable w.r.t. (M,$pred_L$) and J, $\forall M \in S$.

Definition 3.4.1 (*m*-unfounded set): Let P be a *PMP* and J a set of literals. A literal set $U \subseteq HB_P$ is m-unfounded w.r.t. J iff

(i) if M#$L \in U$ and r is a rule with $Head_r = M$#L then $\exists K \in Body_r$ s.t. $K \in U$ or ~$K \in J$,

(ii) if S:$L \in U$ then $\forall M \in S$, M#$L \in U$ and ~S:L is reliable w.r.t. (M,$pred_L$) and J.

The $\mathbf{W}_P$ operator extends the $\mathbf{W}_P$ operator of the well-founded semantics [76], to *PMPs*.

Definition 3.4.2 ($\mathbf{W}_P$ operator): Let P be a *PMP* and J a set of literals. We define:

- $\mathbf{T}(J)$={M#L | $\exists$ M#$L \leftarrow L_1,...,L_n$ in module M s.t. $L_i \in J$, $\forall i \leq n$} $\cup$
  {S:$L \in HP_P$ | M#$L \in J$, $M \in S$ and S:L is reliable w.r.t. (M,$pred_L$) and J}
- $\mathbf{F}(J)$ is the greatest m-unfounded set w.r.t. J.
- $\mathbf{W}_P(J)$=*coh*(T(J)$\cup$~F(J)).

The union of two m-unfounded sets w.r.t. J is an m-unfounded set w.r.t. J. So, F(J) is the union of all m-unfounded sets w.r.t. J. We define the transfinite sequence $\{I_\alpha\}$ as follows:

$I_0=\{\}$, $I_{a+1}=W_P(I_a)$ and $I_a=\cup\{I_b \mid b<a\}$ if a is a limit ordinal.

Proposition 3.4.1: Let P be a *PMP*. $\{I_a\}$ is a monotonically increasing (w.r.t. $\subseteq$) sequence of consistent, coherent interpretations of P.

Proof: We will show that W_P is a monotonic operator. Let I,J be interpretations of P s.t. $I\subseteq J$. Then, $T(I)\subseteq T(J)$ because if a classical literal $S{:}L$ is reliable w.r.t. a predicate definition $(M,pred_L)$ and I then $S{:}L$ is also reliable w.r.t. $(M,pred_L)$ and J. $F(I)\subseteq F(J)$ because if a default literal $\sim S{:}L$ is reliable w.r.t. a predicate definition $(M,pred_L)$ and I then $\sim S{:}L$ is also reliable w.r.t. $(M,pred_L)$ and J. Since *coh* is a monotonic operator, W_P is a monotonic operator and $\{I_a\}$ is a monotonically increasing sequence w.r.t. $\subseteq$.

We will prove by induction that for all a, there is no literal K s.t. $\{K,\neg K\}\subseteq I_a$. This is true for $a=0$. Assume that it is true for ordinals $<a$. We will prove that it is true for a. Assume first that $a=b+1$ is a successor ordinal. Assume that there is literal K s.t. $\{K,\neg K\}\subseteq I_a$. Since I_b is consistent we can assume that $K\notin I_b$. Since $\{K,\neg K\}\subseteq I_a$ and the modules are internally consistent, there is classical or default literal $S{:}L\in I_b$ which in the computation of I_b is considered reliable w.r.t. a predicate definition $(M,pred_L)$ and I_c, for $c<b$, $\neg K\in Pos_{[M,P],I_c}$ and $S{:}L\in Dep_{[M,P],I_c}(K)$. However, this is a contradiction. Thus, I_a is consistent.

Let a be a limit ordinal and assume that there is literal K s.t. $\{K,\neg K\}\subseteq I_a$. Then, there is a successor ordinal $b+1<a$ s.t. $\{K,\neg K\}\subseteq I_{b+1}$. This is a contradiction because of the inductive hypothesis. So, I_a is consistent for all a.

We will prove by induction that for all a, there is no literal L s.t. $L\in I_a$ and $\sim L\in I_a$. It is true for $a=0$. Assume that it is true for ordinals $<a$. We will prove that it is true for a. Assume first that $a=b+1$ is a successor ordinal. We will prove that there is no literal L s.t. $L\in I_a$ and $\sim L\in I_a$. This is true for $a=0$. Assume this is true for ordinals $<a$. Let S be any set of classical literals that has a non-empty intersection with $T(I_b)$. Choose the smallest c s.t. I_{c+1} has a non-empty intersection

with S. Note that $c \leq b$. Let $K \in I_{c+1} \cap S$. If K is a local literal $M\#L$ then $M\#L$ is derived from a rule r s.t. $Body_r \subseteq I_c$. If K is a global literal $S{:}L$ then $\exists M \in S$ s.t. $M\#L$ is derived from a rule r with $Body_r \subseteq I_{c'}$, $c' < c$. From hypothesis, there is no literal $K \in Body_r$, s.t. $\sim K \in I_b$. Moreover, from the way r is defined, there is no classical literal K in $Body_r$ s.t. $K \in S$. So, S is not m-unfounded w.r.t. I_b. This implies that $\mathbf{T}(I_b) \cap \mathbf{F}(I_b) = \emptyset$. Moreover, there is no classical literal L s.t. $L \in \mathbf{T}(I_b)$ and $\neg L \in \mathbf{T}(I_b)$, because I_a does not violate any constraint. So, there is no literal L s.t. $L \in I_a$ and $\sim L \in I_a$. Let a be a limit ordinal and assume that there is L s.t. $L \in I_a$ and $\sim L \in I_a$. Then, there is a successor ordinal $b+1 < a$ s.t. $L \in I_{b+1}$ and $\sim L \in I_{b+1}$. This is a contradiction because of the inductive hypothesis.

I_a is a coherent interpretation, for all a, because of the *coh* operator in the definition of $\mathbf{W}_P$. Proposition 3.4.1 follows. ◊

Since $\{I_a\}$ is monotonically increasing (w.r.t. $\subseteq$), there is a smallest ordinal d s.t. $I_d = I_{d+1}$.

Proposition 3.4.2: Let P be a *PMP*. Then, I_d is an m-model of P.

Proof: From Proposition 3.4.1, I_d is a consistent, coherent interpretation. Let r be a rule in P. We will show that r is m-true w.r.t. I_d.

(i) For every rule r, $I_d(Head_r) \geq I_d(Body_r)$.

(ii) If $S{:}L$ is a classical literal, $I_d(S{:}L) \neq 1$, and $\exists M \in S$ s.t. $I_d(M\#L) = 1$ then $S{:}L$ is unreliable w.r.t. $(M, pred_L)$ and I_d because otherwise, from the definition of $\mathbf{T}(I_d)$, $I_d(S{:}L) = 1$.

(iii) If $S{:}L$ is a classical literal, $I_d(S{:}L) = 0$, and $\exists M \in S$ s.t. $I_d(M\#L) \neq 0$ then $I_d(\neg S{:}L) = 1$ because otherwise, from the definition of $\mathbf{F}(I_d)$, $I_d(S{:}L) \neq 0$. ◊

Definition 3.4.3 (modular reliable semantics): Let P be a *PMP*. The *reliable m-model* of P, denoted as RM_P, is the interpretation I_d. The *modular reliable semantics* of P is the "meaning" represented by RM_P.

It is possible that a local literal $M\#L \in RM_P$ but $\{M\}:L \notin RM_P$. This, intuitively, means that module M concludes L but that conclusion may be erroneous.

Example 3.4.1: Let P be the program of Example 3.3.1 and I be the m-model of Example 3.3.3. Then, I is the reliable m-model of P. I can also be seen as a pair (I_{ind}, I_{loc}), where:

$I_{ind} = coh(\{ \neg\{OD\}:overlap(frag1,frag2),\ \neg\{OD,OF\}:overlap(frag1,frag2)\})$ and

$I_{loc}[F] = coh(\{F\#feat(frag1,x), F\#feat(frag2,x)\})$, $I_{loc}[OF]=\{\}$ and

$I_{loc}[OD] = coh(\{\neg OD\#overlap(frag1,frag2)\})$.

When $<_{def}=\{\}$, $RM_P = coh(\{F\#feat(frag1,x), F\#feat(frag2,x), \neg OD\#overlap(frag1,frag2)\})$.

Proposition 3.4.3: Let P be a *PMP*. The complexity of computing RM_P is $O(|HB_P|^3 * |R_P|)$.

Proof: The following algorithm, **RM**(program P), returns the reliable m-model of P. To compute $\mathbf{F}(I)$, its complement set is constructed first, as in [76].

```
RM(PMP program P)
{ new_I={};
  repeat
    I=new_I;
    for each (M,p)∈ Def_P do        /* Step 1 */
          compute Pos_[M,p],I; endfor          /* Step 1.1 */
          for each L∈HB_P do compute Dep_[M,p],I(L); endfor    /* Step 1.2 */
    endfor

    repeat          /* Step 2: Compute T(I) */
       for each rule r in P do
          if Body_r ⊆new_I then add Head_r to new_I; endif
       endfor
       for each M#L∈ new_I do
         for each S:L∈HB_P do
          if M∈S and S:L is reliable w.r.t. (M,pred_L) and I then add S:L to new_I; endif
         endfor
       endfor
    until no change in new_I;
```

compl_F={*S*:*L*∈HB_P | ~*S*:*L* is unreliable w.r.t. (*M*,$pred_L$) and *I*, for an *M*∈*S*}; /* Step 3 */
repeat /* Step 4: Compute HB_P –**F**(*I*) */
for each rule *r* in *P* **do**
if no literal in $Body_r$ is false w.r.t. *I* and all classical literals in $Body_r$ are in *compl_F*
then add $Head_r$ to *compl_F*;
endif
endfor
until no change in *compl_F*;
for each *L*∈HB_P **do** /* Step 5*/
if *L*∉*compl_F* **then add** ~*L* to *new_I*; **endif**
endfor

new_I=*coh*(*new_I*); /* Step 6: Compute *coh*(**T**(*I*)∪~**F**(*I*)) */
until *I*=*new_I*;
return(*I*);
}

The complexity of computing Pos_I is the same as that of computing the well-founded model of $P_{[M,p],I}$, where $P_{[M,p],I} = P \cup \{S{:}L \leftarrow M'\#L \mid S{:}L \in HB_P, M' \in S, \neg S{:}L \notin I \text{ and } (M', pred_L) \not<_{def} (M,p)\} \cup \{S{:}L \leftarrow u \mid \exists M' \in S, \text{ and } (M', pred_L) <_{def} (M,p)\}$ and every literal $\neg L$ is replaced by a new atom $\neg\_L$. So, the complexity of Step 1.1 is $|HB_P|*|R_P|$ [Witt91, Schl92]. The complexity of Step 1.2 is $|HB_P|*|R_P|$ because the complexity of computing $Dep_I(L)$, for a literal *L*, is $|R_P|$. So, the complexity of Step 1 is $|Def_P|*|HB_P|*|R_P| < |HB_P|^2*|R_P|$. The complexity of Step 2 is $|HB_P|^2*|R_P|$ since $Pos_{[M,p],I}$ and $Dep_{[M,p],I}(L)$, $\forall L \in HB_P$, have already been computed. The complexity of Step 3 is $|HB_P|^2$ and that of Step 4 is $|R_P|$ [DoGa84]. The complexity of Steps 5 and 6 is $|HB_P|$. Since $\{I_a\}$ is a monotonically increasing sequence w.r.t. ⊆, the total number of iterations until *I*=*new_I*, is less than $|HB_P|$. So, the complexity of the algorithm **RM**(*P*) is $O(|HB_P|^3*|R_P|)$. ◊

According to *MRS* presented in the previous sections, the confidence in a globally indexed default literal ~*L*, derived by the default rule for negation, depends on the minimal priorities of (*M*,$pred_L$)∈Def_P. Thus, in case of conflict, ~*L* may not be considered less reliable than literals that their derivation is not based on closed-world assumptions. When this is undesirable for a set of

predicates $Pred_{\sim}$, a new module $M_{\sim}$ can be added which has no rules but exports all predicates in $Pred_{\sim}$. Moreover, $(M_{\sim},p') <_{def} (M,p)$ for all $p' \in Pred_{\sim}$ and definitions (M,p) other than $(M_{\sim},p)$.

Example 3.4.2: Consider the *PMP*, *P*:

module M_1 **exports** *broken*

rules M_1#*broken*$\leftarrow \sim M_1$#*spark*. M_1#*spark*.

module M_2 **exports** *start*

rules M_2#*start* $\leftarrow \sim$*broken*. /* ~*broken* represents a globally indexed literal */

module M_3 **exports** *start* /* *observed data* */

rules $\neg M_3$#*start*.

$<_{def}=\{\}$.

The *MRS* of *P* is: *coh*($\{M_1$#*spark*, $\sim M_1$#*broken*, $\neg M_3$#*start*, $\neg\{M_3\}$:*start*$\}$). Note that the truth value of the literal $\{M_2, M_3\}$:*start* is unknown. This expresses that M_2, M_3 export conflicting results about *start* and that none of these results has higher priority than the other. One may argue that closed-world assumptions should have lower priority than explicit rules. Thus, the result exported from M_2 should count less than that of M_3 and $\{M_2,M_3\}$:*start* should be evaluated as true. This is obtained by adding a module $M_{\sim}$ to *P* as follows:

module $M_{\sim}$ **exports** *broken*

rules {}

$(M_{\sim}$,*broken*$) <_{def} (M_1$,*broken*$)$, $(M_{\sim}$,*broken*$) <_{def} (M_2$,*start*$)$ and $(M_{\sim}$,*broken*$) <_{def} (M_3$,*start*$)$.

The *MRS* of the new *P* is: *coh*($\{M_1$#*spark*, $\sim M_1$#*broken*, $\neg M_3$#*start*, $\neg\{M_3\}$:*start*, $\neg\{M_2,M_3\}$:*start*$\}$).

3.4.2 Stable *m*-models

The reliable *m*-model of a *PMP* corresponds to its skeptical meaning. Credulous meanings can be obtained using the transformation $P/_mI$, where I is an interpretation of P. The transformation P/I is defined in [26, 60] for a normal program P. $P/_mI$ extends P/I to *PMPs*.

Definition 3.4.4 (transformation $P/_mI$): Let P be a *PMP* and I an interpretation. The program $P/_mI$ is obtained from P as follows:

(i) Remove all rules that contain in their body a default literal $\sim L$ such that $I(L)=1$.

(ii) Remove all rules r such that $I(\neg Head_r)=1$.

(iii) Remove from the body of the remaining rules any default literal $\sim L$ such that $I(L)=0$.

(iv) Replace all remaining default literals $\sim L$ with u.

(v) For all $S{:}L \in HB_P$ s.t. $I(S{:}L)=1/2$,

- for all $M \in S$, if $I(M\#L)=1$ then add $S{:}L \leftarrow u$ else add $S{:}L \leftarrow M\#L$,
- if $\exists M \in S$ s.t. $\sim S{:}L$ is unreliable w.r.t. $(M, pred_L)$ and I then add $S{:}L \leftarrow u$.

(vi) For all $S{:}L \in HB_P$ s.t. $I(S{:}L) \neq 1/2$ and $I(\neg S{:}L) \neq 1$, add $S{:}L \leftarrow M\#L$, $\forall M \in S$.

The program $P/_mI$ is a non-negative program with a special proposition u. For any interpretation J, $J(u)=1/2$. When P is a normal program and I is a model of P [60], $P/_mI \equiv P/I$ since Steps (iv) and (v) do not have any effect on $P/_mI$. We say that a model I of P is the *least$_v$ model* of P iff $I(L) \leq I'(L)$, for any model I' and classical literal L of P.

Definition 3.4.5 (stable *m*-model): Let P be a *PMP* and I an *m*-model of P. I is a *stable m-model* of P iff $least_v(P/_mI)=I$.

Let I be a stable *m*-model of P. If $S{:}L$ is unreliable w.r.t. $(M, pred_L)$ and I, for an $M \in S$ then the truth value of $S{:}L$ can be unknown w.r.t. I even if $I(M\#L)=1$. If $\sim S{:}L$ is unreliable w.r.t. $(M, pred_L)$

and I, for an $M \in S$ then the truth value of $S{:}L$ can be unknown w.r.t. I even if $I(M\#L)=0$, for all $M \in S$.

The *export rule set* of P is defined as $ER_P = \{S{:}L \leftarrow M\#L \mid S{:}L \in HB_P \text{ and } M \in S\} \cup \{\sim S{:}L \leftarrow\sim M_1\#L,\ldots,\sim M_n\#L \mid S{:}L \in HB_P \text{ and } S = \{M_1,\ldots,M_n\}\}$. An interpretation I of P satisfies $r \in ER_P$ iff $I(Body_r) \neq 1$ or $I(Head_r)=1$. Let I_1, I_2 be two stable m-models of P. We say that $I_1 \leq_{sat} I_2$ iff (i) $\forall r \in ER_P$, if I_1 satisfies r then I_2 satisfies r and (ii) $I_1 \subseteq I_2$ or $\exists r \in ER_P$ s.t. I_2 satisfies r and I_1 does not satisfy r. In other words, $I_1 \leq_{sat} I_2$ iff I_2 satisfies more export rules than I_1 or ($I_1 \subseteq I_2$ and I_2 satisfies the same export rules as I_1). Maximal (w.r.t. $\leq_{sat}$) stable m-models can be seen as the credulous meanings of P.

Example 3.4.3: Let P be the program of Example 3.3.1. RM_P is given in Example 3.4.1. Then, P has four stable m-models:

$I_1=RM_P$, $I_2=RM_P \cup coh(\{\{F\}{:}feat(frag1,x)\})$, $I_3=RM_P \cup coh(\{\{F\}{:}feat(frag2,x)\})$ and

$I_4=RM_P \cup coh(\{\{F\}{:}feat(frag1,x), \{F\}{:}feat(frag2,x), OF\#overlap(frag1,frag2),$

$\{OF\}{:}overlap(frag1,frag2)\})$.

I_2, I_3 and I_4 are maximal (w.r.t. $\leq_{sat}$) stable m-models of P. Note that

Model I_2 does not satisfy $\{F\}{:}feat(frag2,x) \leftarrow F\#feat(frag2,x)$,

Model I_3 does not satisfy $\{F\}{:}feat(frag1,x) \leftarrow F\#feat(frag1,x)$ and

Model I_4 does not satisfy $\{OD,OF\}{:}overlap(frag1,frag2) \leftarrow OF\#overlap(frag1,frag2)$.

Proposition 3.4.4: Let P be a *PMP*. The reliable m-model of P is a stable m-model of P.

Proof: Let RM be the reliable m-model of P. From Proposition 3.4.2, RM is an m-model of P. So, it is enough to show that $RM=least_v(Pl_m RM)$. Let $least_v(Pl_m RM)=T \cup \sim F$, where T, F are sets of classical literals. Let $I_a=T_a \cup \sim F_a$, where T_a, F_a are sets of classical literals and $RM = I_d$. First, we will prove by induction that $T_b \cup \sim F_b \subseteq T \cup \sim F$, $\forall b \leq d$. It is true that $T_0 \subseteq T$ and $F_0 \subseteq F$. Suppose that $T_a \subseteq T$ and $F_a \subseteq F$, $\forall a < b$. If b is a limit ordinal then $T_b \subseteq T$ and $F_b \subseteq F$ since $I_b = \cup\{I_a \mid a<b\}$.

Assume therefore that $b=a+1$. Let $L \in I_b$. If L is the local literal $M\#L'$ then $\exists\ M\#L' \leftarrow L_1,...,L_n$ in M s.t. $\forall i \leq n$, $L_i \in I_a$. Since $I_a \subseteq T \cup \sim F$, there is a rule $M\#L' \leftarrow L'_1,...,L'_m$ in $P/_m RM$ where $L'_1,...,L'_m$ are all the classical literals in $\{L_1,...,L_n\}$. From the fact $I_a \subseteq T \cup \sim F$ and the definition of $least_v(P/_m RM)$, it follows that $M\#L' \in T$. If L is the indexed literal $S:L'$ then $\exists M \in S$, s.t. $M\#L' \in I_a$. Since $RM(S:L')=1$, the rule $S:L' \leftarrow M:L'$ is in $P/_m RM$. From the fact $I_a \subseteq T \cup \sim F$ and the definition of $least_v(P/_m RM)$, it follows that $S:L' \in T$. This implies that $\mathbf{T}(I_a)=T_b \subseteq T$.

Now, we will show that $F_b \subseteq F$. Since $F_b = \neg T_b \cup \mathbf{F}(I_a)$, it is enough to show that $\neg T_b \subseteq F$ and $\mathbf{F}(I_a) \subseteq F$. If $L \in \neg T_b$ then $\neg L \in RM$ and from Steps (ii), (vi) of Def. 3.4.4, there is no rule with head L in $P/_m RM$. Consequently, $L \in F$ and $\neg T_b \subseteq F$. For all rules $M\#L \leftarrow L'_1,...,L'_m, \sim L_1,..,\sim L_n$ in P (L_i, L'_i are classical literals) with $M\#L \in \mathbf{F}(I_a)$ either $\exists i \leq m$, $L'_i \in \mathbf{F}(I_a) \cup F_a$ or $\exists j \leq n$, $L_j \in T_a$. This implies that for each rule $M\#L \leftarrow L'_1,...,L'_m, \sim L_1,..,\sim L_n$ in P with $M\#L \in \mathbf{F}(I_a)$ either there is a corresponding rule $M\#L \leftarrow A_1,...,A_k$ in $P/_m RM$ (from Steps (iii) and (iv) of Def. 3.4.4) with $A_i \in \mathbf{F}(I_a) \cup F$ for an $i \leq k$ or there is no corresponding rule in $P/_m RM$ (from the Steps (i) and (ii) of Def. 3.4.4). If r is a rule in $P/_m RM$ with head an indexed literal $S:L \in \mathbf{F}(I_a)$ then r is of the form $S:L \leftarrow M\#L$ with $M\#L \in \mathbf{F}(I_a)$. Note that, no rule $S:L \leftarrow u$ is added to $P/_m RM$ (from Step (v) of Def. 3.4.4) because $S:L$ is false w.r.t. RM. So, for each rule $H \leftarrow A_1,...,A_k$ in $P/_m RM$ with $H \in \mathbf{F}(I_a) \cup F$, $\exists i \leq k$ such that $A_i \in \mathbf{F}(I_a) \cup F$. From the definition of $least_v(P/_m RM)$, it follows that $\mathbf{F}(I_a) \subseteq F$. Consequently, $F_b \subseteq F$.

So, we proved that $T_d \subseteq T$ and $F_d \subseteq F$.

We will show that $T \subseteq T_d$. Let a be the first ordinal s.t. there is a literal $L \notin T_d$ and $\Psi_{P'}\uparrow^{a+1}(\emptyset)(L)=1$, where $P'=P/_m RM$. If L is the local literal $M\#L'$ then there is a rule $M\#L' \leftarrow A_1,...,A_k$ in $P/_m RM$ with $\Psi_{P'}\uparrow^{a}(\emptyset)(A_i)=1$, $\forall i \leq k$. This implies that there is a rule in P with head $M\#L'$ whose body literals are true w.r.t. RM. Consequently, $M\#L' \in T_d$ which is a contradiction. If L is the indexed literal $S:L'$ then there is a rule $S:L' \leftarrow M\#L'$ in $P/_m RM$ with

$\Psi_{P'}{}^{\uparrow a}(\emptyset)(M\#L')=1$, $\forall i \leq k$. Consequently, there is an $M \in S$ s.t. $M\#L' \in T_d$. Since $L \notin T_d$, it follows that $\neg L \in T_d$ or L is unknown w.r.t. RM. If $\neg L \in T_d$ then from Step (vi) of Def. 3.4.4, $L \notin T$ which is a contradiction. If L is unknown w.r.t. RM then the rule r should not exist in $P/_m RM$ because of the Step (v) of Def. 3.4.4 and the fact that $M\#L' \in T_d$. So, $L \in T_d$ and consequently $T \subseteq T_d$.

We will show that $F \subseteq F_d$. Let $F_{coh}=\{H \mid \neg H \in T_d\}$. $F_{coh} \subseteq F_d$ because RM is a coherent interpretation. For all rules $H \leftarrow A_1,...,A_k$ in $P/_m RM$ with $H \in F-F_{coh}$, there is $i \leq k$ such that $A_i \in F$. This implies that for each rule $H \leftarrow L'_1,...,L'_m,\sim L_1,...,\sim L_n$ in P (L_i, L'_i are classical literals) with $H \in F-F_{coh}$ either (i) $\exists i \leq m$, $L'_i \in F$ (from Steps (iii) and (iv) of Def. 3.4.4) or (ii) $\exists j \leq n$, $L_j \in T_d$ (from Step (i) of Def. 3.4.4). Let $S:L \in F-F_{coh}$. Then, $RM(S:L) \neq 1$ because otherwise, $S:L \in T$ which is a contradiction. We will show that $\forall M \in S$, $M\#L \in F$. Assume that $\exists M \in S$, $M\#L \notin F$. If $RM(\neg S:L)=1$ then $S:L \in F_{coh}$, which is a contradiction. If $RM(S:L)=1/2$ then there is a rule $S:L \leftarrow u$ in $P/_m RM$ (from Step (v) of Def. 3.4.4), which is a contradiction since $S:L \in F$. Thus, $RM(S:L)=0$ and for all $M \in S$, $M\#L \notin F$. We will show that $\forall M \in S$, $\sim S:L$ is reliable w.r.t. $(M, pred_L)$ and RM. If $\sim S:L$ is unreliable w.r.t. $(M, pred_L)$ and RM for an $M \in S$ then $S:L \notin \mathrm{F}(I_d)$ and consequently, $RM(S:L) \geq 1/2$ since $S:L \notin F_{coh}$. However, $S:L \notin T$ since $S:L \in F$ and consequently, $RM(S:L) \neq 1$. So, $RM(S:L)=1/2$ and the rule $S:L \leftarrow u$ should be added to $P/_m RM$ (from Step (v) of Def. 3.4.4). This implies that $S:L \notin F$, which is a contradiction. So, if $S:L \in F-F_{coh}$ then $\forall M \in S$, $\sim S:L$ is reliable w.r.t. $(M, pred_L)$ and RM. Since $\mathrm{F}(I_d)$ is the maximum set that satisfies the property satisfied by set $F-F_{coh}$, $F-F_{coh} \subseteq \mathrm{F}(I_d)$. So, $F \subseteq F_d$.

Consequently, $RM=T_d \cup \sim F_d = T \cup \sim F = least_v(P/_m RM)$. ◊

Proposition 3.4.5: The reliable m-model of a *PMP* is its least stable m-model.

Proof: Let RM be the reliable m-model of P. From Proposition 3.3.4, RM is a stable m-model of P. So, it is enough to show that if I is a stable m-model of P then $RM \subseteq I = least_v(P/_m M)$. Let $I=T \cup \sim F$, where T, F are sets of classical literals. Let $I_a=T_a \cup \sim F_a$, where T_a, F_a are sets of classical literals

and $RM = I_d$. We will show by induction that $I_b \subseteq T \cup \sim F$, $\forall b \leq d$. It is true that $T_0 \subseteq T$ and $F_0 \subseteq F$. Suppose that $T_a \subseteq T$ and $F_a \subseteq F$, $\forall a < b$. If b is a limit ordinal then $T_b \subseteq T$ and $F_b \subseteq F$ since $I_b = \cup\{I_a | a < b\}$. Assume therefore that $b = a+1$. Let $L \in I_b$. If L is the local literal $M\#L'$ then $\exists\, M\#L' \leftarrow L_1,...,L_n$ in P s.t. $\forall i \leq n$, $L_i \in I_a$. Since $I_a \subseteq T \cup \sim F$, $M\#L' \in T$. If L is the indexed literal $S:L'$ then $\exists M \in S$, s.t. $M\#L' \in I_a$ and $S:L'$ is reliable w.r.t. $(M, pred_{L'})$ and I_a. Since $I_a \subseteq I$, it follows that $S:L'$ is reliable w.r.t. $(M, pred_{L'})$ and I. From the facts that I is an m-model of P, $I_a \subseteq I$, and $S:L'$ is reliable w.r.t. $(M, pred_{L'})$ and I, it follows that $L \in T$. So, $\mathbf{T}(I_a) = T_b \subseteq T$.

Now, we will show that $F_b \subseteq F$. Since $F_b = \neg T_b \cup \mathbf{F}(I_a)$, it is enough to show that $\neg T_b \subseteq F$ and $\mathbf{F}(I_a) \subseteq F$. If $L \in \neg T_b$ then $\neg L \in I$ and from Steps (ii), (vi) of Def. 3.4.4, there is no rule with head L in $P/_m I$. Consequently, $L \in F$ and $\neg T_b \subseteq F$. For all rules $M\#L \leftarrow L'_1,...,L'_m, \sim L_1,..,\sim L_n$ in P (L_i, L'_i are classical literals) with $M\#L \in \mathbf{F}(I_a)$ either $\exists i \leq m$, $L'_i \in \mathbf{F}(I_a) \cup F_a$ or $\exists j \leq n$, $L_j \in T_a$. This implies that for each rule $M\#L \leftarrow L'_1,...,L'_m, \sim L_1,..,\sim L_n$ in P with $M\#L \in \mathbf{F}(I_a)$ either there is a corresponding rule $M\#L \leftarrow A_1,...,A_k$ in $P/_m I$ (from Steps (iii) and (iv) of Def. 3.4.4) with $A_i \in \mathbf{F}(I_a) \cup F$ for an $i \leq k$ or there is no corresponding rule in $P/_m I$ (from the Steps (i) and (ii) of Def. 3.4.4). If r is a rule in $P/_m I$ with head an indexed literal $S:L \in \mathbf{F}(I_a)$ then r is of the form $S:L \leftarrow M\#L$ with $M\#L \in \mathbf{F}(I_a)$. Note that, no rule $S:L \leftarrow u$ is added to $P/_m I$ (from Step (v) of Def. 3.4.4) because the facts $I_a \subseteq I$ and $least_v(P/_m I) = I$ imply that for all $M \in S$, $M\#L \notin T$. So, for each $H \in \mathbf{F}(I_a) \cup F$, if $H \leftarrow A_1,...,A_k$ is a rule in $P/_m I$ then $\exists i \leq k$ such that $A_i \in \mathbf{F}(I_a) \cup F$. From the definition of $least_v(P/_m I)$, it follows that $\mathbf{F}(I_a) \subseteq F$. Consequently, $F_b \subseteq F$.

So, we proved that $T_d \subseteq T$ and $F_d \subseteq F$. ◊

3.5 Modular Reliable Semantics for Prioritized Extended Logic Programs

In this section, we define the modular reliable semantics of prioritized extended logic programs and we relate it with the modular reliable semantics of prioritized modular logic programs.

3.5.1 Definitions

A *prioritized extended logic program* (*PEP*), P, is a pair $<R_P, <_R>$. The set R_P contains a set of rules $r::L_0 \leftarrow L_1,..., L_m, \sim L_{m+1},..., \sim L_n$, where r is the label of the rule and L_i are classical literals. The partial order $<_R \subseteq R_P \times R_P$ denotes the relative reliability of the rules. A *PEP* is an *extended program* iff $<_R = \{\}$.

A *PEP*, P, can be translated into a *PMP*, $tr_M(P)$, by considering each rule as a module that imports all of its body predicates and exports its head predicate. Consequently, we can define the *modular reliable semantics* of P from that of $tr_M(P)$, that is, if I_M is an interpretation, m-model, stable m-model or the reliable m-model of $tr_M(P)$ then $I=\{L \mid S_L:L \in I_M\}$ is an interpretation, m-model, stable m-model or the reliable m-model of P, respectively. Note that, every indexed literal appearing in $tr_M(P)$ is globally indexed. We say that a rule r in P is reliable (resp. unreliable) w.r.t. I iff $S_{Head_r}:Head_r$ is reliable (resp. unreliable) w.r.t. $(r,Head_r)$ and $I_M=\{S_L:L \mid L \in I\}$ in $tr_M(P)$.

Direct definitions of the stable m-models and the reliable m-model of a *PEP* which are equivalent to the above definitions, are given below.

Definition 3.5.1 (possible literal set): Let P be a *PEP*, I,J sets of literals and $r \in R_P$. The *possible literal set* w.r.t. r and I, denoted by $\text{Pos}_{r,I}$, is defined as follows:

- $\mathbf{PT}_{r,I}(J)=\{L \mid \exists\, r'::L \leftarrow L_1,...,L_n$ in P s.t. (i) $L_i \in J$, $\forall i \leq n$, (ii) $r' \not< r$, and (iii) $\neg L \notin I\}$
- $\mathbf{PF}_{r,I}(J)$ is the greatest set $U \subseteq HB_P$ s.t.

  if $L \in U$ and r' is a rule with $Head_{r'} = L$ then $r' \not< r$ and $\exists K \in Body_{r'}$ s.t. $K \in U$ or $\sim K \in J$.
- $\mathbf{PW}_{r,I}(J) = coh(\mathbf{PT}_{r,I}(J) \cup \sim \mathbf{PF}_{r,I}(J))$.
- $\text{Pos}_{r,I}$ is the least fixpoint of the operator $\mathbf{PW}_{r,I}$.

Definition 3.5.2 (dependency set): Let P be a *PEP*, I a set of literals and $r \in R_P$. The *dependency set* of a literal K w.r.t. r and I, denoted by $\text{Dep}_{r,I}(K)$, is the least set $\text{D}(K)$ such that if $K \in I$ then $\text{D}(K)=\{\}$ else

(i) If K is a classical literal then

if r':: $L \leftarrow L_1,...,L_n$ in R_P s.t. $r' \not< r$ and $\{L_1,...,L_n\} \subseteq \text{Pos}_{r,I}$ then $\{L\} \cup \text{D}(L_i) \subseteq \text{D}(L)$, $\forall i \leq n$.

(ii) If $K = \sim L$ is a default literal then

(a) $\{\sim L\} \cup \text{D}(\neg L) \subseteq \text{D}(\sim L)$ and

(b) if for every rule r' with $Head_{r'} = L$, it holds that $r' \not< r$ and $\exists K \in Body_r$ s.t. $\sim K \in \text{Pos}_{r,I}$ then $\forall\ L \leftarrow L_1,...,L_n$ in R_P, if $\sim L_i \in \text{Pos}_{r,I}$ for $i \leq n$ then $\text{D}(\sim L_i) \subseteq \text{D}(L)$.

Definition 3.5.3 (reliable literal): Let P be a *PEP*, I a set of literals and r a rule in P.

- The literal L is *unreliable* w.r.t. r and I iff $\exists\ \neg K \in \text{Pos}_{r,I}$ and $L \in \text{Dep}_{r,I}(K)$.
- The literal L is *reliable* w.r.t. r and I iff L is not *unreliable* w.r.t. r and I.

The definition of an *m-model* of a *PEP* is the same as that of an *r*-model of an *EPP* (given in Definitions 2.2.9 and 2.2.10).

Definition 3.5.4 (W_P operator): Let P be a *PEP* and J a set of literals. We define:

- $\text{T}(J)=\{L \mid \exists r$:: $L \leftarrow L_1,...,L_n$ in P s.t. $L_i \in J$, $\forall i \leq n$ and L is reliable w.r.t. r and $J\}$
- $\text{F}(J)$ is the greatest literal set $U \subseteq HB_P$ s.t. if $L \in U$ and r is a rule with $Head_r = L$ then

  (i) $\exists K \in Body_r$ s.t. $K \in U$ or $\sim K \in J$ and (ii) $\sim L$ is reliable w.r.t. r and J.
- $\text{W}_P(J) = coh(\text{T}(J) \cup \sim \text{F}(J))$.

Definition 3.5.5 (transformation $P/_m I$): Let P be an *PEP* and I be an interpretation of it. The program $P/_m I$ is obtained as follows:

(i) Remove from P all rules that contain in their body a default literal $\sim L$ s.t. $I(L)=1$.

(ii) Remove from P any rule r with $I(\neg Head_r)=1$.

(iii) If r is a rule in P s.t. $I(Body_r)=1$ and $I(Head_r)=1/2$ then replace r with $Head_r \leftarrow u$.

(iv) Remove from the body of the remaining rules of P any default literal $\sim L$ s.t. $I(L)=0$.

(v) Replace all remaining default literals $\sim L$ with u.

(vi) If $I(L)=1/2$ and $\exists$ rule r s.t. $Head_r=L$ and $\sim L$ is unreliable w.r.t. r and I then add the rule $L \leftarrow u$.

(vii) Replace every classically negative literal $\neg A$ with a new atom $\neg\_A$.

The definition of the *reliable m-model* of a *PEP* is the same as that of the reliable *m*-model of a *PMP* (given in Definition 3.4.3). The definition of a *stable m-model* of a *PEP* is the same as that of a stable *m*-model of a *PMP* (given in Definition 3.4.5).

3.5.2 Relationship with the Modular Reliable Semantics of Modular Logic Programs

Under certain conditions, a *PMP* can be translated into a *PEP* with equivalent modular reliable semantics.

Proposition 3.5.1: Let $P = \langle Mod_P, <_{def} \rangle$ be a *PMP* s.t. every indexed literal appearing in P is globally indexed and let R_M denote the set of rules in P. Let $tr_E(P)=\langle R_E, <_R \rangle$ be a *PEP* defined as follows:

– $R_M \subseteq R_E$.

– If $M\#L \in HB_P$ and M exports $pred_L$ then add r_{M_L}:: $S_L: L \leftarrow M\#L$ to R_E.

– $r_{M\_L} <_R r_{M'\_L'}$ iff $(M, pred_L) <_{def} (M', pred_{L'})$ and $r_{M\_L} <_R r$, for each $r_{M\_L} \in R_E$ and $r \in R_M$.

Then, I is a stable *m*-model (resp. reliable *m*-model) of P iff I is a stable *m*-model (resp. reliable *m*-model) of $tr_E(P)$.

Proof: Let $P_E = tr_E(P)$. Operators with index P_E (resp. P) operate on the program P_E (resp. P). Let $W^{\uparrow b}{}_P(\emptyset) = \cup\{W^{\uparrow a}{}_P(\emptyset) \mid a<b\}$ when b is a limit ordinal and $I_a = W^{\uparrow a}{}_P(\emptyset)$ for any ordinal a.

Similarly, let $\mathbf{W}^{\uparrow b}{}_{P_E}(\emptyset)= \cup\{\mathbf{W}^{\uparrow a}{}_{P_E}(\emptyset)|\ a<b\}$ when b is a limit ordinal and $I'_a=\mathbf{W}^{\uparrow a}{}_{P_E}(\emptyset)$ for any ordinal a. We will prove by induction that $I_a=I'_a$, for any ordinal a. It is true that $a=0$. Suppose that it is true $\forall a<b$. If b is a limit ordinal then hypothesis is obviously true. Assume therefore that $b=a+1$. Let $L\in \mathbf{T}_P(I_a)$. If L is the local classical literal $M\#L'$ then $\exists\ r::\ M\#L'\leftarrow L_1,...,L_n$ in R_M s.t. $\forall i\leq n$, $L_i\in I_a$. Since modules are internally consistent and $r_{M\_L}<_R r$ for each $r_{M\_L}\in R_E$ and $r\in R_M$, it follows that $M\#L'$ is reliable w.r.t. r and I'_a. Consequently, $M\#L'\in \mathbf{T}_{P_E}(I'_a)$. If L is the globally indexed classical literal $S:L'$ then $\exists M\#L'\in I_a$ s.t. $S:L'$ is reliable w.r.t. $(M,pred_{L'})$ and I_a. This implies that $\exists\ r_{M\_L}::\ S:L'\leftarrow M\#L'$ s.t. $M\#L'\in I_a$. Moreover, since $S:L'$ is reliable w.r.t. $(M,pred_{L'})$ and I_a in P and the priorities of the rules r_{M_L} in P_E correspond to the priorities of the definitions $(M,pred_L)$ in P, $S:L'$ is reliable w.r.t. r_{M_L} and I'_a in P_E. Consequently, $S:L'\in \mathbf{T}_{P_E}(I'_a)$.

Let $L\in \mathbf{T}_{P_E}(I'_a)$. If L is a local classical literal $M\#L'$ then $\exists\ M\#L'\leftarrow L_1,...,L_n$ in P_E and thus, in P s.t. $\forall i\leq n$, $L_i\in I'_a$. This implies that $M\#L'\in \mathbf{T}_P(I_a)$. If L is a globally indexed classical literal $S:L'$ then $\exists\ r_{M\_L}::\ S:L'\leftarrow M\#L'$ s.t. $M\#L'\in I'_a$ and $S:L'$ is reliable w.r.t. r_{M_L} and I'_a in P_E. This implies that $\exists M\#L'\in I_a$ s.t. $S:L'$ is reliable w.r.t. $(M,pred_{L'})$ and I_a in P. Consequently, $S:L'\in \mathbf{T}_P(I_a)$. Thus, $\mathbf{T}_{P_E}(I'_a)=\mathbf{T}_P(I_a)$.

If $L\in I_b$ is the default literal $\sim L'$ and $\neg L'\in \mathbf{T}_P(I_a)$ then $\neg L'\in \mathbf{T}_{P_E}(I'_a)$ and consequently, $\sim L'\in I'_b$. If $L\in I'_b$ is the default literal $\sim L'$ and $\neg L'\in \mathbf{T}_{P_E}(I'_a)$ then $\neg L'\in \mathbf{T}_P(I_a)$ and consequently, $\sim L'\in I_b$.

If a local literal $M\#L'\in \mathbf{F}_P(I_a)$ then $\sim M\#L'$ is reliable w.r.t. any rule r in P_E with $Head_r=M\#L'$ because modules are internally consistent and $r_{M\_L}<_R r$ for each $r_{M\_L}\in R_E$ and $r\in R_M$. If an indexed literal $S:L'\in \mathbf{F}_P(I_a)$ then $\sim S:L'$ is reliable w.r.t. $(M,pred_{L'})$ and I_a, for any $M\in S$ and since the priorities of the rules r_{M_L} in P_E correspond to the priorities of the definitions $(M,pred_L)$ in P, $\sim S:L'$ is reliable w.r.t. r and I'_a, for any r in P_E with $Head_r=S:L'$. Thus, $S:L'\in \mathbf{F}_{P_E}(I'_a)$. If $S:L'\in \mathbf{F}_{P_E}(I'_a)$ then $\sim S:L'$ is reliable w.r.t. r and I'_a, for any r in P_E with $Head_r=S:L'$. Since $S:L'\leftarrow M\#L'$, $\forall M\in S$

are rules in P_E, $S{:}L'$ is reliable w.r.t. $(M,pred_L)$ and I_a, $\forall M \in S$. Thus, $S{:}L' \in F_P(I_a)$. Consequently, $F_{P_E}(I'_a)=F_P(I_a)$.

So, we have shown that $I_b=I'_b$.

Let I is a stable m-model of P. We will show that I is a stable m-model of P_E. For any local literal $M\#L$, the literals $\sim M\#L$ and $M\#L$ are reliable w.r.t. any rule in P_E with $Head_r=M\#L$. Thus, it is enough to show that if $\sim S{:}L$ (resp. $S{:}L$) is unreliable w.r.t. $(M,pred_L)$ and I in P, for an $M \in S$ then $\sim S{:}L$ (resp. $S{:}L$) is unreliable w.r.t. r_{M_L} and I in P_E. This follows from the fact that the priorities of the rules r_{M_L} in P_E correspond to the priorities of the definitions $(M,pred_L)$ in P. Similarly, we can show that if I is a stable m-model of P then I is a stable m-model of P_E. ◊

Note that the priorities of the rules r_{M_L} in $tr_E(P)$ correspond to the priorities of the definitions $(M,pred_L)$ in P. The indexed atoms of P are treated as conventional atoms in $tr_E(P)$.

We have not obtained a result similar to that of Proposition 3.5.1 when not globally indexed literals appear in P. Consider the obvious translation of P to a *PEP*, $tr'_E(P)=<R_E, <_R>$, defined as follows:

– $R_M \subseteq R_E$.

– If $M\#L \in HB_P$ and M exports $pred_L$ then add r_{M_L}:: $\{M\}{:}L \leftarrow M\#L$ to R_E.

– For each $S{:}L \in HB_P$ and $S'{:}L \in HB_P$ s.t. $S \subseteq S'$, add $r_{S\_S'\_L}$:: $S'{:}L \leftarrow S{:}L$ to R_E.

– $r_{M\_L} <_R r_{M'\_L}$ iff $(M,pred_L) <_{def} (M',pred_L)$.

– $r_{M\_L} <_R r$, for each $r_{M\_L} \in R_E$ and $r \in R_M$.

– $r_{M\_L} <_R r_{S\_S'\_L}$, for each $r_{M\_L} \in R_E$ and $r_{S\_S'\_L} \in R_E$.

The *MRS* of P and that of $tr'_E(P)$, may not be equivalent as it is shown in the following example.

Example 3.5.2: Consider the *PMP*, P:

module M_1 **exports** p	**module** M_2 **exports** p	**module** M_3
rules $M_1\#p$.	**rules** $\neg M_2\#p$.	**rules** $M_3\#r \leftarrow \{M_1\}{:}p, \neg\{M_2\}{:}p$.

and $<_{def}=\{\}$.

Then, $tr'_E(P) = < R_E, <_R >$ where:

$R_E=\{r_1:: M_1\#p.$ $r_2:: \neg M_2\#p.$ $r_3:: M_3\#r \leftarrow \{M_1\}:p, \neg\{M_2\}:p.$

$r_{M1\_p}:: \{M_1\}:p \leftarrow M_1\#p.$ $r_{M2\_\neg p}:: \neg\{M_2\}:p \leftarrow \neg M_2\#p.$

$r_{(M1)\_(M1,M2)\_p}:: \{M_1,M_2\}:p \leftarrow \{M_1\}:p.$ $r_{(M2)\_(M1,M2)\_\neg p}:: \neg\{M_1,M_2\}:p \leftarrow \neg\{M_2\}:p. \}$

and $r_{M1\_p} <_R r$ for all $r \in R_E - \{r_{M1\_p}, r_{M2\_\neg p}\}$.

The *MRS* of P is: $coh(\{M_1\#p, \neg M_2\#p, M_3\#r, \{M_1\}:p, \neg\{M_2\}:p\})$ whereas the *MRS* of $tr'_E(P)$ is: $coh(\{M_1\#p, \neg M_2\#p\})$. Note that, in the *MRS* of $tr'_E(P)$, the truth value of literals $\{M_1\}:p$, $\{M_2\}:p$, $\{M_1,M_2\}:p$ is unknown whereas in the *MRS* of P only the truth value of $\{M_1,M_2\}:p$ is unknown.

3.6 Related Work

In this section, we review related work on combining multiple deductive databases and maintaining consistency in a distributed environment.

3.6.1 Combining Multiple Deductive Databases

In [71], local databases $DB_1,...,DB_n$ are combined with a supervisory database (*DB*) in a framework based on *annotated logic*. Each literal in this approach is annotated with a subset of the names of the local databases and a truth value. For example, L:[DB_1, ***true***] (resp. L:[DB_1, ***false***]) is an annotated atom which expresses that literal L (resp. $\neg L$) is believed to be true in DB_1. A *combination axiom* expresses that the truth value of a literal L according to a set of local databases D is the least upper bound of the truth values of L according to the non-empty subsets of D. For example, if L:[DB_1,***true***] and L:[DB_2, ***false***] are true then L:[{DB_1,DB_2}, ***true***] is true. Note that

this is in contrast with *MRS*. In *MRS*, the truth value of $\{DB_1,DB_2\}:L$ is considered unknown unless the definition of $pred_L$ in one of DB_1 or DB_2 is more reliable than in the other.

The supervisory *DB* can access literals defined in the local *DBs*. Yet, local *DBs* can access only local information. The resolution of conflicts between the local *DBs* is the responsibility of the supervisory *DB*. For example, the supervisory *DB*, *S*, may contain rules such as:
L: $[S, V] \leftarrow L:[DB_1, V]$, expressing that the truth value of a literal L in S is the same as the truth value of L in DB_1 (V is a variable representing the truth value of L) or
L: $[S, glb(V_1,...,V_n)] \leftarrow L:[DB_1, V_1],...,L:[DB_n, V_n]$, expressing that the truth value of a literal L in S is the *greater lower bound* (in the *lattice of truth values*) of the truth values of L in $DB_1,...,DB_n$.

In contrast to this approach, in *MRS*, modules can import information from any other module as indicated in their definitions. Moreover, conflict resolution is incorporated in the semantics.

Work on combining deductive databases has also been done in [10]. Let $DB_1,...,DB_m$ be a set of positive logic programs to be combined in a single *DB* which should comply with a set of constraints. Even though $DB_1,...,DB_m$ are assumed to be consistent, *DB* may violate the constraints. In [10], when a constraint $\perp \leftarrow L_1,...,L_n$ is violated, the rules with head L_i, $i \leq n$, are removed from *DB* and the disjunctions

$(L_1 \vee L_2),...,(L_1 \vee L_n)$, $(L_2 \vee L_3),...,(L_2 \vee L_n)$, ..., $(L_{n-1} \vee L_n)$.

are added to *DB*. Thus, *DB* is a disjunctive logic program. Note that the formula

$(L_1 \vee L_2) \wedge ... \wedge (L_1 \vee L_n) \wedge$

$(L_2 \vee L_3) \wedge ... \wedge (L_2 \vee L_n) \wedge$

...

$(L_{n-1} \vee L_n)$

is equivalent to $(L_1 \wedge ... L_{n-1}) \vee ... \vee (L_1 \wedge ... \wedge L_{i-1} \wedge L_{i+1} \wedge ... \wedge L_n) \vee ... \vee (L_2 \wedge ... \wedge L_n)$.

The maximum information is saved with this approach since to maintain consistency, one of L_i should not be provable in *DB*. However, it is possible that literals $L_1,...,L_n$ are based on unreliable

information which will continue to be evaluated as true in [10]. In *MRS*, not only the literals L_i, $i \leq n$, are considered "suspect" for the violation of the constraint but also the literals used in the derivation of L_i, $i \leq n$.

3.6.2 A Distributed Assumption Truth Maintenance System

In [46], a problem solving framework for distributed assumption based reasoning is described. The framework, called *distributed assumption truth maintenance* (DATM) framework, is based on the problem solving paradigm of result sharing rule-based systems using assumption-based truth maintenance systems [17]. According to [46], each agent is equipped with inference rules, communications rules, facts, and an assumption truth maintenance system (ATMS) which is composed of truth maintenance rules and an assumption data base. *Truth maintenance rules* indicate the sets of facts that constitute an inconsistency and how an inconsistency finding should be propagated to the other agents. *Communication rules* specify the conditions under which local results should be shared with other agents and the list of agents that results should be transmitted to.

Each agent makes assumptions and works with its consequents until inconsistency is detected after an inference step. Each result is associated with the agent responsible for its derivation. This provides a way to trace back results, supporting global belief revision. This belief revision approach is in contrast with our approach where contradiction is avoided before it is generated.

The basic inference engine cycle for an agent is the following:

Inferencing

1. Check current set of beliefs against all rules. Decide which inference rule to fire.

2. Execute the inference rule. Record the inference.

Belief Updating and Result Sharing

3. Check all truth maintenance rules $\perp \leftarrow L_1,\ldots,L_n$.

4. If an inconsistency is detected, record the set of incompatible assumptions in a NOGOOD assumption set.
5. Determine which agents should be contacted with a message containing the NOGOOD assumption set.
6. Execute all the instantiated communications rules and accept any incoming messages.
7. Update NOGOODS database with incoming NOGOODS assumption sets. Update working memory and assumption database with incoming results.

Similarly, to a single-agent TMS (reviewed in subsection 4.4.3), a DATMS considers only one solution per time out of a set of multiple admissible solutions. This is because the main goal of a TMS is to maintain a consistent set of beliefs. In contrast, the main goal of the modular reliable semantics, presented in this Chapter, is to draw conclusions that are satisfied in all of the stable *m*-models of the combined knowledge base. Moreover, in our approach, a complete consistency check is performed before a result is exported whereas in DATMS, results are exported as indicated by the communication rules and when inconsistency is detected, global belief revision is performed.

3.7 Conclusions

We have presented the modular reliable semantics (*MRS*) of *prioritized modular logic programs* (*PMPs*). The purpose of the modular reliable semantics is to derive reliable information from contradictory *PMPs*. Every *PMP* has at least one *stable m-model*. The *reliable m-model* of a program P is the least stable m-model of P and it represents the skeptical "meaning" of P. Maximal (w.r.t. $\leq_{ss}$) stable m-models of P represent the credulous "meanings" of P. The complexity of computing *MRS* is polynomial w.r.t. the size of the program (when HB_P is finite). Since a *prioritized extended logic program* (*PEP*) is naturally translated into a *PMP*, the modular reliable

semantics for a *PEP* is also defined. In subsection 3.5.2, we proved that under certain conditions, a *PMP* can be translated into a *PEP* with equivalent modular reliable semantics.

One application of *MRS* is deriving trustworthy information after combining multiple deductive databases (*DBs*) that are not fully reliable. For example, when the *DBs* of different scientific labs are combined, conflicts may occur because of measurement errors. Each *DB* can be seen as a low-level independent module exporting results computed from local information only. On top of the local *DBs*, supervisory modules may be added for the processing of the local results. Several dependence relationships between the supervisory modules and the local *DBs* and among the supervisory modules themselves can exist. Future work should include the identification of these relationships and their representation in our framework.

In section 3.6, we compare *MRS* with related work on combining multiple deductive databases and maintaining consistency in a distributed environment.

CHAPTER 4

CONTRADICTION-FREE SEMANTICS FOR EXTENDED LOGIC PROGRAMS WITH RULE PRIORITIZATION

4.1 Introduction

A *prioritized extended program* (*PEP*) consists of a set of partially ordered rules. Every rule r has a corresponding set $C_r \subseteq Body_r$ which is called the *contrapositive set* of r. We define the *contradiction-free semantics* (*CFS*) of a *PEP*, P, by expanding it with the contrapositives r' of every rule r such that $Head_{r'} \in \{\neg L |\ L \in C_r\}$. The motivation for this expansion was given in section 1.3. *CFS* is always defined and non-contradictory. Every *PEP* has at least one *stable c-model*. We show that the *CFS* of a program P is the least fixpoint of a monotonic operator and the least stable *c*-model of P. When the Herbrand base of P is finite, the complexity of computing *CFS* is polynomial w.r.t. the size of the expansion of P. The *SLCF*-resolution (linear resolution with selection function for contradiction-free semantics) for computing answers for extended program with rule prioritization is presented. The *SLCF*-resolution is shown to be sound and complete w.r.t. *CFS*.

CFS extends the well-founded semantics for normal programs [76] to *PEPs*. The use of contrapositives for resolving contradictions in *CFS* has been supported by [28, 78]. Yet, in these works, rule prioritization is not considered, the semantics is not always defined, and $C_r = Body_r$ for every rule r. *CFS* semantics is also related to ordered logic [24, 43] (reviewed in subsection 2.4.7). Let P be an ordered logic program, i.e., P does not contain default literals and $C_r = \{\}$ for every rule r. Then, the *CFS* of P coincides with the *skeptical c-partial model* of P [24] and is a subset of the *well-founded partial model* of P [43].

4.2 *c*-models for Prioritized Extended Programs

A *prioritized extended program* (*PEP*) is a tuple $P=<R_P,<_R>$. R_P is a set of rules r: $L_0 \leftarrow L_1,...,L_m,\sim L_{m+1},...,\sim L_n$, where r is a label and L_i are classical literals. Every rule r has a corresponding set $C_r \subseteq Body_r$, called the *contrapositive set* of r. The precise meaning of C_r will be given in the definitions. Intuitively, when there is a rule r s.t. both $Body_r$ and $\neg Head_r$ are derived from P, the value of C_r indicates the "suspects" for the contradiction. When $C_r=\{\}$, the rule r is considered incomplete[1]. When $C_r \neq \{\}$, the contradiction is considered as evidence that one of the literals in C_r was wrongly derived. Thus, the *CWAs* and/or rules used in some step of the derivation of literals in C_r are considered unreliable. To facilitate this reasoning, P is expanded with the contrapositives r' of every rule r such that $Head_{r'} \in \{\neg L \mid L \in C_r\}$.

For example, consider the program $P=\{r_1: a.\ \ r_2: b.\ \ r_3: p \leftarrow a.\ \ r_4: \neg p \leftarrow b.$ with $C_{r_3}=\{\}$ and $C_{r_4}=\{\}\}$. Because both a, $\neg p$ are derived in P and $C_{r_3}=\{\}$, the rule r_3 is considered incomplete, i.e., r_3 should be $p \leftarrow a,\sim\neg p$. A similar argument applies to rule r_4. Thus, literals a, b can be reliably evaluated as true but the truth value of p is unknown. In contrast, consider the program $P'=\{r_1: a.\ \ r_2: b.\ \ r_3: p \leftarrow a.\ \ r_4: \neg p \leftarrow b.$ with $C_{r_3}=\{a\}$ and $C_{r_4}=\{b\}\}$. Since $C_{r_3}=\{a\}$, the rule r_1 used for the derivation of a is considered unreliable. Similarly, the rule r_2 used for the derivation of b is considered unreliable. By expanding P' with the contrapositives $r'_3: \neg a \leftarrow \neg p$ and $r'_4: \neg b \leftarrow p$, the derivation of a, b from rules r_1, r_2 is blocked and the literals a, b, p are evaluated as unknown. The view $C_r=\{\}$ for every rule r is implicit in ordered logic [24, 43] and vivid logic [77]. The view $C_r = Body_r$ for every rule r is adopted in [28, 78]. Yet, other views such as $C_r \neq \{\}$ and $C_r \neq Body_r$ for a rule r are also possible.

The relation $<_R \subseteq R_P \times R_P$ is a strict partial order (irreflexive, asymmetric and transitive), denoting the relative reliability of the rules. Let r and r' be two rules. The notation $r<r'$ means that r is less reliable than r', that is, $r<r'$ iff $(r,r') \in <_R$. The notation $r \not< r'$ means that r is not less reliable that r'. Note that, $r \not< r$ since $<_R$ is irreflexive. Intuitively, when $Body_r$ is true, $Head_r$ is

---

[1] We say that a rule is *incomplete* if not all possible exceptions are enumerated in its body.

evaluated as true iff $\neg Head_r$ cannot be derived from rules with priority no lower than r. Thus, deciding if $Head_r$ is true depends only on the rules $r' \not< r$. Note that a *PEP* with $<_R=\{\}$ is an extended logic program. In all sections but section 4.5, we assume that programs have been instantiated and thus all rules are propositional. For every classical literal L, $\neg(\sim L)=L$.

Definition 4.2.1 (program expansion): Let P be a *PEP*. The *expansion* $exp(P)$ of P is also a *PEP* defined as follows:

- For every rule $r: H \leftarrow L_1,\ldots,L_n$ of P, $exp(P)$ contains the rules $\{r'_i: \neg L_i \leftarrow L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,\neg H \mid L_i \in C_r$ and $C_{r'_i}=(C_r-\{L_i\})\cup\{\neg H\}\}$ (called *contrapositives* of r) and the rule r.
- The partial ordering of the rules of P is extended to the rules of $exp(P)$ as follows: If r and r' are two rules of P with $r<r'$ (resp. $r \not< r'$) then r and any contrapositive of r has less (resp. neither less nor more) priority than r' and any contrapositive of r'. If r and r' are contrapositives then $r \not< r'$.

Note that $exp(exp(P))=exp(P)$.

Definition 4.2.2 (interpretation): Let P be a *PEP*. A set $I=T\cup\sim F$ is an interpretation of P iff T and F are disjoint subsets of HB_P. An interpretation I is *consistent* iff there is no L such that both $L \in T$ and $\neg L \in T$. An interpretation I is *coherent* iff it satisfies the *coherence property*: if $L \in T$ then $\neg L \in F$.

Definition 4.2.3 (truth valuation of a literal): A literal L is true (resp. false) w.r.t. an interpretation I iff $L \in I$ (resp. $\sim L \in I$). A literal that is neither true nor false w.r.t. I, it is undefined w.r.t. I.

In Definition 4.2.4, the concept of *c*-unfounded classical literal w.r.t. a rule r and interpretation I is defined. I represents a set of literals known to be true. This concept is used in the fixpoint computation of *CFS*. In particular, a rule r is used for the derivation of $Head_r$ only if $\neg Head_r$ is *c*-unfounded w.r.t. r and *CFS*. Intuitively, a literal L is *c*-unfounded w.r.t. r and I if L cannot be derived from the rules $r' \not< r$ when literals are assumed to be false as indicated in I.

Definition 4.2.4 (*c*-unfounded literal w.r.t. *r* and *I*): Let P be a *PEP*, r a rule and I an interpretation. A classical literal L is called *c*-unfounded w.r.t. r and I iff there is a classical literal set S s.t. $L \in S$ and $\forall H \in S$, if r' is a rule in $exp(P)$ s.t. $Head_{r'} = H$ and $r' \not< r$ then (i) $I(Body_{r'})=0$ or (ii) there is a classical literal $L' \in C_{r'}$ s.t. $L' \in S$ or (iii) there is a default literal $\sim L' \in C_{r'}$ s.t. $\neg L' \in S$.

Note that if a literal L is *c*-unfounded w.r.t. r and I then L is *c*-unfounded w.r.t. r and any interpretation $I' \supseteq I$. If a rule r is unidirectional ($C_r=\{\}$) and $I(Body_r)\neq 0$ then $Head_r$ is not *c*-unfounded w.r.t. any rule r' s.t. $r \not< r'$. Intuitively, when $C_r = Body_r$ $\forall$rule r, every rule is given higher priority than the *CWAs*. In Example 4.2.1, we show that this is not true when there is a literal $L \in Body_r - C_r$ for a rule r. An algorithm that decides if a literal L is *c*-unfounded w.r.t. r and I is given in Appendix A. The time-complexity of the algorithm is linear w.r.t. the size of $exp(P)$.

Example 4.2.1: Let P be the expanded (with contrapositives) *PEP*:

$R_P = \{r_1$: *fly*. r_2: $\neg$*fly* $\leftarrow \sim$*bird*. r'_2: *bird* $\leftarrow$ *fly*. with $C_{r_2}=\{\sim bird\}$, $C_{r'_2}=\{fly\}\}$ and $<_R = \{\}$.

Then, the literal $\neg$*fly* is *c*-unfounded w.r.t. r_1 and $\emptyset$ (in Def. 4.2.4 take $S=\{\neg fly, \neg bird\}$). This implies that *fly* can be reliably derived from rule r_1. Intuitively, in this case, rule r_1 is given higher priority than the *CWA*, $\sim$*bird*. However, this is not the case if $C_{r_2}=\{\}$. Consider the program P':

$R_{P'} = \{r_1$: *fly*. r_2: $\neg$*fly* $\leftarrow \sim$*bird*. with $C_{r_2}=\{\}\}$ and $<_R = \{\}$.

Then, the literal $\neg$*fly* is not *c*-unfounded w.r.t. r_1 and $\emptyset$ since there is no S to satisfy conditions in Def. 4.2.4. This, intuitively, implies that r_1 is blocked and *fly* is evaluated as unknown.

Example 4.2.2: (credit confusion problem) Consider the following expanded *PEP*, $P=<R_P, <_R>$:

R_P={ /* If Ann is a foreign student (resp. teaching assistant) then she needs 12 (resp. 6) credits */

r_1: need_credits(ann,12)$\leftarrow$foreign_stud(ann). r_2: need_credits(ann,6)$\leftarrow$TA(ann).

r_3: TA(ann). r_4: foreign_stud(ann).

r_5:$\neg$need_credits(ann,6)$\leftarrow$need_credits(ann,12).

r'_5:$\neg$need_credits(ann,12)$\leftarrow$need_credits(ann,6).

with $C_{r_i}=\{\}$ for i=1,2,3,4, C_{r_5}={need_credits(ann,12)} and $C_{r'_5}$={need_credits(ann,6)}}

and $r_1<r_5$, $r_2<r_5$, $r_3<r_5$, $r_4<r_5$, $r_1<r'_5$, $r_2<r'_5$, $r_3<r'_5$, $r_4<r'_5$ and $r_1<r_2$.

/* Rules r_5 and r'_5 have higher priority than the other rules */

The literal ¬TA(ann) is *c*-unfounded w.r.t. r_3 and Ø (in Def. 4.2.4 take S={¬TA(ann)}). So, TA(ann) can be reliably derived from rule r_3. Similarly, foreign_stud(ann) can be reliably derived from rule r_4. Since $r_1<r_2$, the literal ¬need_credits(ann,6) is *c*-unfounded w.r.t. r_2 and Ø (in Def. 4.2.4 take S={¬need_credits(ann,6), need_credits(ann,12)}). So, need_credits(ann,6) can be reliably derived from rule r_2. In contrast, ¬need_credits(ann,12) is not *c*-unfounded w.r.t. r_1 and Ø. However, if P' is as P with $<_R$={} then ¬need_credits(ann,6) is not *c*-unfounded w.r.t. r_2 and Ø in P'.

Definition 4.2.5 (truth valuation of a rule): Let P be a *PEP*. A rule r in $exp(P)$ is *c-true* w.r.t. an interpretation I iff: (i) $I(Head_r)\geq I(Body_r)$ or (ii) $I(Body_r)=1/2$ and $I(\neg Head_r)=1$ or (iii) $I(Body_r)=1$ and ($I(Head_r)=1/2$ or $I(\neg Head_r)=1$) and $\neg Head_r$ is not *c*-unfounded w.r.t. r and I.

Definition 4.2.6 (*c*-model): Let P be a *PEP*. A consistent, coherent interpretation I of P is a *c-model* of P iff every rule in $exp(P)$ is *c*-true w.r.t. I.

Example 4.2.3: Let P be as in Example 4.2.1 and M be a *c*-model of P. Then, $fly\in M$ because r_1 should be *c*-true w.r.t. M and $\neg fly$ is *c*-unfounded w.r.t. r_1 and $M\supseteq\emptyset$. In contrast, *fly* is not true in all models of P' of Example 4.2.1. The *c*-models of P' are M_1={$\sim bird$}, $M_2=coh(\{fly,\sim bird\})$ and $M_3=coh(\{\neg fly,\sim bird\})$.

Example 4.2.4: Let P be as in Example 4.2.2. Then, $M=coh$({TA(ann), foreign_stud(ann), need_credits(ann,6), ¬need_credits(ann,12)}) is a *c*-model of P. We will show that M is the unique *c*-model of P. Let M' be a *c*-model of P. Then, ¬TA(ann), ¬foreign_stud(ann), ¬need_credits(ann,6), need_credits(ann,12) are *c*-unfounded w.r.t. $M\supseteq\emptyset$ and rules r_3, r_4, r_2 and r'_5, respectively. Thus, $M\subseteq M'$. The literal need_credits(ann,12)$\notin M'$ because otherwise ¬need_credits(ann,6)$\in M'$ (need_credits(ann,6) is *c*-unfounded w.r.t. r_5 and $M\supseteq\emptyset$) and thus, M' is contradictory.

Let P be a normal program and I an interpretation as defined in [58, 60]. In [60], a rule r is true w.r.t. I iff $I(Head_r) \geq I(Body_r)$. Since P is a normal program, the heads of the rules are atoms. Consequently, the bodies of all contrapositives of a rule r in P contain the classically negative literal $\neg Head_r$. This implies that all classically negative literals are c-unfounded w.r.t. any rule and I. If $I' = I \cup \{\sim\neg A \mid A \text{ is an atom of } P\}$ then conditions (ii) and (iii) in Def. 4.2.5 are not satisfied by I', for all rules in P. This implies that a rule r in P is c-true w.r.t. I' iff r is true w.r.t. I.

Proposition 4.2.1: Let P be a normal program. M is a model of P iff $M \cup \{\sim\neg A \mid A \text{ is an atom of } P\}$ is a c-model of P, independently of the values of C_r.

4.3 Contradiction-Free Semantics

In this section, we define the *contradiction-free model*, *stable c-models* and *contradiction-free semantics* of a *PEP*, P. We define the contradiction-free model of P as the least fixpoint of a monotonic operator and we show that it is the least stable c-model of P.

Definition 4.3.1 (W_P operator): Let P be a *PEP* and J a set of literals. We define:

- $T_J(T) = \{L \mid \exists r: L \leftarrow L_1, \ldots, L_n \text{ in } exp(P) \text{ s.t. } L_i \in T \cup J, \forall i \leq n \text{ and } \neg L \text{ is } c\text{-unfounded w.r.t. } r \text{ and } J\}$.
- $T(J) = \cup\{T_J \uparrow a(\emptyset) \mid a < \omega\}$, where ω is the first limit ordinal.
- $F(J)$ is the greatest set of classical literals S s.t. $\forall L \in S$, if r is a rule in $exp(P)$ with $Head_r = L$ then $J(Body_r) = 0$ or $\exists L' \in Body_r$ s.t. $L' \in S$.
- $W_P(J) = coh(T(J) \cup \sim F(J))$.

When $\neg Head_r$ is not c-unfounded w.r.t. r and J, we say that r is blocked w.r.t. J. Note that the sequence $\{T_J \uparrow a\}$ is monotonically increasing (w.r.t. $\subseteq$). So, $T(J)$ is the least fixpoint of T_J. We define the transfinite sequence: $I_0 = \{\}$, $I_{a+1} = W_P(I_a)$ and $I_a = \cup\{I_b \mid b < a\}$ if a is a limit ordinal.

Proposition 4.3.1: Let P be a *PEP*. $\{I_a\}$ is a monotonically increasing (w.r.t. $\subseteq$) sequence of consistent, coherent interpretations of P.

Proof: Given in Appendix A. ◊

Since $\{I_\alpha\}$ is monotonically increasing (w.r.t. $\subseteq$), there is a smallest countable ordinal d s.t. $I_d=I_{d+1}$.

Proposition 4.3.2: Let P be a *PEP*. Then, I_d is a *c*-model of P.

Proof: Given in Appendix A. ◊

Definition 4.3.2 (contradiction-free semantics): Let P be a *PEP*. The *contradiction-free model* of P (CFM_P) is the *c*-model I_d. The *contradiction-free semantics* (*CFS*) of P is the "meaning" represented by CFM_P.

In Example 4.3.1, we show that contrapositives are necessary in order to avoid the derivation of complementary literals in $\{I_\alpha\}$.

Example 4.3.1: Consider the *PEP*, $P=<R_P, <_R>$:

$R_P=\{r_1$: *OK_M*. r_2: ¬*rings*. r_3: *rings*← *OK_M*. with $C_{r_3}=Body_{r_3}\}$ and $r_1<r_2<r_3$.

Rule r_3 expresses that if machine M is OK then it rings. Rule r_2 expresses the observation that machine M does not ring. Rule r_1 expresses the assumption that machine M is OK.

The program $exp(P)$ is as follows:

$R_{exp(P)}=\{r_1$: *OK_M*. r_2: ¬*rings*. r_3: *rings*← *OK_M*. r'_3: ¬*OK_M* ← ¬*rings*.

with $C_{r_3}=Body_{r_3}$ and $C_{r'_3}=Body_{r'_3}\}$ and $r_1<r_2<r_3$, $r_1<r_2<r'_3$.

Since *rings* is *c*-unfounded w.r.t. r_2 and Ø, ¬*rings*∈ T(Ø). Since, the literal ¬*OK_M* is not *c*-unfounded w.r.t. r_1 and Ø, rule r_1 is blocked w.r.t. Ø. Since *OK_M* is *c*-unfounded w.r.t. r'_3 and Ø, ¬*OK_M*∈ T(Ø) (derived from r'_3). So, T(Ø)={¬*rings*, ¬*OK_M*}, F(Ø)={} and $W_P(\emptyset)=coh(\{\neg rings, \neg OK\_M\})$. Because $W_P{\uparrow}^2(\emptyset)=W_P(\emptyset)$, it follows that $CFM_P = coh(\{\neg rings, \neg OK\_M\})$.

We will show that contrapositives are necessary in order to avoid the derivation of complementary literals. Assume that $exp(P)=P$. Then, ¬*OK_M* is *c*-unfounded w.r.t. r_1 and Ø and thus *OK_M*∈ $W_P(\emptyset)$. Consequently, *rings* is derived from r_3, since ¬*rings* is *c*-unfounded w.r.t. r_3

and Ø. However, $\neg rings \in W_P(\emptyset)$ (derived from r_2), since *rings* is *c*-unfounded w.r.t. r_2 and Ø. So, $W_P(\emptyset)=coh(\{OK\_M, rings, \neg rings\})$ which is inconsistent.

Let P' be as P with the additional rule r_4: *rings*. Let $r_3 < r_4$, expressing that r_4 is a more reliable observation than r_3. Then, $W_{P'}(\emptyset)=coh(\{rings\})$ where *rings* is derived from rule r_4. Note that $\neg OK\_M$ is not *c*-unfounded w.r.t. r_1 and Ø in P' and thus r_1 is blocked w.r.t. Ø. In contrast, $\neg OK\_M$ is *c*-unfounded w.r.t. r_1 and $W_{P'}(\emptyset)$ because $Body_{r'_3}= \{\neg rings\}$ is false w.r.t. $W_{P'}(\emptyset)$. Thus, $CFM_{P'} = W_{P'}{}^{\uparrow 2}(\emptyset)=coh(\{OK\_M, rings\})$.

Example 4.3.2: Let P be the program of Example 4.2.2. Then, CFM_P= *coh*({TA(ann), foreign_stud(ann), need_credits(ann,6), ¬need_credits(12)}). If P' is as P with $<_R=\{\}$ then $CFM_{P'}$ =*coh*({TA(ann), foreign_stud(ann)}) which corresponds to the skeptical meaning of P'. If P' is as P with $C_r = Body_r$ ∀ rule r then ¬TA(ann) (resp. ¬foreign_stud(ann)) is not *c*-unfounded w.r.t. r_3 (resp. r_4) and Ø because of the contrapositive of r_2 (resp. r_1) in $exp(P)$ Thus, $CFM_{P'}=\{\}$.

Proposition 4.3.3: Let P be a *PEP*. The complexity of computing CFM_P is $O(|HB_P|^*|exp(P)|^2)$.

Proof: Given in Appendix A. ◊

An algorithm for computing CFM_P is given in Appendix A. The contradiction-free model of a *PEP* corresponds to the skeptical meaning of the program. Other meanings can be obtained using the transformation $P/_cI$, where I is an interpretation of P. The transformation P/I is defined in [26, 60] for a normal program P. $P/_cI$ extends P/I to *PEPs*.

Definition 4.3.3 (transformation $P/_cI$): Let P be an expanded *PEP* and I be an interpretation of it. The program $P/_cI$ is obtained as follows:

(i) Remove from P all rules that contain in their body a default literal $\sim L$ s.t. $I(L)=1$.

(ii) Remove from P any rule r with $I(\neg Head_r)=1$.

(iii) If r is a rule in P s.t. $I(Body_r)=1$ and $I(Head_r)=1/2$ then replace r with $Head_r \leftarrow u$.

(iv) Remove from the body of the remaining rules of P any default literal $\sim L$ s.t. $I(L)=0$.

(v) Replace all remaining default literals $\sim L$ with u.

(vi) Replace every classically negative literal $\neg A$ with a new atom $\neg\_A$.

Example 4.3.5: Let P be as in Example 4.2.2 and M be as in Example 4.2.4. Then

$P/_c I$ ={need_credits(6)←TA(mary). TA(mary). foreign_stud(mary).

¬need_credits(12)←need_credits(6). }

The program $P/_c I$ is a non-negative program with a special proposition u. For any interpretation J, $J(u)$=1/2. When P is a normal program and M is a model of P [60], $P/_c M \equiv P/M$ since Steps (ii), (iii), and (vi) do not have any effect on $P/_c M$.

Definition 4.3.4 (stable *c*-model): Let P be a *PEP* and M a *c*-model of P. M is a *stable c-model* of P iff $least_v(exp(P)/_c M)=M$.

Example 4.3.6: Let P' be as P in Example 4.2.2 with $<_R=\{\}$. Let

$M_1=coh(\{TA(ann), foreign\_stud(ann), need\_credits(ann,6), \neg need\_credits(12)\})$ and

$M_2=coh(\{TA(ann), foreign\_stud(ann), need\_credits(ann,12), \neg need\_credits(6)\})$.

Then, M_1 and M_2 are stable *c*-models of P'.

The program P of Example 4.2.2 has a unique stable *c*-model equal to CFM_P.

Let M be a stable *c*-model of P. Changing the ordering of the rules in P, the condition $least_v(exp(P)/_c M)=M$ will still be satisfied but M may not be a *c*-model of the new program. For example, let P be as in Example 4.2.2 and M be as in Example 4.2.4. Then, M is a stable *c*-model of P. If we replace $r_1<r_2$ in P with $r_2<r_1$, the condition $least_v(\exp(P)/_c M)=M$ is still satisfied. However, M is not a *c*-model of the new program P' because ¬need_credits(12) becomes *c*-unfounded w.r.t. r_1 and M and thus, r_1 is not *c*-true w.r.t. M in P'.

Proposition 4.3.4: Let P be a *PEP*. Then, CFM_P is a stable *c*-model of P.

Proof: Given in Appendix A. ◊

Proposition 4.3.5: Let P be a *PEP*. Then, CFM_P is the least stable *c*-model of P.

Proof: Given in Appendix A. ◊

4.4 Related Work

In this Section, we relate the contradiction-free semantics with existing work.

4.4.1 Semantics Covered in Section 2.4

The contradiction-free semantics for *PEPs* is a generalization of the 3-*valued stable model semantics* which is defined for normal programs [60].

Proposition 4.4.1: Let P be a normal program. Then, M is a 3-valued stable model of P iff $M \cup \{\sim\neg A \mid A$ is an atom of $P\}$ is a stable c-model of P, independently of the values of C_r in P.

Proof: $\Rightarrow$) Let M be a 3-valued stable model of P. It is easy to verify that $M' = M \cup \{\sim\neg A \mid A$ is an atom of $P\}$ is a c-model of P. Let $r: A_0 \leftarrow A_1,\ldots,A_m,\sim A_{m+1},\ldots,\sim A_n$ be a rule in P where A_i, $i \in \{0,\ldots,n\}$ are atoms. Since M is a 3-valued stable model of P, $least_v(P/M)=M$. Because the heads of all rules in P are atoms, every rule r in $exp(P)$ with $Head_r = \neg A$ has a literal $\neg B$ in the body, where A,B are atoms. This implies that every literal $\neg A$, where A is an atom, is c-unfounded w.r.t. any rule r and M. Moreover, all literals $\neg A$, where A is an atom, are false w.r.t. $least_v(exp(P)/_c M')$ and every contrapositive of a rule in P has a false literal in its body w.r.t. $least_v(exp(P)/_c M')$. So, $least_v(exp(P)/_c M') = least_v(P/M) \cup \{\sim\neg A \mid A$ is an atom$\} = M \cup \{\sim\neg A \mid A$ is an atom$\} = M'$. So, $M \cup \{\sim\neg A \mid A$ is an atom of $P\}$ is a stable c-model of P.

$\Leftarrow$) The proof is similar to the proof of $\Rightarrow$. $\Diamond$

Proposition 4.4.1 implies that the contradiction-free model of a normal program P coincides with the well-founded model of P [76].

The following relationship between *CFS* and the answer set semantics [27] can be shown.

Proposition 4.4.2: Let P be an extended program with $C_r = \{\}$ $\forall$ rule r. If $M \neq HB_P$ is an answer-set [27] of P then $M \cup \{\sim A \mid A \notin M\}$ is a stable c-model of P.

Proof: P is non-contradictory since $M \neq HB_P$ is an answer-set of P. So, if M is an answer-set of P then $M \cup \{\sim A \mid A \notin M\}$ is an extended stable model of P [54]. Proposition 4.4.2 now follows from Proposition 4.4.3. ◊

The relationship between *CFS* and extended well-founded semantics [54] is given in the next two propositions.

Proposition 4.4.3: Let P be a non-contradictory extended program with $C_r=\{\}$ ∀ rule r. If M is an extended stable model [54] of P then M is a stable c-model of P.

Proof: Since $C_r=\{\}$ ∀ rule r, $exp(P)=P$. Let M be an extended stable model of P. From the definition of extended stable model [54], M is a c-model of P and $least_v(P/_c M)=M$. So, M is a stable c-model of P. ◊

The reverse of Proposition 4.4.3 does not hold. For example, consider $P=\{r_1: p \leftarrow \sim p.\ r_2: a \leftarrow p.\quad r_3: \neg a.$ with $C_r=\{\}$ ∀ rule $r\}$. Then, {} is a stable c-model of P since a is not c-unfounded w.r.t. r_2 and Ø. However, {} is not an extended stable model of P.

Proposition 4.4.4: Let P be an extended program with $C_r=Body_r$ ∀ rule r. If $XWFS_{exp(P)}$ is defined then it coincides with CFS_P.

Proof: Assume that $XWFS_{exp(P)}$ is defined. We will show by induction that $\mathbf{W}_P{}^{\uparrow a}(\emptyset) = \Phi_{exp(P)}{}^{\uparrow a}(\emptyset)$, for all a (the operator Φ_P is defined in subsection 2.4.3). This is true when $a=0$. Suppose that it is true for all ordinals $\leq a$. We will show that $\mathbf{W}_P{}^{\uparrow a+1}(\emptyset) = \Phi_{exp(P)}{}^{\uparrow a+1}(\emptyset)$. It is enough to show that for each rule r in $exp(P)$, if $Body_r \subseteq \Phi_{exp(P)}{}^{\uparrow a}(\emptyset)$. Then, $Body_r \subseteq \mathbf{W}_P{}^{\uparrow a}(\emptyset)$ and $\neg Head_r$ is c-unfounded w.r.t. r and $\mathbf{W}_P{}^{\uparrow a}(\emptyset)$. Let r be a rule in $exp(P)$ with $Body_r \subseteq \Phi_{exp(P)}{}^{\uparrow a}(\emptyset)$ then from hypothesis, $Body_r \subseteq \mathbf{W}_P{}^{\uparrow a}(\emptyset)$. Since $XWFS_{exp(P)}$ is defined, $\Phi_{exp(P)}{}^{\uparrow a+1}(\emptyset)$ is consistent. Since $C_r=Body_r$ ∀ rule r and $<_R=\{\}$, every classical literal not in $\Phi_{exp(P)}{}^{\uparrow a+1}(\emptyset)$ is c-unfounded w.r.t. any rule and $\Phi_{exp(P)}{}^{\uparrow a}(\emptyset)$. Since $Head_r \in \Phi_{exp(P)}{}^{\uparrow a+1}(\emptyset)$, it follows that $\neg Head_r \notin \Phi_{exp(P)}{}^{\uparrow a+1}(\emptyset)$ and thus, $\neg Head_r$ is c-unfounded w.r.t. r and $\mathbf{W}_P{}^{\uparrow a}(\emptyset)$. ◊

The above proposition is not true when $C_r \neq Body_r$ for a rule r. For example consider the program $P = \{r_1: p \leftarrow \sim p. \quad r_2: a \leftarrow p. \quad r_2: \neg a.$ with $C_r = \{\} \ \forall$ rule $r\}$. Then, $CFS_P = \{\}$ whereas the $XWFS_P$ is $\{\neg a, \sim a, \sim\neg p\}$.

The relationship between *CFS* and ordered logic [24, 43] is given in the next two propositions.

Proposition 4.4.5: Let $P = \langle R_P, <_R \rangle$ be a *PEP* which is free from default literals and $C_r = \{\} \ \forall$ rule r. Then, the set of classical literals in CFM_P coincides with the skeptical c-partial model of P [24].

Proof: To simplify the proof, we redefine the operator $T(J)$ of Def. 4.3.1 as follows: $T(J) = \{L \mid \exists$ rule r in P s.t. $Head_r = L$, $Body_r \subseteq J$ and $\neg L$ is c-unfounded w.r.t. r and $J\}$. Note that both definitions give equivalent semantics. Let $I_a = W_P{}^{\uparrow a}(\emptyset)$, for all a.

We will show by induction that the set of classical literals in I_a is a subset of $S^{\uparrow a}(\emptyset)$, for all a. This is true when $a=0$. Suppose that it is true for all ordinals $\leq a$. We will show that the set of classical literals in I_{a+1} is a subset of $S^{\uparrow a+1}(\emptyset)$. Since $S(I) = \{L \mid \exists$ rule r s.t. $Head_r = L$, $Body_r \subseteq I$ and r is not c-defeasible w.r.t. $I\}$, it is enough to show that for each rule r, if $Body_r \subseteq I_a$ and $\neg Head_r$ is c-unfounded w.r.t. r and I_a then $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and r is not c-defeasible w.r.t. $S^{\uparrow a}(\emptyset)$.

$Body_r \subseteq I_a$ and $\neg Head_r$ is c-unfounded w.r.t. r and I_a

(From the inductive hypothesis and the fact that $Body_r$ is free of default literals, it follows that $Body_r \subseteq S^{\uparrow a}(\emptyset)$.)

$\Rightarrow Body_r \subseteq S^{\uparrow a}(\emptyset)$ and $\neg Head_r$ is c-unfounded w.r.t. r and I_a

(From the fact $S_r = \{\} \ \forall$ rule r and Def. 4.2.4, it follows that $\neg Head_r$ is c-unfounded w.r.t. r and I_a iff $\exists$ no rule $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $I_a(Body_{r'}) \neq 0$)

$\Rightarrow Body_r \subseteq S^{\uparrow a}(\emptyset)$ and $\exists$ no rule $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $Body_{r'} \cap \sim I_a = \emptyset$

$\Rightarrow Body_r \subseteq S^{\uparrow a}(\emptyset)$ and $\exists$ no rule $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $Body_{r'} \cap U^c(S^{\uparrow a}(\emptyset)) = \emptyset$

$\Rightarrow Body_r \subseteq S^{\uparrow a}(\emptyset)$ and $\exists$ no $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $(Body_{r'} \cup Head_{r'}) \cap U^c(S^{\uparrow a}(\emptyset)) = \emptyset$

$\Rightarrow Body_r \subseteq S^{\uparrow a}(\emptyset)$ and r is not c-defeasible w.r.t. $S^{\uparrow a}(\emptyset)$.

So, we have shown that the set of classical literals in CFM_P is a subset of the skeptical c-partial model of P. We will show by induction that $S^{\uparrow a}(\emptyset)$ is a subset of the set of classical literals in I_{a+1}, for all a. This is true when a=0. Suppose that it is true for all ordinals $\leq a$. We will show that $S^{\uparrow a+1}(\emptyset)$ is a subset of the set of classical literals in I_{a+2}. It is enough to show that for each rule r, if $Body_r \subseteq S^{\uparrow a}(\emptyset)$ and r is not c-defeasible w.r.t. $S^{\uparrow a}(\emptyset)$ then $Body_r \subseteq I_{a+1}$ and $\neg Head_r$ is c-unfounded w.r.t. r and I_{a+1}.

$Body_r \subseteq S^{\uparrow a}(\emptyset)$ and r is not c-defeasible w.r.t. $S^{\uparrow a}(\emptyset)$

(From the inductive hypothesis and the fact that $Body_r$ is free of default literals, it follows that $Body_r \subseteq I_{a+1}$.)

$\Rightarrow Body_r \subseteq I_{a+1}$ and $\exists$ no $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $(Body_{r'} \cup Head_{r'}) \cap U^c(S^{\uparrow a}(\emptyset)) = \emptyset$

$\Rightarrow Body_r \subseteq I_{a+1}$ and $\exists$ no rule $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $Body_{r'} \cap \sim I_{a+1} = \emptyset$

$\Rightarrow Body_r \subseteq I_{a+1}$ and $\exists$ no rule $r' \not< r$ with $Head_{r'} = \neg Head_r$ and $I_{a+1}(Body_{r'}) \neq 0$

$\Rightarrow Body_r \subseteq I_{a+1}$ and $\neg Head_r$ is c-unfounded w.r.t. r and I_{a+1}.

So, we have shown that the skeptical c-partial model of P is a subset of the set of classical literals in CFM_P. Proposition 4.4.5 now follows. ◊

Proposition 4.4.6 shows that the reliable semantics is more skeptical than the assumption-free semantics of [43]. The proposition follows immediately from Proposition 4.4.5 and the fact that the skeptical c-partial model of an ordered logic program P is a subset of the well-founded partial model of P [Theorem 8, 24].

Proposition 4.4.6: Let $P = \langle R_P, <_R \rangle$ be a *PEP* which is free from default literals and $C_r = \{\}$ $\forall$ rule r. Then, the set of classical literals in CFM_P is a subset of the well-founded partial model of P [43].

4.4.2 Semantics Following the Contrapositive Rule Approach

The use of contrapositives for resolving contradictions appears in [28, 29, 78]. Yet, in these works, rule prioritization is not considered and $C_r = Body_r$ $\forall$ rule r (all contrapositives of a rule are

considered). Let P be an extended program. Giordano and Martelli [28] define a *generalized stable model* (*GSM*) of P as a 2-valued stable model [26] of $exp(P)$ with the constraints $\{\bot \leftarrow L, \neg L | L \in HB_P\}$. The *generalized stable model semantics* (*GSMS*) is defined as the intersection of all the *GSMs* of P. Since not all programs have a 2-valued stable model, the *GSMS* of a program P is not always defined. The next proposition gives the relationship between *GSMS* and *CFS* and follows directly from Proposition 4.4.1.

Proposition 4.4.7: Let P be an extended program with $C_r = Body_r$ $\forall$ rule r then every *generalized stable model* of P is a stable c-model of P.

The reverse of Proposition 4.4.7 is not valid. For example, $\{a, \sim\neg a\}$, $\{a, \sim\neg a, \sim\neg p\}$ are stable c-models of $P=\{p. \quad \neg q. \quad a.\}$ and $P' = \{p \leftarrow\sim p. \quad a.\}$, respectively. Yet, the *GSMS* of P and P' are undefined.

Witteveen [78] defines the *strong belief revision model* (*SBRM*) of an extended program P as the *WFM* of the expansion of P with (i) the contrapositives of the rules, (ii) the rules $\{L \leftarrow \sim\neg L | L \in HB_P\}$, and (iii) the constraints $\{\bot \leftarrow L, \neg L | L \in HB_P\}$. Let $P=<R_P, <_R>$ with $<_R=\{\}$ and $C_r = Body_r$ $\forall$ rule r. When the rules $\{L \leftarrow \sim\neg L | L \in HB_P\}$ are removed from the expansion of P in [78] and *coherence* is enforced then the *SBRM* of P coincides with the $XWFM_{exp(P)}$ [54] (see Section 2.4.3). Then, it follows from Proposition 4.4.3 that when the *SBRM* of P exists, it coincides with CFS_P.

4.4.3 Doyle's Truth Maintenance System

A truth maintenance system (TMS) is a subsystem of an overall reasoning system. The problem solver of the reasoning system passes to the TMS the inferences it makes (*justifications*) and the TMS decides which of the propositions should be believed or not. The primary tasks of a TMS are:

1. *truth maintenance*: to provide for a consistent interpretation of the nodes in the network and to update the belief status of the nodes after the addition of new justifications.
2. *belief revision*: to perform conflict-resolution when a node which has been declared to be a contradiction is found to have a valid justification.

The need for belief revision has been the basic motivation for the introduction of TMSs. Many times decisions should be made with incomplete information. Because of this, choices may be found to be wrong and alternatives need to be considered. The two most influential types of TMS are: Doyle's TMS [18], also called *Justification-based TMS*, and the *Assumption-based TMS* [17], abbreviated ATMS. An ATMS supports reasoning from multiple hypothetical premise sets. Its main purpose is to compute for each proposition, minimal consistent assumption sets such as the proposition is derived from the corresponding hypothetical premises. Here, we will be concerned only with Doyle's TMS.

A *justification* is a formula of the form: j=< c←A,OUT(B) >. The belief of a proposition c is *justified* by a justification j=< c←A,OUT(B) > if all propositions in $A=\{a_1,...,a_n\}$ are believed and all propositions in $B=\{b_1,...,b_m\}$ are disbelieved. A distinguished proposition ⊥ is used to represent contradiction and when it is used as consequent of a justification, it allows constraints to be stated. A *dependency network* is a tuple (N,J), where J is a set of justifications and N contains all propositions appearing in some justification $j \in J$. The purpose of a TMS is to assign to all propositions in N, a label IN (believed) or OUT (disbelieved) in such a way that:

1) A proposition justified by at least one justification is labelled IN.

2) A proposition is labelled IN iff it is justified by a non-circular argument, i.e. a proposition cannot justify itself.

3) The proposition ⊥ is labelled OUT.

A label that satisfies the above three conditions is called *admissible*. Given a set of justifications J, the TMS tries to compute an admissible labelling. If it is the case that the proposition ⊥ is labelled *IN*, a contradiction has been found and the TMS has to trace back (*dependency-directed backtracking*) to find the non-monotonic justifications underlying the contradiction so as to revise the labelling. There are two possibilities: either there exists an alternative admissible labelling where no contradiction-node is IN or there is no such alterative labelling. In the second case, a new justification is created to get rid of the contradiction by making IN one of the OUT propositions supporting the belief in ⊥. Therefore, not only dependency-

directed backtracking produces a switching from one belief state to another for the given set of justifications but it also modifies the set of justifications itself. It can be seen that the conflict resolution process mainly relies on reasoning backwards from the contradiction ⊥ using justifications in their contrapositive directions. In fact, performing several inference steps through the contrapositives of the justifications, starting from ⊥ allows belief revision by forcing OUT-assumptions to be labelled IN. This will become clearer with the following example. Let

J={ TA(x).

foreign(x).

takes_credits(x,6)←TA(x), OUT(ab_TA(x)).

takes_credits(x,9)←foreign(x), OUT(ab_foreign(x)).

⊥← takes_credits(x,6), takes_credits(x,9).

}

The only well-founded and closed labelling of *J* is: IN={TA(x), foreign(x), takes_credits(x,6), takes_credits(x,9), ⊥}, OUT={ab_TA(x), ab_foreign(x)} which is inconsistent.

To eliminate the contradiction ⊥, the TMS adds the internal justifications:

ab_foreign(x)← takes_credits(x,6), foreign(x) or ab_TA(x)← takes_credits(x,9), TA(x).

This can be seen equivalently to adding the contrapositives:

{¬takes_credits(x,9)← takes_credits(x,6). ab_foreign(x)←foreign(x), ¬takes_credits(x,9).}

or {¬takes_credits(x,6)←takes_credits(x,9). ab_TA(x)←TA(x), ¬takes_credits(x,6).}

Then, the resulting admissible labellings will be

IN={TA(x), foreign(x), takes_credits(x,6), ab_foreign(x)}, OUT={ab_TA(x), takes_credits(x,9)}

or

IN={TA(x), foreign(x), takes_credits(x,9), ab_TA(x)}, OUT={ab_foreign(x),takes_credits(x,6)}.

An extended program *P* can be mapped to a dependency network (N_P, J_P) as follows [29]. Let N_P={*L*| *L* is a classical literal of *P*}. The set of justifications J_P is defined as follows:

• If *r*: $L_0 \leftarrow L_1,...,L_m,\sim L_{m+1},...,\sim L_n$ is a rule in *P* then J_P contains the justifications:

– $L_0 \leftarrow L_1,...,L_m,\text{OUT}(L_{m+1},...,L_n)$.

– $\neg L_i \leftarrow \neg L_0, L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_m, \text{OUT}(L_{m+1}, \ldots, L_n)$, where $i=1,\ldots,m$ and $L_i \in C_r$

– $L_j \leftarrow \neg L_0, L_1, \ldots, L_m, \text{OUT}(L_{m+1}, \ldots, L_{j-1}, L_{j+1}, \ldots, L_n)$, where $j=m+1,\ldots,n$ and $\sim L_j \in C_r$

- If A is an atom of P then J_P contains the justifications: $\bot \leftarrow A, \neg A$.
- J_P does not contain any other justification.

The following relationship between the *GSMS* [28] (described in section 4.4.2) of an extended program P with $C_r = Body_r$ $\forall$ rule r and the admissible labellings of (N_P, J_P) is shown in [29]:

Proposition 4.4.8 [29]: Let P be an extended program with $C_r = Body_r$ $\forall$ rule r. Then, an admissible labelling G of (N_P, J_P) is mapped to a 2-valued generalized stable model M_G of P and vice versa, as follows: L is IN (resp. OUT) w.r.t. G iff L is true (resp. false) w.r.t. M_G, where $L \in HB_P$.

An admissible labelling G of (N_P, J_P) is mapped to a stable c-model M_G of P in a similar way. However, the reverse is not true since there is no admissible labelling for all dependency networks. For example, the dependency networks

$D_1 = (\{p, \neg p, a\}, J)$ where $J=\{p. \quad \neg p. \quad a. \quad \bot \leftarrow p, \neg p.\}$ and

$D_2 = (\{p, a\}, J)$ where $J=\{p \leftarrow \text{OUT}(p). \quad a.\}$

do not have any admissible labelling.

Elkan [22] has shown that given a dependency network $D=(N,J)$ without constraints, the problem of finding whether D has an admissible labelling is *NP*-complete w.r.t. the size of N.

Given a set of rules which admits multiple admissible solutions, a TMS considers one solution per time. This is because the main goal of a TMS is to maintain a consistent set of beliefs. However, the main goal of our semantics is to reason about knowledge that can be inconsistent and draw conclusions that are satisfied in all of stable c-models of the knowledge base. This is the *skeptical* view of a knowledge source. If only the computation of one admissible state is required from an application then a TMS has the advantage that additional information about the modelled world will not require the recomputation of all the derived beliefs since many of them will still be valid. According to the *CFS*, additional information entails the computation of a new contradiction-

free model, that is, we have to start from scratch. This is not a major disadvantage when the skeptical "meaning" of the knowledge base is desired.

4.5 Procedural Semantics

In this section, we present *SLCF*-resolution (*CF* for contradiction-free), a proof procedure to answer queries on *PEPs* based on the contradiction-free semantics. *SLCF*-resolution is inspired by the approach to constructive negation taken in *SLDFA*-resolution [59].

Substitutions in goals are replaced by constraints which are represented by Greek letters. The idempotent substitution $\{x_1/t_1,..., x_n/t_n\}$ corresponds to the constraint $x_1=t_1,...,x_n=t_n$. A constraint θ is *satisfiable* iff $CET \models \theta$, where *CET* stands for the Clark equality theory [45]. We use the letter Q to represent a list of literals. A *goal* is a formula $\leftarrow\theta,Q$, where θ is a satisfiable constraint. An *SLCF-refutation* of a goal is defined in Def. 4.5.1. Let $\leftarrow\theta,Q$ be a goal. $W_P{}^{\uparrow a}(\emptyset) \models \exists\theta,Q$ iff there exists an *SLCF*-refutation of rank a for the goal $\leftarrow\theta,Q$. A selection function selects a literal from a goal. The selected literal of a goal is underlined, e.g., the selected literal of $\leftarrow\theta,a,\underline{L},b$ is L.

Definition 4.5.1: Let P be a *PEP* and a be a countable ordinal. An *SLCF-refutation of rank* $a \geq 1$ for a goal G is a sequence of goals $G_1,...,G_n$ s.t. $G_1=G$, $G_n= \leftarrow\theta''$ and $\forall G_i$ one of the following is true:

1. (i) $G_i= \leftarrow\theta,Q,\underline{L}(x_1,...,x_n),Q'$ and

   (ii) $\exists$ a variant r: $L(t_1,...,t_n)\leftarrow L_1,...,L_m$ of a rule in $exp(P)$ and

   (iii) $\exists\theta'$ s.t. $\leftarrow\theta,\theta',c(\neg L)$ r-fails at rank $\leq a$ (Def. 4.5.3) and

   (iv) $G_{i+1}= \leftarrow\theta,\theta',(x_1=t_1,...,x_n=t_n),Q, L_1,...,L_m,Q'$.

or

2. $G_i= \leftarrow\theta,Q,\sim\underline{L},Q'$ where L is a classical literal and one of the following is true:

   (i) there is θ' s.t. $\leftarrow\theta,\theta',L$ fails at rank $< a$ (Def. 4.5.2) and $G_{i+1}= \leftarrow\theta,\theta',Q,Q'$ or

   (ii) there is an *SLCF*-computed answer θ' of rank $\leq a$ for $\leftarrow\theta,\neg L$ and $G_{i+1}= \leftarrow\theta,\theta', Q,Q'$.

The constraint θ'' restricted to the free variables of G is an *SLCF-computed answer of rank* a for G.

A goal $\leftarrow\theta,\theta',c(L)$ *r*-fails at rank a+1 iff every ground instance of θ,θ',L is *c*-unfounded w.r.t. *r* and $\mathbf{W}_P{}^{\uparrow a}(\emptyset)$. Thus, condition 1 expresses that the head L of a rule *r* is true if all literals in the body of the rule are true and $\neg L$ is *c*-unfounded w.r.t. *r* and *CFS*. Note that an *SLCF*-computed answer of rank a+2 for $\leftarrow\theta,\sim L$ is θ,θ' iff (i) $\exists\theta'$ s.t. the goal $\leftarrow\theta,\theta',L$ fails at rank a+1 which means that every ground instance of $\theta,\theta',L\in \mathbf{F}(\mathbf{W}_P{}^{\uparrow a}(\emptyset))$ or (ii) $\exists\theta'$ s.t. θ,θ' is satisfiable and every ground instance of $\theta,\theta',\neg L\in \mathbf{W}_P{}^{\uparrow a+2}(\emptyset)$ (*coherence*). The constraint θ' in conditions 1(iii) and 2(i) of Def. 4.5.1 can be computed similarly to the way failed answers are computed in [59].

Definition 4.5.2: Let *P* be a *PEP* and $a\geq 1$ a countable ordinal. A goal *G fails at rank a* iff there exists a tree *T* s.t. (i) *T* has root *G*, (ii) no node of *T* has the form $\leftarrow\theta$, and (iii) $\forall$ node *N*, *N* is a goal and

1. If $N=\leftarrow\theta,Q,\underline{L}(x_1,\ldots,x_n),Q'$ s.t. *L* is a classical literal then one of the following is true:
   (i) *N* is not the root, there is an *SLCF*-computed answer θ' of rank $< a$ for $\leftarrow\theta,\neg L(x_1,\ldots,x_n)$, and *N* has children: $\leftarrow\theta,\theta_1,Q,L(x_1,\ldots,x_n),Q',\ldots,\leftarrow\theta,\theta_m,Q,L(x_1,\ldots,x_n),Q'$, where $CET\models\theta\rightarrow\theta'\vee\theta_1\vee\ldots\vee\theta_m$.
   (ii) For every variant *r*: $L(t_1,\ldots,t_n)\leftarrow L_1,\ldots,L_m$ of a rule in $exp(P)$ s.t. $\theta,(x_1=t_1,\ldots,x_n=t_n)$ is satisfiable, *N* has a child: $\leftarrow\theta,(x_1=t_1,\ldots,x_n=t_n),Q,L_1,\ldots,L_m,Q'$.
2. If $N=\leftarrow\theta,Q,\sim\underline{L}(x_1,\ldots,x_n),Q'$ s.t. *L* is a classical literal then one of the following is true:
   (i) There is an *SLCF*-computed answer θ' of rank $<a$ for $\leftarrow\theta,L(x_1,\ldots,x_n)$ and *N* has children: $\leftarrow\theta,\theta_1,Q,\sim L(x_1,\ldots,x_n),Q',\ \ldots,\ \leftarrow\theta,\theta_m,Q,\sim L(x_1,\ldots,x_n),Q'$, where $CET\models\theta\rightarrow\theta'\vee\theta_1\vee\ldots\vee\theta_m$.
   (ii) *N* has a child: $\leftarrow\theta,Q,Q'$.

Definition 4.5.3: Let *P* be a *PEP*, $a\geq 1$ a countable ordinal and $G=\leftarrow\theta'',c(K)$ a goal where *K* is a classical literal. *G r-fails at rank a* iff there exists a tree *T* s.t. (i) *T* has root *G* (ii) no node of *T* has the form $\leftarrow\theta$, and (iii) $\forall$ node *N*, *N* is a goal and

1. If $N=\leftarrow\theta,Q,c(\underline{L}(x_1,\ldots,x_n)),Q'$ s.t. *L* is a classical literal then one of the following is true:

(i) N is not the root, there is an *SLCF*-computed answer θ' of rank $< a$ for $\leftarrow\theta,\neg L(x_1,..., x_n)$, and N has children: $\leftarrow\theta,\theta_1,Q,c(L(x_1,..., x_n)),Q'$, ..., $\leftarrow\theta,\theta_m,Q,c(L(x_1,..., x_n)),Q'$, where $CET \models \theta \rightarrow \theta' \vee \theta_1 \vee ... \vee \theta_m$.

(ii) For every variant r': $L(t_1,..., t_n) \leftarrow L_1,...,L_m$ of a rule in $exp(P)$ s.t. $\theta,(x_1=t_1,..., x_n=t_n)$ is satisfiable and $r' \not< r$, N has a child: $\leftarrow\theta,(x_1=t_1,..., x_n=t_n),Q,L'_1,...,L'_m,Q'$, where:

– if $L_i \in C_r$ and L_i is a classical literal then $L'_i = c(L_i)$,

– if $L_i \in C_r$ and L_i is the default literal $\sim L'$ then $L'_i = c(\neg L')$,

– if $L_i \notin C_r$ and L_i is a classical literal then $L'_i = L_i$,

– if $L_i \notin C_r$ and L_i is the default literal $\sim L'$ then $L'_i = \neg L'$.

2. If $N = \leftarrow\theta,Q,L(x_1,..., x_n),Q'$ s.t. L is a classical literal then one of the following is true:

(i) There is an *SLCF*-computed answer θ' of rank $< a$ for $\leftarrow\theta,\neg L(x_1,..., x_n)$ and N has children: $\leftarrow\theta,\theta_1,Q,L(x_1,..., x_n),Q'$, ..., $\leftarrow\theta,\theta_m,Q,L(x_1,..., x_n),Q'$, where $CET \models \theta \rightarrow \theta' \vee \theta_1 \vee ... \vee \theta_m$.

(ii) N has a child $\leftarrow\theta, Q,Q'$.

We will give a different characterization of *CFS* where $W_P(I)$ is expressed through the $least_v$ models of transformations of P w.r.t I. The new characterization is used in the proof of the soundness and completeness of the *SLCF*-resolution. Let P be a *PEP*, *ground*(P) the ground instantiation of P and I an interpretation of it.

Let r be a rule of *ground*(*exp*(P)). The program $P/_u(r,I)$ is defined as follows:

(i) Remove from *ground*(*exp*(P)) any rule r' s.t. $r' < r$.

(ii) Replace every default literal $\sim L$ in the new program with $\neg L$.

(iii) Replace every literal L in the body of a rule in the new program s.t. $I(\neg L)=1$ with *false*.

(iv) If L is in the body of a rule r in the new program and $L \notin S_r$, replace L with *true*.

Note that the $least_v$ model of $P/_u(r,I)$ contains all literals that are not c-unfounded w.r.t. r and I.

The program $P/_t I$ is defined as follows:

(i) Remove from *ground*(*exp*(P)) any rule r s.t. $\neg Head_r \in least_v(P/_u(r,I))$.

(ii) Add to the new program the rules $\{\sim L. \mid \sim L \in I\}$.

Note that the $least_v$ model of $P/_t I$ contains (i) the classical literals that are true in $\mathbf{W}_P(I)$ and (ii) the default literals in I. Step (i) expresses that if the body of a rule r is true then $Head_r$ is derived only if $\neg Head_r$ is c-unfounded w.r.t. r and I.

The program $P/_f I$ is defined as follows:

(i) Replace every classical literal L in the body of a rule in $ground(exp(P))$ s.t. $I(\neg L)=1$ with *false*.

(ii) Add to the new program the rules $\{\sim L. \mid \sim L \notin I\}$.

The $least_v$ model of $P/_f I$ contains all classical literals that are not in $\mathbf{F}(I)$.

The programs $P/_u(r,I)$, $P/_t I$ and $P/_f I$ are positive logic programs because literals $\neg A$ and $\sim A$, where A is an atom, are treated as new atoms. It can be seen that $\mathbf{T}_P(I)=least_v(P/_t I)$, $\mathbf{F}_P(I)=HB_P - least_v(P/_f I))$. Thus, $\mathbf{W}_P(I)= coh(least_v(P/_t I) \cup \sim(HB_P - least_v(P/_f I)))$. Note that if P is a positive program then $least_v(P)= T_P{\uparrow}\omega(\emptyset)$, where T_P is the immediate consequence operator of van Emden and Kowalski [75]. Let $I_0=\{\}$, $I_{a+1}=\mathbf{W}_P(I_a)$ and $I_a= \cup\{I_b \mid b<a\}$ if a is a limit ordinal.

Lemma 4.5.1 (Soundness): Let P be a *PEP*, R a selection rule, L a classical literal, K a literal, and r a rule.

1. If θ' is an *SLCF*-computed answer of rank a for goal $\leftarrow\theta,K$ then every ground instance of θ',K is true w.r.t. I_a.
2. If goal $\leftarrow\theta,L$ fails at rank $a+1$ then every ground instance of θ,L is in $\mathbf{F}(I_a)$.
3. If goal $\leftarrow\theta,c(L)$ r-fails at rank $a+1$ then every ground instance of θ,L is c-unfounded w.r.t. r and I_a.

Proof: We will prove the proposition by transfinite induction. Assume that it is true for $b \leq a$. We will prove that it is also true for $a+1$.

1. By part 2 and 3 of the inductive hypothesis, every ground instance of an *SLCF*-refutation of rank $a+1$ is an *SLD*-refutation [45] w.r.t the positive program $P/_t I_a$. By the soundness of the

SLD-resolution, every ground instance of $\theta\vartheta,L$ is true w.r.t. $least_v(P/_t I_a)$. Thus, every ground instance of θ,L belongs to $W_P(I_a)$ since $W_P(I)= coh(least_v(P/_t I) \cup \sim(HB_P - least_v(P/_f I)))$.

2. Assume L' is a ground instance of θ,L s.t. part 2 does not hold. This implies that there is a tree T that satisfies the conditions of Def. 4.5.2 and L' is not in $F(I_a)$. Since L' is not in $F(I_a)$, $L' \in least_v(P/_f I_a)$. By the completeness of the *SLD*-resolution, for the computation rule used in the tree T, there exists an *SLD*-refutation for $\leftarrow L'$ in $P/_f I_a$. Thus, T has a node of the form $\leftarrow\theta'$ which is a contradiction.
3. Assume L' is a ground instance of θ,L s.t. part 3 does not hold. This implies that there is a tree T that satisfies the conditions of Def. 4.5.3 and L' is not c-unfounded w.r.t. r and I_a. Since L' is not c-unfounded w.r.t. I_a, $L' \in least_v(P/_u(r, I_a))$. By the completeness of the *SLD*-resolution, for the computation rule used in the tree T, there exists an *SLD*-refutation for $\leftarrow L'$ in $P/_u(r, I_a)$. Thus, T has a node of the form $\leftarrow\theta'$ which is a contradiction. ◊

Lemma 4.5.2 (completeness): Let P be a *PEP*, R a selection rule, L a classical literal, and $\leftarrow\theta,L$ a goal.

1. If a ground instance L' of θ,L is true w.r.t. I_a then there is an *SLCF*-computed answer θ' of rank a for goal $\leftarrow\theta,L$ s.t. L' is a ground instance of θ',L.
2. If every ground instance of θ,L is in $F(I_a)$ then goal $\leftarrow\theta,L$ fails at rank a+1.
3. If every ground instance of θ,L is c-unfounded w.r.t. a rule r and I_a then goal $\leftarrow\theta,c(L)$ r-fails at rank a+1.

Proof: We will prove the proposition by transfinite induction. Assume that it is true for $b \leq a$. We will prove that it is also true for a+1.

1. Since $L' \in I_{a+1}$, it follows that $L' \in least_v(P/_t I_a)$. By the completeness of the *SLD*-resolution there is an *SLD*-refutation for $\leftarrow L'$ and $P/_t I_a$. By the mgu lemma [45], there is an *SLD*-refutation R of $\leftarrow L$ with computed answer τ s.t. L' is an instance of τL. From parts 2 and 3 of the inductive hypothesis, there is θ s.t. adding θ to every goal of R results in an *SLCF*-refutation of rank a+1 of $\leftarrow\theta,L$ with answer θ' s.t. L' is a ground instance of θ',L.

2. Assume that every ground instance of θ,L is in $F(I_a)$ but $\leftarrow\theta,L$ does not fail at rank a+1. Consider a tree T with root $\leftarrow\theta,L$ that satisfies the conditions (iii)1 and (iii)2(i) of Def. 4.5.2. Then, T should have a node of the form $\leftarrow\theta''$. Consider a ground instance T' of T. Then, T' is an *SLD*-derivation of a ground instance of θ,L in $P/_f I_a$ with root L'. By the soundness of the *SLD*-resolution, $L'\in least_v(P/_f I_a)$. So, there is a ground instance of θ,L which is not in $F(I_a)$ which is a contradiction.
3. Assume that every ground instance of θ,L is c-unfounded w.r.t. a rule r and I_a but $\leftarrow\theta,L$ does not r-fail at rank a+1. Consider a tree T with root $\leftarrow\theta,L$ that satisfies the conditions (iii)1 and (iii)2(i) of Def. 4.5.3. Then, T should have a node of the form $\leftarrow\theta''$. Consider a ground instance T' of T. Then, T' is an *SLD*-derivation of a ground instance of θ,L in $P/_u(r,I)$ with root L'. By the soundness of the *SLD*-resolution, $L'\in least_v(P/_u(r,I))$. So, there is a ground instance of θ,L which is not c-unfounded w.r.t. r and I_a which is a contradiction. ◊

The next proposition follows from lemmas 4.5.1 and 4.5.2.

Proposition 4.5.1 (soundness, completeness): Let P be a *PEP*, R a selection rule, Q a list of classical and default literals and $G\leftarrow\theta,Q$ a goal. If θ' is an *SLCF*-computed answer for G then every ground instance of θ',Q is true w.r.t. CFM_P. If $Q\tau$ is ground and true w.r.t. CFM_P then there is an *SLCF*-computed answer θ' for G s.t. $Q\tau$ is a ground instance of θ',Q. Every ground instance of θ,Q is false w.r.t. CFM_P iff the goal $\leftarrow\theta,Q$ fails.

Example 4.5.1: Consider the *PEP*, $P=<R_P, <_R>$:

$R_P=\{r_1: \neg q(0).\quad r_2: \neg q(1).\quad r_3: p(X).\quad r_4: \neg q(X)\leftarrow \sim s(X).\quad r_5: q(X)\leftarrow p(X).$

with $C_r=Body_r\ \forall$ rule $r\}$ and $r_1<r_3$. Then, the expanded program $exp(P)$ is as follows:

$R_{exp(P)}=\{r_1: \neg q(0).\quad r_2: \neg q(1).\quad r_3: p(X).\quad r_4: \neg q(X)\leftarrow \sim s(X).\quad r'_4: s(X)\leftarrow q(X).$

$r_5: q(X)\leftarrow p(X).\quad r'_5: \neg p(X)\leftarrow \neg q(X).$ with $C_r=Body_r\ \forall$ rule $r\}$ and $r_1<r_3$.

An *SLCF*-refutation for $\leftarrow q(X)$ is: $G_1 = \leftarrow \underline{q}(X)$, $G_2 = \leftarrow X \neq 1, \underline{p}(X)$ (using rule r_5 in condition 1 of Def. 4.5.1).

The goal $\leftarrow X \neq 1, c(\neg q(X))$ r_5-fails since there exists a tree satisfying the conditions of Def. 4.5.3.

$\leftarrow X \neq 1, c(\neg \underline{q}(X))$

| (using rule r_4 in condition 1(ii) of Def. 4.5.3)

$\leftarrow X \neq 1, c(\neg \underline{s}(X))$ (it is a leaf)

$G_3 = \leftarrow X \neq 1$ (using rule r_3 in condition 1 of Def. 4.5.1).

The goal $\leftarrow X \neq 1, c(\neg p(X))$ r_3-fails since there exists a tree satisfying the conditions of Def. 4.5.3.

$\leftarrow X \neq 1, c(\neg \underline{p}(X))$

| (using rule r'_5 in condition 1(ii) of Def. 4.5.3)

$\leftarrow X \neq 1, c(\neg \underline{q}(X))$

Note that, since $r_1 < r_3$, rule r_1 cannot be used in condition 1(ii) of Def. 4.5.3.

Since $X \neq 1, X = 1$ is unsatisfiable, rule r_2 cannot be used in condition 1(ii) of Def. 4.5.3.

| (using rule r_4 in condition 1(ii) of Def. 4.5.3)

$\leftarrow X \neq 1, c(\neg \underline{s}(X))$ (it is a leaf)

So, an *SLCF*-computed answer for $\leftarrow q(X)$ is $X \neq 1$.

SLCF-resolution is an ideal procedural semantics since termination is not guaranteed for all programs. A proof procedure for propositional logic programs that incorporates loop detection and always terminates is given in Appendix B.

4.6 Conclusions

In this chapter, we presented the *contradiction-free semantics* (*CFS*) for prioritized extended programs (*PEP*). We gave both a fixpoint and model theoretic characterization of *CFS* and proved

that they are equivalent. *CFS* is always defined and non-contradictory. The *CFS* fixpoint operator avoids contradictions by taking the second approach presented in section 1.3.

Every rule r has a corresponding set $C_r \subseteq Body_r$ which is called the *contrapositive set* of r. If r is a rule s.t. both $Body_r$ and $\neg Head_r$ are derived from P then the value of C_r indicates which literals are "suspect" for the contradiction. When $C_r=\{\}$, only $Head_r$ is "suspect." When $C_r \neq\{\}$, the literals in C_r are also considered "suspect." To facilitate this reasoning, P is expanded with the contrapositives r' of every rule r such that $Head_{r'} \in \{\neg L \mid L \in C_r\}$. The motivation behind the idea of the contrapositive sets is given in section 1.3. Criteria for defining the values of the contrapositive sets are given in subsection 4.2.

In the computation of the fixpoint of the *CFS* fixpoint operator, when $Body_r$ is true for a rule r in $exp(P)$ then $Head_r$ is evaluated as true iff $\neg Head_r$ cannot be derived from rules in $exp(P)$ with priority no lower than r. The model theoretic characterization of the *CFS* of P is given by defining the *stable c-models* of P. In section 4.3, we proved that *CFS* is the least stable c-model of P and that when the Herbrand base of P is finite, the complexity of computing *CFS* is polynomial w.r.t. $|exp(P)|$.

In section 4.4, we relate *CFS* with existing semantics. *CFS* extends the well-founded semantics for normal programs [76] to *PEPs*. *CFS* is a proper generalization of the approaches followed in *ordered logic* [24, 43] and *conservative vivid logic* [77] which correspond to the case that $S_r=\{\}$ $\forall$ rule r. However, $S_r=\{\}$ $\forall$ rule r, expresses only exceptions. *CFS* also generalizes the approach followed in the *generalized stable model semantics* [28] and *strong belief revision semantics* [78] which corresponds to the case that $S_r=Body_r$ $\forall$ rule r. However, $S_r=Body_r$ $\forall$ rule r is not always correct since the contraposition may not hold for default rules as it was indicated in section 1.3. Thus, *CFS* gives a new unifying definition of these approaches.

Specifically, if P is an extended program with $S_r=Body_r$ $\forall$ rule r then the *CFS* of P is a subset of the *generalized stable model semantics* of P [28], if the latter is defined. Moreover, the *CFS* of P coincides with the *strong belief revision semantics* of P, if the latter is defined. If P is an ordered logic program then the *CFS* of P coincides with the *skeptical c-partial model* of P [24] and

is a subset of the *well-founded partial model* of P [43]. Ordered logic does not support negation by default. Generalized stable model semantics and strong belief revision semantics do not support rule prioritization and fail to give semantics to every contradictory extended program.

A procedural counterpart to the declarative semantics for *CFS* is developed in section 4.5. The *SLCF*-resolution for computing answers for *PEPs* is presented. *SLCF*-resolution is based on the *constructive negation* approach in which answering of non-ground queries with negated atoms is attempted. The *SLCF*-resolution is shown to be sound and complete w.r.t. *CFS*.

CHAPTER 5

RELIABLE OBJECT LOGIC

5.1 Introduction

In object-oriented databases, data and behavior are encapsulated into object classes which are structured in a generalization hierarchy. A class lower in the hierarchy (subclass) inherits the general behavior of its ancestors (superclasses). General behavior may be overridden by special behavior defined in the subclasses. Logic programming has a profound effect on object-oriented databases providing both their logical foundations and extending their power.

Many proposals have tried to combine object-oriented and logic programming [81, 37, 38, 32, 13, 11, 47, 36]. In [32], *object-preserving rules* are used to extend an object of a class C' to an object of a subclass C of C'. Each class C has a set of explicit attributes E_C. Objects of class C can be assigned a value for any of the E_C attributes. They can also be assigned a value for any of the explicit attributes of the superclasses of the class C. This way, each class inherits all attribute definitions from its superclasses. However, the sets of explicit attributes in different classes should be disjoint and inherited attributes cannot be redefined. Thus, only *monotonic inheritance* is supported. The *unambiguous naming requirement* imposes severe constraints which contradict with the properties of object-oriented programming such as modularity, reusability and incremental design. Moreover, no parameterized attributes (methods) are expressed.

In *F-Logic* [37, 38], classes and individual objects are indistinguishable and both are considered as objects in the class hierarchy. Individual objects are leafs in the class hierarchy even though the reverse is not always true. Deductive rules manipulate the whole class hierarchy which is not fixed a priory. Deductive rules can create new objects on the fly. An *IS-A* term $C{:}C'$ in the head of a rule can be used to indicate that a class C is a subclass of C' or that an individual object

C is an instance of a class *C'*. Among all *untyped* models of a program, the *minimal for inheritance models* are selected. The defined inheritance is inheritance of values and not of rules. However, we feel that behavior (rules) and not actual values is what should be inherited.

In [11], rules are inherited from superclasses to subclasses. As it was indicated in subsection 2.4.6, in [11], rules are considered to be clauses, i.e., there is distinction between the head and the body of the rule. The authors assume that methods have only one signature and that object names indicate the class of the object. Thus, the presence of the *most specific class* for each object is assumed. This requires an unreasonable large number of classes to support multiple object roles [51]. The authors define the *intended* semantics of an object. Their definition, given in subsection 2.4.6, captures the non-monotonic inheritance of clauses. When inherited and local clauses contradict, the local clauses are given higher priority. The *global semantics* of a program does not always exist.

In this Chapter, we describe an object-oriented logic programming language, called *reliable object logic* (*ROL*). In *ROL*, data and behavior are encapsulated into classes which are structured in a generalization lattice. Object-registration rules are used to register an object to a class or to exclude it from a class. For example, the information that a student is (resp. is not) graduate is represented if the student is registered to (resp. excluded from) the *grad-student* class. In *ROL*, object-registration rules can register an object to multiple classes. So, in contrast to [11], *ROL* supports multiple object roles without the requirement of the most specific class for an object. For example, a student can be member of the *TA* class and the *foreign_student* class even though no *foreign_TA* class exists.

Method rules define the behavior of the objects of a class. Method rules are inherited from the superclasses to subclasses. This means that a member of a class *C* should satisfy the method rules of the superclasses of *C* unless a conflict occurs. Some of the conflicts can be resolved using a rule ordering relation. According to the specificity dominance principle, the method rules of a class usually have higher priority than these of its superclasses. The reliable semantics of a *ROL* program is defined by translating the *ROL* program to an equivalent *EPP*. In *DOODs*, rule

prioritization can be used (i) to express the fact that specific rules are more reliable than general ones, (ii) to give priorities to inherited rules that are in conflict as a result of multiple inheritance and (iii) to give priorities to class rules that are in conflict as a result of multiple specializations of the same object.

5.2 Reliable Object Logic Programs

The alphabet of *ROL* contains a finite set of class, object and method names, variable symbols including *Self*. Classes are related with a subclass-superclass relationship, $<_C$, which is a strict partial order. Objects are registered into classes and manipulated through the *ROL* rules.

Each class C has a set of parameters, called *parameters* of C. The signature of C specifies the classes of these parameters.

Definition 5.2.1 (class signature): The *signature* of a class C has a form: $C(C_1,...,C_n)$ where $C_1,...,C_n$ are the classes of the parameters of C.

The signature of a method *meth* on a class C specifies the classes of the input/output parameters of *meth* when it is applied to an object of class C.

Definition 5.2.2 (declared method signature): The *declared signature*, $DSIG_C(meth)$, of a method *meth* on a class C has a form: $meth(C_1,...,C_n)$ where n is the arity of *meth* and $C_1,...,C_n$ are the classes of the parameters of *meth*. $DSIG_C(meth)$ may not be defined.

When the declared signature of a method on a class C is not defined, the signatures of the method on the direct superclasses of C are inherited.

Definition 5.2.3 (method signature): The *signature*, $SIG_C(meth)$, of a method *meth* on class C is defined as follows: If $DSIG_C(meth)$ is defined then $SIG_C(meth)=\{DSIG_C(meth)\}$. Otherwise, $SIG_C(meth)=\{DSIG_{C'}(meth) \mid DSIG_{C'}(meth)$ is defined, $C<_C C'$ and there is $C<_C...<_C C''<_C..<_C C'$ such that for all C'' in the sequence with $C''<_C C'$, $DSIG_{C''}(meth)$ is not defined$\}$.

An object registration atom t_0:C or t_0:$C(t_1,...,t_n)$ indicates that the object t_0 is member of the class C with parameters $t_1,...,t_n$. A method atom $t_0.meth(t_1,...,t_n)$ indicates that the method *meth* is applied to the object t_0 with input-output parameters $t_1,...,t_n$.

Definition 5.2.4 (term, atom, literal):

- A *term* is a variable or an object name.
- An *object registration atom* has the form: t_0:C or t_0:$C(t_1,...,t_n)$ where C is a class name with n parameters and $t_0,...,t_n$ are terms.
- A *method atom* has the form: $t_0.meth(t_1,...,t_n)$ where *meth* is a method with arity n and $t_0,...,t_n$ are terms.
- An *object registration* (resp. *method*) *literal* is an object registration (resp. method) atom A, its classical negation $\neg A$ or its default negation $\sim A$.

Definition 5.2.5 (rule, constraint):

- An *object registration rule* has the form: r: $L \leftarrow L_1,...,L_m$ where L is an object registration literal and $L_1,...,L_m$ are literals. The preliminary suspect set S_r of rule r is a subset of $\{L_1,...,L_m\}$.
- A *method rule* has the form r: $Self.meth(t_1,..,t_n) \leftarrow L_1,...,L_m$ or r: $\neg Self.meth(t_1,..,t_n) \leftarrow L_1,...,L_m$, where $Self.meth(t_1,..,t_n)$ is a method atom and $L_1,...,L_m$ are literals. The preliminary suspect set S_r of rule r is a subset of $\{L_1,...,L_m\}$.
- A *constraint* has the form $\perp \leftarrow L_1,...,L_m$, where $L_1,...,L_m$ are literals.

When the head of a rule r is t:C or t:$C(t_1,...,t_n)$, we say that r registers object t to C. When the head of a rule r is $\neg t$:C or $\neg t$:$C(t_1,...,t_n)$, we say that r excludes object t from C.

Definition 5.2.6 (class specification): The specification of a class C is a tuple $SPEC_C$=<SIG_C, $MSIG_C$, $label_C$, R_C> where:

– SIG_C is the signature of class C.

– $label_C$ has the form $C(X_1,...,X_n)$ where $X_1,...,X_n$ are universally quantified variables, representing the parameters of C.

– $MSIG_C$ is the set of declared signatures of methods on C.

– MR_C is a finite set of method rules.

When a rule $r \in MR_C$, we say that r is applied to the members of class C.

Definition 5.2.7 (*ROL* program): A *ROL* program is a tuple $P=<CSPEC_P, OR_P, IC_P, <_C, <_R>$ where:

– $CSPEC_P$ is the set of all class specifications.

– OR_P is a finite set of object registration rules.

– IC_P is a finite set of constraints.

– $<_C$ is a strict partial ordering relation between classes.

– $<_R$ is a strict partial ordering relation between method rules.

If r,r' are method rules such that $r<r'$ then r' is considered more reliable than r only when r,r' are applied to the same object.

Example 5.2.1: The following is a *ROL* program (signatures are ignored and terms starting with a capital letter are variables). The class hierarchy is given in Figure 5.1.

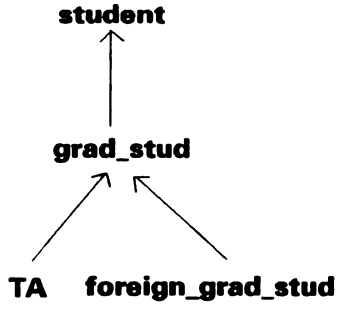

Figure 5.1: The class hierarchy

Class Specifications ($CSPEC_P$):

student(Name,Country,Credits){

r_1: Self.name(Name).

r_2: Self.citizenship(Country).

r_3: Self.full_time()←Self.full_time_credits(X), Credits≥X. S_{r3}={Self.full_time_credits(X)}

/* If a student takes more credits than needed for the student to be considered full-time then the student is full-time. */

grad_stud(){

r_4: Self.full_time_credits(9).

TA(Course){ /* TA stands for teaching assistant */

r_5: Self.teaches(Course).

r_6: Self.full_time_credits(6).}

foreign_grad_stud(){

r_7: Self.full_time_credits(12).}

Object Registration Rules (OR_P):

r_8: X:foreign_grad_stud←X:grad_stud, X.citizenship(Y),Y≠"USA".

S_{r8}={X:grad_stud, X.citizenship(Y)}

r_9: ann:student("Ann","UK",12).

r_{10}: ann:TA.

Constraints (IC_P):

⊥←X.full_time_credits(Y), X.full_time_credits(Z),Y≠Z. /* *full_time_credits* is functional */

Rule Priorities: $r_4<r_6$, $r_4<r_7$, $r_7<r_6$. /* Specificity dominance principle and regulation */

The program gives the number of credits that a graduate student needs to take to be considered full time. If a graduate student is not TA (teaching assistant) or foreign student then he/she needs 9 credits. If a foreign graduate student is not TA then he/she needs 12 credits. A TA needs only 6 credits.

5.3 Reliable Semantics for *ROL* programs

Let P be a *ROL* program. To define the *reliable semantics* of P, P is mapped to an *EPP*, $f_P(P)$.

Definition 5.3.1 (f_L transformation) Let P be a *ROL* program. The f_L transformation maps a *ROL* literal to a conventional literal as follows:

- If $t_0{:}C(t_1,...,t_n)$ is an atom of P then $f_L(t_0{:}C(t_1,...,t_n))=C(t_0,t_1,...,t_n)$ and $f_L(t_0{:}C)$=member_$C(t_0)$.
- If $t_0.meth(t_1,...,t_n)$ is an atom of P then $f_L(t_0.meth(t_1,...,t_n))=meth(t_0,t_1,...,t_n)$.
- If A is an atom of P then $f_L(\neg A)= \neg f_L(A)$ and $f_L(\sim A)=\sim f_L(A)$.

If S is a set of *ROL* literals then $f_L(S)=_{def}\{f_L(L)\mid L\in S\}$. In the next definition, we symbolize $Body_r$ with Bd_r and S_r is subscript of $\leftarrow$ in rule r.

Definition 5.3.2 (f_P transformation) f_P maps a *ROL* program, P, to an *EPP* as follows:

(i) If C is a class with n parameters then $f_P(P)$ contains the rule:

r: member_$C(X_0)\leftarrow_{Bd_r} C(X_0,X_1,...,X_n)$.

(ii) If C is a class with signature $C(C_1,...,C_n)$ then $f_P(P)$ contains the rule:

r: wtyped_$C(X_1,...,X_n)\leftarrow_{Bd_r}$ member_$C_1(X_1)$,..., member_$C_n(X_n)$.

(iii) Let *meth* be a method and C be a class. If $C'<_e C$, $meth(C_1,...,C_n)\in SIG_C(meth)$ and $C'_1,...,C'_m$ are the direct subclasses of C' then $f_P(P)$ contains the rule:

r: wtyped_*meth*_$C(X',X_1,...,X_n)\leftarrow_{Bd_r}$ member_$C'(X')$,~member_$C'_1(X')$,...,~member_$C'_m(X')$, member_$C_1(X_1)$,...,member_$C_n(X_n)$.

(iv) If C is a direct subclass of C' then $f_P(P)$ contains the rules:

r: member_$C'(X)\leftarrow_{Bd_r}$ member_$C(X)$ and r': $\neg$member_$C(X)\leftarrow_{Bd_{r'}}\neg$member_$C'(X)$.

(v) Let r: $Self.meth(t_1,...,t_m)\leftarrow_{S_r} L_1,...,L_n$ be a method rule in MR_C. Then, $f_P(P)$ contains the rules:

$\{meth(Self,t_1,...,t_m)\leftarrow_{f_L(S_r)} f_L(Self{:}label_C)$, wtyped_*meth*_$C(Self,t_1,...,t_m), f_L(L_1),..., f_L(L_n)\mid$ $Self\in OBJ\}$, where OBJ is the set of object names in P.

(vi) Let r: $t_0{:}C(t_1,...,t_m)\leftarrow_{S_r} L_1,...,L_n$ be an object registration rule. Then, $f_P(P)$ contains the rule:

$C(t_0,t_1,...,t_m) \leftarrow f_L(S_r)$ wtyped_$C(t_1,...,t_m), f_L(L_1),...f_L(L_n)$.

(vii) Let $\perp \leftarrow L_1,...,L_n$ be a constraint in P. Then, $f_P(P)$ contains the constraint $\perp \leftarrow f_L(L_1),...,f_L(L_n)$.

(viii) The $<_R$ relation in $f_P(P)$ is defined as follows:

- If r,r' are rules created in (i),(ii),(iii) or (iv) then $r <> r'$.
- If r is a rule created in (i),(ii),(iii) or (iv) then $r' < r$ $\forall$ rule r' created in (v) or (vi).
- Let r,r' be rules created in (v). If r, r' have heads the literals $meth(o,t_1,...,t_m)$ and $meth'(o,t'_1,...,t'_m)$, and correspond to rules r_{ROL} and r'_{ROL} in P with $r_{ROL} < r'_{ROL}$ then $r < r'$.

The literals wtyped_$C(t_1,...,t_n)$ and wtyped_$meth$_$C(t_1,...,t_n)$ are called *well-typing literals*. A rule in $f_P(P)$ whose head is a *well-typing literal*, is called *well-typing rule*. An atom $o_0{:}C(o_1,...,o_n)$ (resp. $o_0.meth(o_1,...,o_n)$) is well-typed if objects $o_i, \forall i \leq n$, are members of the classes as indicated in the signature of the class C (resp. method *meth*). An *interpretation* of a *ROL* program P is a set $T \cup \sim F$ where T,F are disjoint sets of ground classical literals of P.

Definition 5.3.3 (well-typed atom, literal) Let P be a *ROL* program and I an interpretation of it.

- A ground atom $o{:}C(o_1,...,o_n)$ is *well-typed* w.r.t. I iff C has signature $C(C_1,...,C_n)$ and $o_i{:}C_i \in I$, $\forall i \leq n$.
- A ground atom $o.meth(o_1,...,o_n)$ is *well-typed* w.r.t. I iff there is a class C' such that $meth(C_1,...,C_n) \in SIG_C(meth)$, $o_i{:}C_i \in I$, $\forall i \leq n$ and if $C'_1,...,C'_m$ are all the direct subclasses of C' then $o{:}C' \in I$, $\sim o{:}C'_i \in I$ $\forall i \leq m$.
- A well-typed ground literal is a well-typed ground atom or its negation.

A member of a class C is a member of all superclasses of C. An object that is not a member of a class C is not member of any subclass of C.

Definition 5.3.4 (well-typed interpretation) Let P be an *ROL* program. An interpretation I of P is *well-typed* iff all classical literals in I are well-typed w.r.t. I and

(i) if $o{:}C(o_1,...,o_n) \in I$ then $o{:}C \in I$.

(ii) if $o{:}C \in I$ then $o{:}C' \in I$, for all classes C' with $C <_C C'$.

(iii) if $\neg o{:}C \in I$ (resp. $\sim o{:}C \in I$) then $\neg o{:}C' \in I$ (resp. $\sim o{:}C' \in I$), for all classes C' with $C' <_C C$.

To compute the reliable model of a *ROL* program, the reliable model of $f_P(P)$ is computed first.

Definition 5.3.5 (reliable model) Let P be a *ROL* program and M be an interpretation of P. M is an *o-model* (resp. *stable o-model, reliable model*) of P iff M' is an *r*-model (resp. stable *r*-model, reliable model) of $f_P(P)$ and $M = f_L^{-1}(M' - \{L \mid L \text{ is a well-typing literal}\})$.

Let r be a rule in $f_P(P)$ created in Steps (v) or (vi) of Definition 5.3.2. Note that the well-typing literal L_{wt} in $Body_r$ does not belong to the preliminary suspect set of r. This implies that *CWAs* and rules used in the derivation of L_{wt} are not "suspects" for constraint violations that depend on r.

The rules in $f_P(P)$ created in Steps (i),(ii),(iii) and (iv) of Definition 5.3.2 are reliable w.r.t. any interpretation because they have higher priority than any other rule in $f_P(P)$. This guarantees that every *o*-model of P is a well-typed interpretation of P.

Proposition 5.3.1 Let P be a *ROL* program. If M is an *o*-model of P then M is a well-typed interpretation of P.

Proof: (Steps in the proof are referring to Definition 5.3.2). From the well-typing rules added to $f_P(P)$ in Steps (ii) and (iii) and the well-typing literal in the body of a rule added to $f_P(P)$ in Steps (v) and (vi), it follows that all classical literals in M are well-typed. From the rules added to $f_P(P)$ in Step (i), it follows that if $o{:}C(o_1,\ldots,o_n) \in M$ then $o{:}C \in M$. From the rules added to $f_P(P)$ in Step (iv), it follows that if $o{:}C \in M$ then $o{:}C' \in M$ and if $\neg o{:}C' \in M$ (resp. $\sim o{:}C' \in M$) then $\neg o{:}C \in M$ (resp. $\sim o{:}C \in M$), for all classes C, C' such that $C <_C C'$.

Proposition 5.3.2 Let P be a *ROL* program. If M is a stable *o*-model of P then there is only one M' such that M' is a stable *r*-model of $f_P(P)$ and $M = f_L^{-1}(M' - \{L \mid L \text{ is a well-typing literal}\})$.

Proof: Assume that M' and M'' are r-models of $f_P(P)$ and $M=f_L^{-1}(M'-\{L|\ L$ is a well-typing literal$\})=f_L^{-1}(M''-\{L|\ L$ is a well-typing literal$\})$. Then, M' and M'' differ only on well-typing literals. However, this is not possible because of the rules added to $f_P(P)$ in Steps (ii) and (iii) of Definition 5.3.2.

Example 5.3.1: Let P be the *ROL* program of Example 5.2.1. Then, $f_P(P)$ is as follows:

(well_typing rules and literals are ignored)

Method Rules (created in Step (v) of Definition 5.3.2):

r_1: name(ann,Name)←student(ann,Name,Country,Credits).

r_2: citizenship(ann,Country)←student(ann,Name,Country,Credits).

r_3: full_time(ann)←student(ann,Name,Country,Credits), full_time_credits(ann,X),Credits≥X.

S_{r3}={full_time_credits(ann,X)}

r_4: full_time_credits(ann,9)←member_grad_stud(ann).

r_5: teaches(ann,Course)←TA(ann,Course).

r_6: full_time_credits(ann,6)←TA(ann,Course).

r_7: full_time_credits(ann,12)←member_foreign_grad_stud(ann).

Object-Registration Rules (created in Step (vi) of Definition 5.3.2):

r_8: member_foreign_grad_stud(X)←member_grad_stud(X), citizenship(X,Y),Y≠"USA".

S_{r8}={member_grad_stud(X), citizenship(X,Y)}

r_9: student(ann,"Ann","UK",12).

r_{10}: member_TA(ann).

Class Membership Definitions (created in Step (i) of Definition 5.3.2):

r_{11}: member_student(X)←student(X,Name,Country,Credits).

r_{12}: member_grad_stud(X)←grad_stud(X,Credits).

r_{13}: member_TA(X)←TA(X,Course).

r_{14}: member_foreign_grad_stud(X)←foreign_grad_stud(X).

Class Membership Relationships (created in Step (iv) of Definition 5.3.2):

r_{15}: member_student(X)←member_grad_stud(X).

r_{16}: ¬member_grad_stud(X)← ¬member_student(X).

r_{17}: member_grad_stud(X)← member_TA(X).

r_{18}: ¬member_TA(X)← ¬member_grad_stud(X).

r_{19}: member_grad_stud(X)←member_foreign_grad_stud(X).

r_{20}: ¬member_foreign_grad_stud(X)←¬member_grad_stud(X).

with $S_{r_i}=\{\}$, $\forall i \leq 11$ and $i \neq 3,8$ and $S_{r_i}=Body_{r_i}$, $\forall i>11$.

Constraints:

⊥←full_time_credits(X,Y),full_time_credits(X,Z),Y≠Z.

Rule Priorities:

$r_4<r_6$, $r_4<r_7$, $r_7<r_6$ and $r_i<r_j$, $\forall i \leq 10$ and $j>10$.

The reliable model of P is $RM_P=T_P \cup \sim F_P$ where:

T_P={ann:student("Ann","UK",12), ann:student, ann:grad_stud, ann:TA, ann:foreign_grad_stud, ann.name("Ann"), ann.citizenship("UK"), ann.full_time_credits(6), ann.full_time()}

and $F_P=HB_P-T_P-\{\text{ann.full\_time\_credits}(x) \mid x=9,12\}$.

Let P' be the *ROL* program that results if we eliminate $r_7<r_6$ from $<_R$ of P. Then, the reliable model of P' is $RM_{P'}=T_{P'} \cup \sim F_{P'}$ where:

$T_{P'}$={ann:student("Ann","UK",12), ann:student, ann:grad_stud, ann:TA, ann:foreign_grad_stud, ann.name("Ann"), ann.citizenship("UK")}

and $F_{P'}=HB_{P'}-T_{P'}-\{\text{ann.full\_time\_credits}(x) \mid x=6,9,12\}-\{\text{ann.full\_time}()\}$.

It is arguable that ann.full_time() should be true because even though it is unknown which of {ann.full_time_credits(6), ann.full_time_credits(12)} is true, the literal ann.full_time() is true in both cases. A similar argument applies against ordered logic [43], too. However, in [10] (reviewed in subsection 3.6.1), ann.full_time() is derived because the disjunction ann.full_time_credits(6)∨ann.full_time_credits(12) is added to the knowledge base.

5.4 Conclusions

An application of *RS* to deductive object-oriented databases (*DOODs*) is described. We present a simple but powerful logic, called reliable object logic (*ROL*). *ROL* models important aspects of the object-oriented paradigm such as object-identity, class hierarchy, multiple inheritance, multiple roles of an object and defaults. Each object of a class C has a set of attributes which are defined when the object is registered into the class. An *object registration atom* $t_0{:}C(t_1,...,t_n)$ indicates that object t_0 is a member of class C and has attribute values $t_1,...,t_n$. Each class has a set of methods which are applied to the objects of the class. A *method atom* $t_0{:}meth(t_1,...,t_n)$ indicates that the method *meth* is applied to the object t_0 with input-output parameters $t_1,...,t_n$. Object registration rules and method rules are used to register objects into classes and to compute the results of the application of methods to objects, respectively. Rule prioritization makes it possible to deal with default properties, exceptions and contradictions in the context of object-oriented programming.

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

We have presented two approaches, namely reliable semantics (*RS*) and contradiction-free semantics (*CFS*), for dealing uniformly with incomplete information, exceptions and contradictions in logic programs. Incomplete information is handled with the incorporation of classical negation. Exceptions are expressed (i) by using classical negation and rule prioritization or (ii) by using classical negation and negation be default. Contradiction is avoided by using the concepts of reliable rule and reliable closed-world assumption (*CWA*) in the first approach and *c*-unfounded literal in the second approach.

- *Incomplete information*

According to the closed-world reasoning, negative information $\sim A$ is true if A cannot be derived from the rules of the program. However, when the rules give only an incomplete definition of a predicate, closed-world reasoning cannot be applied for this predicate. It is possible that A is not derived from the program rules, not because A is false but because of missing information. Classical negation ($\neg$) allows us to reason in the absence of the domain-closure assumption for a set of predicates $Pred_{OW}$ (*OW* stands for *open-world*). Specifically, if A is an atom with predicate in $Pred_{OW}$ then A should be considered false only if $\neg A$ is derivable from the program. Closed-world reasoning can still be used for predicates with complete positive definitions.

- *Exceptions*

Allowing classical negation in the heads of rules is also useful in expressing exceptions to general rules. For example, the rule $\neg fly(X) \leftarrow penguin(X)$ expresses an exception to the general rule $fly(X) \leftarrow bird(X)$. However, to avoid contradiction, either the default literal $\sim penguin(X)$ should be added to the body of the general rule or the exception rule should be given higher priority than the

general rule. Even though both approaches can be expressed in the *RS* framework, the former approach is not modular since for each exception, a default literal should be added to the body of the general rule.

• *Contradiction*

Positive programs never cause contradiction. In contrast, an extended program can be contradictory, i.e., it is possible that both literals A and $\neg A$ are derived from the program. Let P be such a contradictory program. Our semantics, instead of simply noticing the contradiction, derive useful conclusions that are irrelevant to the contradiction. A characteristic of our approaches is that they do not only restrict inferences from the contradictory literals A and $\neg A$ but also from literals (*possible sources of the contradiction*) contributing to the derivation of A and $\neg A$. The preliminary suspect sets of the rules in the first approach and the contrapositive sets of the rules in the second approach indicate how far back in the derivation path the sources of the contradiction can be. Rule prioritization indicates our relative confidence in the conflicting rules. In our first approach, the priorities and the preliminary suspect sets of the rules are used in the definition of reliable rule and reliable *CWA*. A conclusion is considered irrelevant to contradiction only if it is derived from reliable rules and reliable *CWAs*. In our second approach, the priorities and the contrapositive sets of the rules are used in the definition of a *c*-unfounded literal w.r.t. a rule r. A conclusion L is considered irrelevant to contradiction only if it is derived from a rule r s.t. $\neg L$ is *c*-unfounded w.r.t. r.

RS and *CFS* are defined as the fixpoints of two monotonic operators. The model theoretic characterization of the *RS* (resp. *CFS*) of a program P is given by defining the *stable r-models* (resp. *stable c-models*) of P. Both *RS* and *CFS* are interpreted as the skeptical view of the world. Stable *r*-models and stable *c*-models of P are interpreted as alternative enlarged consistent belief sets standing for different possible views of P. We have shown that *RS* (resp. *CFS*) is a stable *r*-model (resp. stable *c*-model) of P and coincides with the intersection of all the stable *r*-models (resp. stable *c*-models) of P. Because of this property, proof procedures for capturing skeptical reasoning are related to one stable model in the semantics and imply validity in all stable models.

In section 4.5, the *SLCF*-resolution (linear resolution with selection function for contradiction-free semantics) for computing answers for extended programs with rule prioritization is presented. The *SLCF*-resolution is shown to be sound and complete w.r.t. *CFS*.

In subsections 2.4.1 and 4.4.1, we showed that *RS* and *CFS* generalize the well-founded semantics for normal programs [76]. In subsection 2.4.3, we showed that *RS* generalizes the extended well-founded semantics for non-contradictory extended programs [54]. *RS* and *CFS* can express uniformly various recent proposals for reasoning with contradictions in logic programs:

• **Ordered Logic [24, 43] (subsections 2.4.7 and 4.4.1).** Ordered logic can be expressed in the *RS* and *CRS* framework, by taking S_r={} ∀ rule r. However, S_r={} ∀ rule r, expresses only exceptions and not reliability. Thus, ordered logic can only handle contradictions caused by exceptions to general rules. Since ordered logic does not support negation by default, exceptions can only be expressed by giving them higher priority than the general rules. When complementary literals A and $\neg A$ are derived from conflicting rules with priorities that cannot be compared, ordered logic blocks the inferences from the contradictory literals A and $\neg A$ but not from the possible sources of the contradiction.

• **Conservative Vivid Logic [77] (subsection 2.4.5).** Conservative vivid logic behaves similarly to ordered logic when $<_R$={}. Thus, when complementary literals A and $\neg A$ are derived from conflicting rules, derivations from A and $\neg A$ are blocked. Yet, possible sources of this contradiction will still be used for further inferences.

• **Relevant Expansion [79] and Contradiction Removal Semantics [52, 55] (subsection 2.4.4).** Both approaches can be expressed in the *RS* framework by taking S_r=$Body_r$ ∀ rule r and $<_R$={}. In contrast to *RS*, these approaches do not support rule prioritization and they are applicable only for a special kind of extended programs whose inconsistencies can be removed by retracting some *CWAs*.

• **Generalized Stable Model Semantics [28] and Strong Belief Revision Semantics [79] (subsection 4.4.2).** The strong belief revision semantics extends the generalized stable model semantics from 2-valued to 3-valued logic. These approaches can be expressed in the *CFS*

framework by taking $C_r=Body_r$ $\forall$ rule r and $<_R=\{\}$. Thus, all contrapositives of the original rules are considered to be valid. In *CFS*, the use of contrapositives can be controlled through the contrapositive sets C_r. Strong belief revision semantics and generalized stable model semantics do not support rule prioritization. Moreover, they are not defined for every contradictory extended program.

The main contributions of *RS* and *CFS* are summarized as follows:

- **Broader Domain.** They cover a broader domain of logic programs than semantics proposed earlier.
- **Uniform Framework.** They provide a framework for dealing uniformly with incomplete information, exceptions and contradictions in logic programs.
- **Generalization.** They extend semantics proposed earlier for non-contradictory logic programs. They express uniformly various proposals of reasoning with contradictions in logic programs.
- **Universality.** They are well-defined for every contradictory program, i.e., they derive useful conclusions that are irrelevant to the inconsistency.
- **Uniqueness.** They coincide with the intersection of all stable models of the program. This implies that the skeptical meaning of the program can be computed without the expensive computation of all stable models.
- **Efficiency.** They can be computed in polynomial time w.r.t. the size of the program, when the Herbrand base is finite.

In Chapter 3, the *modular reliable semantics* (*MRS*) for prioritized modular logic programs (*PMPs*) is presented. A *PMP* consists of a set of modules and a partial order $<_{def}$ on the predicate definitions. By combining a set of modules to a single program, information may be obtained that is not derivable from a single module. This way, large deductive databases can be developed in parts and later be combined. Also, cooperative problem solving is possible by combining the knowledge of individual agents. We assume that the knowledge of an agent is encapsulated in a module. The problem of combining the knowledge of different agents is non-trivial because individual agents usually hold conflicting views on their domain of expertise. *MRS* provides a

framework in which only agreeing results are exported from the modules but individual modules can still maintain their internal beliefs. The concepts of local literal and reliable indexed literal are developed. A local literal $M\#L$ represents the internal belief of agent M for the truth value of literal L. A reliable indexed literal $\{M_1,...,M_n\}:L$ represents the agreeing belief of agents $M_1,...,M_n$ for the truth value of literal L.

An application of *RS* to deductive object-oriented databases (*DOODs*) is described in Chapter 5. We present a simple but powerful logic, called *reliable object logic* (*ROL*). *ROL* models important aspects of the object-oriented paradigm such as object-identity, class hierarchy, multiple inheritance, multiple roles of an object and defaults. Rule prioritization can be used to give higher priority to local than inherited rules, in case of conflict. Thus, rule prioritization makes it possible to deal with default properties and exceptions in the context of object-oriented programming. Rule prioritization can also be used for the avoidance of general contradictions.

The modular reliable semantics for *PMPs* can also have application in *DOODs* since each object class can be seen as a module. When classes are independently developed and then put together, conflicts are likely. Since the class internal details may be unknown, we would like to derive reliable information without class internal revision. The $<_{def}$ relationship between predicate definitions can be used to express that in case of conflict, results from subclasses have higher priority than results from superclasses. Indexed literals can be useful in the case of *multiple inheritance* because the derivation of a literal from only one superclass or a set of superclasses of an object can be queried. The "*point of view*" notion of multiple inheritance, proposed in [15], can also be implemented using indexed literals $S:L$, where S is a set of classes representing the "*point of view*" of a class C and an object o [15]. The details of the representation of a *DOOD* in the *MPS* framework remain to be investigated.

Rule prioritization in *RS* can express explicit priorities on the rules. For example, it can indicate that rule r_1: $\neg fly(X) \leftarrow penguin(X)$ has higher priority than rule r_2: $fly(X) \leftarrow bird(X)$. Touretzky [73, 74] claimed that prioritization among defaults can be obtained using implicit specificity information even when no explicit rule prioritization exists. For example, since every

penguin is a bird, rule r_1 is implicitly more specific than rule r_2. Note that explicit priorities supplied by the user will still be useful to decide among conflicting rules which cannot be ordered using implicit specificity information. Geerts and Vermeir [25] presented an approach in which implicit and explicit priorities are combined in the framework of ordered logic. Future work should be concerned with the combination of implicit and explicit priorities in our frameworks.

In subsection 3.6.1, we discussed work on combining local deductive databases $P_1,\ldots,P_n$. In [10], when a constraint $\bot \leftarrow L_1,\ldots,L_n$ is violated in the combined database P, the disjunction $L_1 \vee \ldots \vee L_n$, is added and the rules with head L_i, $i \leq n$, are removed from P. Thus, P is a consistent disjunctive program containing more information than the *RS* of $P' = P_1 \cup \ldots \cup P_n$. For example, if P contains the rules $A \leftarrow L_1$, ..., $A \leftarrow L_n$ and A does not appear in any other rule in P then A will be evaluated as true in [10] but as unknown in the *RS* of P'. Yet, a disadvantage of the approach in [10] is that literals $L_1,\ldots,L_n$ may be based on unreliable information which will be evaluated as true. In the *RS* of P', not only the literals L_i, $i \leq n$, are considered "suspect" for the violation of the constraint but also the literals used in the derivation of L_i, $i \leq n$. A combination of the approaches in [10] and *RS* should be investigated.

When the constraints of an application are modified, it is possible that the application programs do not satisfy the new constraints. Our semantics can be useful to applications whose constraints change frequently. Though the application programs may violate the constraints, our semantics will represent only the reliable information derived from them. Future work should include the investigation of specific applications and their requirements.

APPENDICES

APPENDIX A

PROOFS OF SECTION 4.3

Proof of Proposition 4.3.1: We will show that W_P is a monotonic operator. Let I,J be interpretations of P s.t. $I \subseteq J$. $T(I) \subseteq T(J)$ follows from the fact that if a literal L is c-unfounded w.r.t. a rule r and I then L is c-unfounded w.r.t. a rule r and J. Since $F(I) \subseteq F(J)$ and coh is a monotonic operator, W_P is a monotonic operator and $\{I_a\}$ is a monotonically increasing sequence of interpretations w.r.t. $\subseteq$.

We will prove by induction that for all a, there is no literal K s.t. $\{K,\neg K\} \subseteq I_a$. This is true for $a=0$. Assume that it is true for ordinals $<a$. We will prove that it is true for a. Assume first that $a=b+1$ is a successor ordinal. Assume that there is literal K s.t. $\{K,\neg K\} \subseteq I_a$. This implies that there is a literal $L \in I_a$ derived from a rule r in P s.t. L is used in the derivation of K or $\neg K$ and $\neg L$ is not c-unfounded w.r.t. a rule r and I_a. However, this is a contradiction because when a classical literal L is derived from a rule r then $\neg L$ is c-unfounded w.r.t. a rule r and I_a (Def. 4.3.1). Thus, I_a is consistent.

Let a be a limit ordinal and assume that there is literal K s.t. $\{K,\neg K\} \subseteq I_a$. Then, there is a successor ordinal $b+1<a$ s.t. $\{K,\neg K\} \subseteq I_{b+1}$. This is a contradiction because of the inductive hypothesis. So, I_a is consistent for all a.

We will prove by induction that for all a, there is no literal L s.t. $L \in I_a$ and $\sim L \in I_a$. It is true for $a=0$. Assume that it is true for ordinals $<a$. We will prove that it is true for a. Assume first that $a=b+1$ is a successor ordinal. We will prove that there is no literal L s.t. $L \in I_a$ and $\sim L \in I_a$. This is true for $a=0$. Assume this is true for ordinals $<a$. Let S be any set of classical literals that has a

non-empty intersection with $T(I_b)$. Choose the smallest c s.t. I_{c+1} has a non-empty intersection with S. Note that $c \leq b$. Let $K \in I_{c+1} \cap S$. Then K is derived from a rule r in $exp(P)$ s.t. $Body_r \subseteq I_c$. From hypothesis, there is no literal $K \in Body_r$, s.t. $\sim K \in I_b$. Moreover, from the way r is defined, there is no classical literal K in $Body_r$ s.t. $K \in S$. So, S is not c-unfounded w.r.t. I_b. This implies that $T(I_b) \cap F(I_b) = \emptyset$. Moreover, there is no classical literal L s.t. $L \in T(I_b)$ and $\neg L \in T(I_b)$, because I_a does not violate any constraint. So, there is no literal L s.t. $L \in I_a$ and $\sim L \in I_a$. Let a be a limit ordinal and assume that there is L s.t. $L \in I_a$ and $\sim L \in I_a$. Then, there is a successor ordinal $b+1<a$ s.t. $L \in I_{b+1}$ and $\sim L \in I_{b+1}$. This is a contradiction because of the inductive hypothesis.

I_a is a coherent interpretation, for all a, because of the *coh* operator in the definition of W_P. Proposition 4.3.1 follows. ◊

Proof of Proposition 4.3.2: From Proposition 4.3.1, I_d is a consistent, coherent interpretation. Let r be a rule in $exp(P)$. We will show that r is c-true w.r.t. I_d.

(i) If $I_d(Body_r)=1/2$ and $I_d(Head_r)=0$ then $I_d(\neg Head_r)=1$ because otherwise $I_d(Head_r)=1/2$.

(ii) If $I_d(Body_r)=1$ and $I_d(Head_r)=1/2$ then $\neg Head_r$ is not c-unfounded w.r.t. r and I_d because otherwise, from the definition of $T(I_d)$, $I_d(Head_r)=1$.

(iii) If $I_d(Body_r)=1$ and $I_d(Head_r)=0$ then $\neg Head_r$ is not c-unfounded w.r.t. r and I_d because otherwise, from the definition of $T(I_d)$, $I_d(Head_r)=1$. Since $\neg Head_r$ is not c-unfounded w.r.t. r and I_d, it follows that $I_d(\neg Head_r)=1$ because otherwise, from the definition of $F(I_d)$, $I_d(Head_r)=1/2$.

(iv) In all the other cases, $I_d(Head_r) \geq I_d(Body_r)$. ◊

Proof or Proposition 4.3.3: Let P be a propositional *PEP*. The algorithm *CFM*(program P) returns the contradiction-free model of P. To compute $F(I)$ and the set of c-unfounded literals w.r.t. a rule r and I, the complement set is constructed first, as in [76]. The complexity of the algorithm is $O(|HB_P|^* |R_{exp(P)}|^2)$.

```
CFM(PEP program P)
{ new_I={};
  repeat
    I=new_I;
    repeat          /* Step 1: compute T(I) */
      for each rule r of exp(P) do
        if Body_r ⊆new_I and c-unfounded(¬Head_r, r, I) then add Head_r to new_I;
      end /* for */
    until no change in new_I;

    compl_F={};
    repeat          /* Step 2: compute F(I) */
      for each rule r of exp(P) do
        if I(Body_r)≠0 and all body classical literals of r are true w.r.t. I
        then add Head_r to compl_F;
      end /* for */
    until no change in compl_F;
    for each L∈HB_P do      /* Step 3 */
      if L∉compl_F then add ~L to new_I;
    end /* for */

    new_I = coh(new_I);     /* Step 4: Compute coh(T(I)∪~F(I)) */
  until I=new_I;
  return(I);
}

c-unfounded(literal L, rule r, interpretation I) /* returns TRUE is L is c-unfounded w.r.t. r and I */
{  compl_U={};
   repeat
     for each rule r' of exp(P) s.t. r' ≠r do
       if I(Body_r')≠0 and for each L∈Body_r' either L is true w.r.t. compl_U or L∉C_r'
       then add Head_r' and ~¬Head_r' to compl_U;
     end /* for */
   until no change in compl_U;
   if L∉compl_U then return(TRUE); else return(FALSE);
 }
```

The complexity of computing if a literal L is c-unfounded w.r.t. a rule r and interpretation I is $|R_{exp(P)}|$ [19]. So, the complexity of Step 1 is $|R_{exp(P)}|^2$. The complexity of Step 2 is $|R_{exp(P)}|$ [19]. The complexity of Steps 3 and 4 is $|HB_P|$. Since $\{I_a\}$ is a monotonically increasing sequence w.r.t. $\subseteq$, the total number of iterations until I=*new_I*, is less than $|HB_P|$. So, the complexity of the algorithm **CFM**(P) is $O(|HB_P|*|R_{exp(P)}|^2)$. ◊

Proof of Proposition 4.3.4: Let *CFM* be the contradiction-free model of P and let P' be equal to $exp(P)/_c CFM$. From Proposition 4.3.2, *CFM* is a *c*-model of P. So, it is enough to show that $CFM=least_v(P')$. Let $least_v(P')=T\cup\sim F$, where T, F are sets of classical literals. Let $I_a=T_a\cup\sim F_a$, where T_a, F_a are sets of classical literals and $CFM = I_d$. First, we will prove by induction that $T_b\cup\sim F_b \subseteq T\cup\sim F$, $\forall b\leq d$. It is true that $T_0\subseteq T$ and $F_0\subseteq F$. Suppose that $T_a\subseteq T$ and $F_a\subseteq F$, $\forall a<b$. If b is a limit ordinal then $T_b\subseteq T$ and $F_b\subseteq F$ since $I_b= \cup\{I_a|\ a<b\}$. Assume therefore that $b=a+1$. It is true that $T_{I_a}{}^{\uparrow 0}(\emptyset)\subseteq T$. Assume that $T_{I_a}{}^{\uparrow a'}(\emptyset)\subseteq T$, we will show that $T_{I_a}{}^{\uparrow a'+1}(\emptyset)\subseteq T$. Let $L\in T_{I_a}{}^{\uparrow a'+1}(\emptyset)$. Then, $\exists\ r: L\leftarrow L_1,...,L_n$ in $\exp(P)$ s.t. $\neg L$ is *c*-unfounded w.r.t. r and I_a and $\forall i\leq n$ either (i) $L_i\in I_a$ or (ii) L_i is a classical literal and $L_i\in T_{I_a}{}^{\uparrow a'}(\emptyset)$. Since $I_a\subseteq T\cup\sim F$ and $L\in CFM$, there is a rule $L\leftarrow L'_1,...,L'_m$ in P' where $L'_1,...,L'_m$ are all the classical literals in $\{L_1,...,L_n\}$. From the facts $T_{I_a}{}^{\uparrow a'}(\emptyset)\subseteq T$, $I_a\subseteq T\cup\sim F$ and the definition of $least_v(P')$, it follows that $L\in T$. This implies that $T(I_a)=T_b\subseteq T$.

Now, we will show that $F_b\subseteq F$. Since $F_b= \neg T_b\cup F(I_a)$, it is enough to show that $\neg T_b\subseteq F$ and $F(I_a)\subseteq F$. If $L\in\neg T_b$ then $\neg L\in CFM$ and from Step (ii) of Def. 4.3.3, $L\in F$. Consequently, $\neg T_b\subseteq F$. For all rules $H\leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in $exp(P)$ (L_i, L'_i are classical literals) with $H\in F(I_a)$ either $\exists i\leq m$, $L'_i\in F(I_a)\cup F_a$ or $\exists j\leq n$, $L_j\in T_a$. This implies that, for each rule $H\leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in $exp(P)$ with $H\in F(I_a)$ either there is a corresponding rule $H\leftarrow A_1,...,A_k$ in P' (from Steps (iv) and (v) of Def. 4.3.3) with $A_i\in F(I_a)\cup F$ for an $i\leq k$ or there is no corresponding rule in P' (from the Steps (i) and (ii) of Def. 4.3.3). Note that, no rule $H\leftarrow u$ is added to P' (from Step (iii) of Def. 4.3.3) because H is false w.r.t. *CFM*. So, for each rule $H\leftarrow A_1,...,A_k$ in P' with $H\in F(I_a)\cup F$, $\exists i\leq k$ such that $A_i\in F(I_a)\cup F$. From the definition of $least_v(P')$, it follows that $F(I_a)\subseteq F$. Consequently, $F_b\subseteq F$. So, we proved that $T_d\subseteq T$ and $F_d\subseteq F$.

We will show that $T\subseteq T_d$. Let a be the first ordinal s.t. there is a literal $L\notin T_d$ and $\Psi_{P'}{}^{\uparrow a+1}(\emptyset)(L)=1$. Then, there is a rule r: $L\leftarrow A_1,...,A_k$ in P' with $\Psi_{P'}{}^{\uparrow a}(\emptyset)(A_i)=1$, $\forall i\leq k$. This implies that there is a rule in $exp(P)$ whose body literals are true w.r.t. *CFM*. Since $L\notin T_d$, it

follows that $\neg L \in T_d$ or L is unknown w.r.t. *CFM*. If $\neg L \in T_d$ then from Step (ii) of Def. 4.3.3, $L \notin T$ which is a contradiction. If L is unknown w.r.t. *CFM* then the rule r should not exist in P' because of the Step (iii) of Def. 4.3.4 and the fact that all of the body literals of r are true w.r.t. $\Psi_{P'}{}^{\uparrow a}(\emptyset)$ and thus w.r.t. *CFM*. So, $L \in T_d$ and consequently $T \subseteq T_d$.

We will show that $F \subseteq F_d$. Let $F_{coh}=\{H \mid \neg H \in T_d\}$. $F_{coh} \subseteq F_d$ because *CFM* is a coherent interpretation. For all rules $H \leftarrow A_1,...,A_k$ in P' with $H \in F-F_{coh}$, there is $i \leq k$ such that $A_i \in F$. This implies that for each rule $H \leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in $exp(P)$ (L_i, L'_i are classical literals) with $H \in F-F_{coh}$ either (i) $\exists i \leq m$, $L'_i \in F$ (from Steps (iv) and (v) of Def. 4.3.3) or (ii) $\exists j \leq n$, $L_j \in T_d$ (from Step (i) of Def. 4.3.4). Since $\mathbf{F}(I_d)$ is the maximum set that satisfies the property satisfied by $F-F_{coh}$, $F-F_{coh} \subseteq \mathbf{F}(I_d)$. So, $F \subseteq F_d$.

Consequently, $RM=T_d \cup \sim F_d = T \cup \sim F = least_v(P/_r RM)$. ◊

Proof of Proposition 4.3.5: Let *CFM* be the contradiction-free model of P. From Proposition 4.3.4, *CFM* is a stable c-model of P. So, it is enough to show that if M is a stable c-model of P then $RM \subseteq M = least_v(exp(P)/_r M)$. Let $M=T \cup \sim F$, where T, F are sets of classical literals. Let $I_a=T_a \cup \sim F_a$, where T_a, F_a are sets of classical literals and $RM = I_d$. We will show by induction that $I_b \subseteq T \cup \sim F$, $\forall b \leq d$. It is true that $T_0 \subseteq T$ and $F_0 \subseteq F$. Suppose that $T_a \subseteq T$ and $F_a \subseteq F$, $\forall a<b$. If b is a limit ordinal then $T_b \subseteq T$ and $F_b \subseteq F$ since $I_b = \cup\{I_a \mid a<b\}$. Assume therefore that $b=a+1$. It is true that $\mathrm{T}_{I_a}{}^{\uparrow 0}(\emptyset) \subseteq T$. Assume that $\mathrm{T}_{I_a}{}^{\uparrow a'}(\emptyset) \subseteq T$, we will show that $\mathrm{T}_{I_a}{}^{\uparrow a'+1}(\emptyset) \subseteq T$. Let $L \in \mathrm{T}_{I_a}{}^{\uparrow a'+1}(\emptyset)$. Then, $\exists\, r: L \leftarrow L_1,...,L_n$ in P s.t. $\neg L$ is c-unfounded w.r.t. r and I_a and $\forall i \leq n$ either (i) $L_i \in I_a$ or (ii) L_i is a classical literal and $L_i \in \mathrm{T}_{I_a}{}^{\uparrow a'}(\emptyset)$. Since $I_a \subseteq M$, it follows that $\neg L$ is c-unfounded w.r.t. r and M. From the facts that M is an c-model of P, $I_a \subseteq M$, $\mathrm{T}_{I_a}{}^{\uparrow a'}(\emptyset) \subseteq T$ and $\neg L$ is c-unfounded w.r.t. r and M, it follows that $L \in T$. So, $\mathbf{T}(I_a)=T_b \subseteq T$.

Now, we will show that $F_b \subseteq F$. Since $F_b = \neg T_b \cup \mathbf{F}(I_a)$, it is enough to show that $\neg T_b \subseteq F$ and $\mathbf{F}(I_a) \subseteq F$. If $L \in \neg T_b$ then $\neg L \in M$ and from Step (ii) of Def. 4.3.3, $L \in F$. Consequently, $\neg T_b \subseteq F$. For all rules $H \leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in P (L_i, L'_i are classical literals) with $H \in \mathbf{F}(I_a)$ either $\exists i \leq m$, $L'_i \in \mathbf{F}(I_a) \cup F_a$ or $\exists j \leq n$, $L_j \in T_a$. This implies that, for each rule $r: H \leftarrow L'_1,...,L'_m,\sim L_1,..,\sim L_n$ in P

with $H \in \mathbf{F}(I_a)$ either there is a corresponding rule $H \leftarrow A_1,\ldots,A_k$ in $exp(P)/_c M$ (from Steps (iv) and (v) of Def. 4.3.3) with $A_i \in \mathbf{F}(I_a) \cup F$ for an $i \leq k$ or there is no corresponding rule in $exp(P)/_c M$ (from Steps (i) and (ii) of Def. 4.3.3). Note that, r is not transformed into $H \leftarrow u$ in $exp(P)/_c M$ in Step (iii) of Def. 4.3.3 because the facts $I_a \subseteq M$ and $least_v(exp(P)/_c M)=M$ imply that $\exists i \leq m$, $L'_i \notin T$ or $\exists j \leq n$, $L_j \notin F$. From the definition of $least_v(exp(P)/_c M)$, it follows that $L \in F$. So, $F_b \subseteq F$ and thus $T_d \subseteq T$ and $F_d \subseteq F$. Consequently, $RM=T_d \cup \sim F_d \subseteq T \cup \sim F=M$. ◊

APPENDIX B

PROPOSITIONAL PROOF THEORY AND PROLOG META-INTERPRETERS

B.1 Contradiction-Free Semantics

Let P be a propositional *PEP* and L a literal. *CFS*_deriv(L,{}) returns SUCC (resp. FAIL) iff L is true (resp. false or unknown) w.r.t. *CFS*. The routine distance_in_list(L, *Literal_list*, *Distance*) returns SUCC if L is a member of the input *Literal_list*, i.e., *Literal_list* $=L_1,\ldots, L_i, L, L_{i+1},\ldots, L_n$. The output *Distance* is ZERO if all literals $L,L_{i+1},\ldots,L_n$ are default literals or have the form $c(K)$.

***CFS*_deriv**(classical literal L, ancestor list *Anc*)
{ **if** L unifies with $\sim\neg L'$ or $\neg\neg L'$ or $\sim\neg L'$ **then** return(*CFS*_deriv(L', $\{L\}\cup Anc$));
  if $\sim L \in Anc$ **then** return (FAIL); **endif**
  if distance_in_list(L, *Anc*, *Dist*)=SUCC **then**
    if *Dist*= ZERO and L is a default literal **then** return(SUCC); **else** return(FAIL); **endif**
  endif
  if L unifies with $\sim L'$ and *CFS*_deriv($\neg L'$, $\{L\}\cup Anc$)=SUCC **then** return(SUCC); /* coherence */
  endif
  if L unifies with $\sim L'$ **then** /* L is a default literal */
    for every rule r: $L' \leftarrow Body_r \in exp(P)$ **do**
      if not($\exists K \in Body_r$ s.t. *CFS*_deriv($\sim K$, $\{L\}\cup Anc$)=SUCC) **then** return(FAIL); **endif**
    endfor
    return(SUCC);
  else /* L is a classical literal */
    flag=RETRY;
    for every rule r: $L \leftarrow Body_r \in exp(P)$ and **if** flag=RETRY **do**
      flag=SUCC;
      for every literal $K \in Body_r$ **do**
        if *CFS*_deriv(K, $\{L\}\cup Anc$)=FAIL **then** flag=RETRY; break; **endif**
      endfor
    endfor
    if flag=SUCC and c-unfounded($\neg L$, r, $\{L\}\cup Anc$)=SUCC **then** return(SUCC);
    else return(FAIL);
    endif

```
  endif
}

c-unfounded(literal L, rule label r, ancestor list Anc)
/* return SUCC if L is unfounded w.r.t. r and CFS */
{
  if L unifies with ¬¬L' then return(c-unfounded(L', r, Anc));
  if L unifies with ~L' then return(c-unfounded(¬L', r, Anc));
  if distance_in_list(c(L), Anc, Dist)=SUCC then
     if Dist=ZERO then return(SUCC); else return(FAIL); endif
  endif
  for every rule r': L ← Body_r' s.t. r' ⊀ r do
     if not (∃K∈ C_r' s.t. c-unfounded(K, r, {c(L)}∪Anc)=SUCC or
           ∃K∈ Body_r' such that CFS_deriv(~K, {c(L)}∪Anc)=SUCC)
     then return(FAIL);
     endif
  endfor
  return(SUCC);
}
```

The following Prolog program works as a *CFS* inference engine for extended programs with rule prioritization. P is supposed to be already expanded and any rule r of $exp(P)$ is stored as a prolog fact: rule(r, $Head_r$ <- $Body_r$, C_r). Let G be a sequence of literals. The call *CFS_derive*(G,{}) succeeds (resp. fails) iff L is true (resp. false or unknown) in the *CFS* of P. If a literal L succeeds (resp. fails) as an intermediate result in the evaluation of *CFS_derive*(G,{}) then this result is stored in the relation *result*(*literal, value*) by asserting the fact *results*(*L,succ*) (resp. *results*(*L,fail*)).

```
CFS_derive(true,_):- !.
CFS_derive(~true,_):- !, fail.
CFS_derive(~((G1,G2)),Anc) :- CFS_derive(~G1,Anc).
CFS_derive(~((G1,G2)),Anc) :- !, CFS_derive(~G2,Anc).
CFS_derive((G1,G2),Anc) :- !, CFS_derive(G1,Anc), CFS_derive(G2,Anc).
CFS_derive(~(~G),Anc) :- !, CFS_derive(G,Anc).
CFS_derive(-(-G),Anc) :- !, CFS_derive(G,Anc).
CFS_derive(G,_):- results(G,succ), !.
CFS_derive(G,_) :- results(G,fail), !, fail.
CFS_derive(~G,Anc) :- member(G,Anc), !, fail.
CFS_derive(G,Anc) :- member(~G,Anc), !, fail.
CFS_derive(~G,_) :- results(G,succ), !, fail.
CFS_derive(G,_) :- results(~G,succ), !, fail.
CFS_derive(G,Anc) :- distance_in_list(G,Anc,F,Dist), !, ((Dist=zero, G=(~_))->true).
```

```
CFS_derive(~G,Anc) :- CFS_derive(--G,[~G|Anc]), !, assert(results(~G,succ)).
CFS_derive(~G,Anc):- !, findall(Body, rule(_,G<-Body,_), List),
                        (all_false(List,[~G|Anc])-> assert(results(~G,succ));
                                                    assert(results(~G, fail)), !, fail).
CFS_derive(G,Anc) :- ((rule(Label,G<-Body,_), CFS_derive(Body,[G|Anc]),
                         c_unfounded(--G,Label,[G|Anc]))-> assert(results(G,succ));
                                                          assert(results(G,fail)), !, fail).


% all_false(List,Anc) succeeds iff all goals in List are false w.r.t. CFS.

all_false([],_):- !.
all_false([G|List],Anc):- CFS_derive(~G,Anc), all_false(List,Anc).


% c_unfounded(L,Label,Anc) succeeds iff literal L is c-unfounded w.r.t. the rule with label Label
% and the CFS.

c_unfounded($,_,_):- !, fail.
c_unfounded(true,_,_):- !, fail.
c_unfounded((G1,G2),Label,Anc) :- c_unfounded(G1,Label,Anc).
c_unfounded((G1,G2),Label,Anc) :- !, c_unfounded(G2,Label,Anc).
c_unfounded(~G,Label,Anc) :- !, c_unfounded(--G,Label,Anc).
c_unfounded(--(--G),Label,Anc):- !, c_unfounded(G,Label,Anc).
c_unfounded(G,Label,_):- results(~G,succ), !.
c_unfounded(G,Label,_) :- results(G,succ), !, fail.
c_unfounded(G,Label,Anc) :- distance_in_list(u(G),Anc,F,Dist),!,(Dist=zero->true).
c_unfounded(G,Label,Anc) :- findall(r(Label1,G<-Body,C), (rule(Label1,G<-Body,C),
              not less_priority(Label1,Label)), List), all_c_unfounded(List,Label,[u(G)|Anc]).


% all_c_unfounded(Rule_list,Label,Anc) succeeds iff for all rules r in Rule_list one of the
% following is true: (i) a literal in the body of r is c-unfounded w.r.t. the rule with label Label and
% the CFS or (ii) a literal in the body of r is false w.r.t. CFS.

all_c_unfounded([],Label,_):- !.
all_c_unfounded([r(_,G<-Body,C)|List],Label,Anc):- all_members(Body,C,Body1),
                        c_unfounded(Body1,Label,Anc), all_c_unfounded(List,Label,Anc).
all_c_unfounded([r(_,G<-Body,_)|List],Label,Anc):- CFS_derive(~Body,Anc),
                                                   all_c_unfounded(List,Label,Anc).

% distance_in_list(L, Literal_list, Distance) succeeds iff L is a member of the input Literal_list,
% i.e., Literal_list =L_1,..., L_i, L, L_i+1,..., L_n. The output Distance is zero iff all literals
% L,L_i+1,...,L_n are default literals or have the form c(K). Otherwise, it is non_zero.

distance_in_list(G,[G|Anc],F,zero) :- var(F),!.
distance_in_list(G,[G|Anc],n_z,non_zero) :- !.
distance_in_list(G,[~_,~X|Anc],F,Dist):- !, distance_in_list(G,[~X|Anc],F,Dist).
```

```
distance_in_list(G,[~_,X|Anc],_,Dist):- !, distance_in_list(G,[X|Anc],n_z,Dist).
distance_in_list(G,[_,~X|Anc],_,Dist):- !, distance_in_list(G,[~X|Anc],n_z,Dist).
distance_in_list(G,[u(_),u(X)|Anc],F,Dist):- !, distance_in_list(G,[u(X)|Anc],F,Dist).
distance_in_list(G,[u(_),X|Anc],_,Dist):- !, distance_in_list(G,[X|Anc],n_z,Dist).
distance_in_list(G,[_,u(X)|Anc],_,Dist):- !, distance_in_list(G,[u(X)|Anc],n_z,Dist).
distance_in_list(G,[_,X|Anc],F,Dist):- !, distance_in_list(G,[X|Anc],F,Dist).

% all_members(G,C,Members) succeeds if Members is a list of literals s.t. if L is the ith literal in
% G and L is in C then L is the ith literal in Members. Otherwise, the ith literal in Members is $.

all_members((G1,G2),C,Members):- !, all_members(G1,C,Members1),
                    all_members(G2,C,Members2), Members=(Members1,Members2).
all_members(G,C,Members):- member(G,C)-> Members=G; Members=$.


member(X,[X|_]) :- !.
member(X,[_|L]) :- member(X,L).

append([],L,L).
append([X|L1],L2,[X|L3]):- append(L1,L2,L3).
```

B.2 **Reliable Semantics**

The following Prolog program works as an *RS* inference engine for extended programs with rule prioritization. Any rule r in P is stored as a prolog fact: rule(r, $Head_r$ <– $Body_r$, S_r). Let G be a sequence of literals. The call *RS_derive*(G,{}) succeeds (resp. fails) iff L is true (resp. false or unknown) in the *RS* of P. If a literal L succeeds as an intermediate result in the evaluation of *CFS_derive*(G,{}) then this result is stored in the relation *succeeds*(*literal*) by asserting the fact *succeeds*(L). To simplify the Prolog program, we assume that $S_r = Body_r$ for every rule r.

```
RS_derive(true,_):- !.
RS_derive(~true,_):- !, fail.
RS_derive(~((G1,G2)),Anc) :- RS_derive(~G1,Anc).
RS_derive(~((G1,G2)),Anc) :- !, RS_derive(~G2,Anc).
RS_derive((G1,G2),Anc) :- !, RS_derive(G1,Anc), RS_derive(G2,Anc).
RS_derive(~(~Lit),Anc) :- !, RS_derive(Lit,Anc).
RS_derive(--(--Lit),Anc) :- !, RS_derive(Lit,Anc).
RS_derive(Lit,_):- succeeds(Lit), !.
RS_derive(~Lit,Anc) :- member(Lit,Anc), !, fail.
RS_derive(Lit,Anc) :- member(~Lit,Anc), !, fail.
RS_derive(~Lit,_) :- succeeds(Lit), !, fail.
RS_derive(Lit,_) :- succeeds(~Lit), !, fail.
RS_derive(Lit,Anc) :- distance_in_list(Lit,Anc,_,Dist),!,((Dist=zero, Lit=(~_))->true).
```

```
RS_derive(~Lit,Anc) :- literal_unreliable(~Lit,[~Lit|Anc]), !, fail.
RS_derive(~Lit,Anc) :- RS_derive(--Lit,[~Lit|Anc]), !, assert(succeeds(~Lit)).
RS_derive(~Lit,Anc):- !, findall(Body, rule(_,Lit<-Body,_),List)
                         (all_false(List,[~Lit|Anc])-> assert(succeeds(~Lit)); fail).
RS_derive(Lit,Anc) :- rule(Rule,Lit<-Body,_), RS_derive(Body,[Lit|Anc]),
                        rule_unreliable(Rule,[Lit|Anc])-> fail; assert(succeeds(Lit)).

all_false([],_):- !.
all_false([G|List],Anc):- RS_derive(~G,Anc), all_false(List,Anc).
```

% *literal_unreliable*(*DefLit*,*Anc*) succeeds iff the default literal *DefLit* is unreliable w.r.t. *RS*.

```
literal_unreliable(DefLit,Anc) :- constraint(<-Body), select_lit(Body,Lit,Rest),
                                    depend(Lit,DefLit,[],Anc),possible(Rest,[],Anc).
```

% *depend*(*L*,*DefLit*,[],*Anc*) succeeds iff the literal *DefLit* is in the dependency set of the literal *L*
% w.r.t. *RS*.

```
depend(true,_,_):- !,fail.
depend((G1,G2),DefLit,LAnc,Anc) :- depend(G1,DefLit,LAnc,Anc).
depend((G1,G2),DefLit,LAnc,Anc) :- !, depend(G2,DefLit,LAnc,Anc).
depend(--(--Lit),DefLit,LAnc,Anc) :- !, depend(Lit,DefLit,LAnc,Anc).
depend(DefLit,DefLit,_,_) :- !.
depend(~Lit,DefLit,LAnc,Anc) :- !, depend(--Lit,DefLit,LAnc,Anc).
depend(Lit,DefLit,LAnc,_) :- member(d(Lit),LAnc), !, fail.
depend(Lit,DefLit,LAnc,Anc) :- rule(Rule,Lit<-Body,_),possible(Body,[d(Lit)|LAnc],Anc),
                                 depend(Body,DefLit,[d(Lit)|LAnc],Anc).
```

% *possible*(*L*,*LAnc*,*Anc*) succeeds iff the literal *L* is in the possible set w.r.t. *RS*.

```
possible(true,_,_):- !.
possible(~true,_,_):- !, fail.
possible(~(G1,G2),LAnc,Anc) :- possible(~G1,LAnc,Anc).
possible(~(G1,G2),LAnc,Anc) :- !, possible(~G2,LAnc,Anc).
possible((G1,G2),LAnc,Anc) :- !, possible(G1,LAnc,Anc), possible(G2,LAnc,Anc).
possible(~(~Lit),LAnc,Anc) :- !, possible(Lit,LAnc,Anc).
possible(--(--Lit),LAnc,Anc) :- !, possible(Lit,LAnc,Anc).
possible(~Lit,LAnc,Anc) :- possible(--Lit,[p(~Lit)|LAnc],Anc,), !.
possible(Lit,_,_):- succeeds(Lit), !.
possible(~Lit,LAnc,Anc) :- member(p(Lit),LAnc), !, fail.
possible(Lit,LAnc,Anc) :- member(p(~Lit),LAnc), !, fail.
possible(~Lit,_,_) :- succeeds(Lit), !, fail.
possible(Lit,_,_) :- succeeds(~Lit), !, fail.
possible(Lit,LAnc,Anc) :- distance_in_list(p(Lit),LAnc,_,List), !, ((List=zero, Lit=(~_))->true).
possible(~Lit,LAnc,Anc):- !, findall(Body, rule(_,Lit<-Body,_),List),
                              all_possibly_false(List,[p(~Lit)|LAnc],Anc).
possible(Lit,Anc) :- rule(Rule,Lit<-Body,_), not RS_derive(--Lit,Anc),
                       possible(Body,[p(Lit)|LAnc],Anc).
```

```
all_possibly_false([],_,_):- !.
all_possibly_false([Body|List],LAnc,Anc):- possible(~Body,LAnc,Anc),
                                           all_possibly_false(List,LAnc,Anc).
```

% *rule_unreliable*(*Rule*,*Anc*) succeeds iff the rule with label *Rule* is unreliable w.r.t. *RS*.

```
rule_unreliable(Rule,Anc) :- constraint(<-Body), select_lit(Body,Lit,Rest),
                             r_depend(Lit,Rule,[],Anc),r_possible(Rest,Rule,[],Anc).
```

% *r_depend*(*L*,*Rule*,*LocalAnc*,*Anc*) succeeds iff the head of the rule with label *Rule* is in
% dependency set of the literal *L* w.r.t. rule *Rule* and *RS*.

```
r_depend(true,_,_,_):- !,fail.
r_depend((G1,G2),Rule,LAnc,Anc) :- r_depend(G1,Rule,LAnc,Anc).
r_depend((G1,G2),Rule,LAnc,Anc) :- !, r_depend(G2,Rule,LAnc,Anc).
r_depend(--(--Lit),Rule,LAnc,Anc) :- !, r_depend(Lit,Rule,LAnc,Anc).
r_depend(Lit,Rule,_,_) :- !, rule(Rule,Lit<-Body,_,_).
r_depend(~Lit,Rule,LAnc,Anc) :- !, r_depend(--Lit,Rule,LAnc,Anc).
r_depend(Lit,Rule,LAnc,_) :- member(r_d(Lit,Rule),LAnc) !, fail.
r_depend(Lit,Rule,LAnc,Anc) :- rule(Rule1,Lit<-Body,_), not less_prior(Rule1,Rule),
                               r_possible(Body,Rule,[r_d(Lit,Rule)|LAnc],Anc),
                               r_depend(Body,Rule,[r_d(Lit,Rule)|LAnc],Anc).
```

% *r_possible*(*L*,*Rule*,*LocalAnc*,*Anc*) succeeds iff the literal *L* is in the possible set w.r.t. the rule
% with label *Rule* and *RS*.

```
r_possible(true,_,_):- !.
r_possible((G1,G2),Rule,LAnc,Anc) :- !, r_possible(G1,Rule,LAncAnc),
                                       r_possible(G2,Rule,LAncAnc).
r_possible(--(--Lit),Rule,LAncAnc) :- !, r_possible(Lit,Rule,LAncAnc).
r_possible(~Lit,Rule,LAncAnc) :- !, r_possible(--Lit,Rule,LAncAnc).
r_possible(Lit,Rule,LAncAnc) :- distance_in_list(r_p(Lit,Rule),LAnc,_,_), !, fail.
r_possible(Lit,Rule,LAncAnc) :- rule(Rule1,Lit<-Body,_), not less_prior(Rule1,Rule),
                                not RS_der(--Lit,Anc),
                                r_possible(Body,Rule,[r_p(Lit,Rule)|LAnc],Anc).
```

% *select_lit*(*List*, *Lit*, *Rest*) selects a literal *Lit* from the list of literals *List*. The rest of the
% literals are stored in *Rest*.

```
select_lit(Lit,Lit,()) :- Lit \=(G1,G2).
select_lit((Left,Right),Lit,Rest) :- select_lit(Left,Lit,RestLeft), Rest=(RestLeft,Right).
select_lit((Left,Right),Lit,Rest) :- select_lit(Right,Lit,RestRight), Rest=(Left,RestRight).
```

% *distance_in_list*(*L*, *Literal_list*, *Distance*) succeeds iff *L* is a member of the input *Literal_list*,
% i.e., *Literal_list* =L_1,..., L_i, *L*, L_{i+1},..., L_n. The output *Distance* is *zero* iff all literals
% *L*,L_{i+1},...,L_n are either (i) default literals or (ii) have the form *p*(*K*), or (iii) have the form
r_p(*K*).
% Otherwise, it is *non_zero*.

```
distance_in_list(Lit,[Lit|Anc],F,zero) :- var(F),!.
distance_in_list(Lit,[Lit|Anc],n_z,non_zero) :- !.
distance_in_list(Lit,[~_,~X|Anc],F,List):- !, distance_in_list(Lit,[~X|Anc],F,List).
distance_in_list(Lit,[~_,X|Anc],_,List):- !, distance_in_list(Lit,[X|Anc],n_z,List).
distance_in_list(Lit,[_,~X|Anc],_,List):- !, distance_in_list(Lit,[~X|Anc],n_z,List).
distance_in_list(Lit,[p(_),p(X)|Anc],F,List):-!, distance_in_list(Lit,[p(X)|Anc],F,List).
distance_in_list(Lit,[p(_),X|Anc],_,List):- !, distance_in_list(Lit,[X|Anc],n_z,List).
distance_in_list(Lit,[_,p(X)|Anc],_,List):- !, distance_in_list(Lit,[p(X)|Anc],n_z,List).
distance_in_list(Lit,[r_p(_),r_p(X)|Anc],F,List):-!, distance_in_list(Lit,[r_p(X)|Anc],F,List).
distance_in_list(Lit,[r_p(_),X|Anc],_,List):- !, distance_in_list(Lit,[X|Anc],n_z,List).
distance_in_list(Lit,[_,r_p(X)|Anc],_,List):- !, distance_in_list(Lit,[r_p(X)|Anc],n_z,List).
distance_in_list(Lit,[_,X|Anc],F,List):- !, distance_in_list(Lit,[X|Anc],F,List).

member(X,[X|_]) :- !.
member(X,[_|L]) :- member(X,L).
```

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] S. Abiteboul, G. Lausen, H. Uphoff, E. Waller, *Methods and Rules*, Proc. of the 1993 ACM SIGMOD Conference on the Management of Data, 1993, pp. 32-41.

[2] H. Ait-Kaci, R. Nasr, *Login: a Logic Programming Language with Built-In Inheritance*, Journal of Logic Programming, 3, 1986, pp. 185-215.

[3] A. Analyti, S. Pramanik, *Reliable Semantics for Extended Logic Programs with Rule Prioritization*, Proc. of the Workshop on Logic Programming with Incomplete Information, in conjunction with the Intern. Logic Programming Symposium (ILPS'93), 1993, pp. 142-160.

[4] A. Analyti, S. Pramanik, *Semantics for Reasoning with Contradictory Extended Logic Programs*, to appear in the Proc. of the GULP-PRODE' 94 Joint Conference on Declarative Programming, 1994.

[5] A. Analyti, S. Pramanik, *Declarative Semantics for Contradictory Modular Logic Programs*, to appear in the Proc. of the 8th International Symposium on Methodologies for Intelligent Systems (ISMIS'94), 1994.

[6] A. Analyti, S. Pramanik, *Reliable Semantics for Extended Logic Programs with Rule Prioritization*, accepted for publication in the Journal of Logic and Computation, Oxford University Press.

[7] K.R. Apt, H.A. Blair, A. Walker, *Towards a Theory of Declarative Knowledge*, J. Minker (ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 1988, pp. 89-148.

[8] P. Atzeni, L.Cabibbo, G.Mecca, *ISALOG: A declarative language for complex objects with hierarchies*, Proc. of the 1993 IEEE Conference on Data Engineering, 1993, pp. 219-228.

[9] M. Baldoni, L. Giordano, A. Martelli, *A Multimodal Logic to Define Modules in Logic Programming*, Proc. of the 1993 Intern. Logic Programming Symposium (ILPS'93), pp. 473-487.

[10] C. Baral, S. Kraus, J. Minker, *Combining multiple knowledge bases*, IEEE Transactions on Knowledge and Data Engineering, 3(2), June 1991, pp. 200-220.

[11] S. Brass, U.W. Lipeck, *Semantics of Inheritance in Logical Object Specifications*, C. Delobel, M.Kifer, Y. Masunaga (eds.), Proc. of the 2nd Intern. Conference on Deductive and Object-Oriented Databases, 1991, pp. 411-430.

[12] S. Brass, U.W. Lipeck, *Bottom-Up Query Evaluation with Partially Ordered Defaults*, Proc. of the 3rd Intern. Conference on Deductive and Object-Oriented Databases (DOOD'93), 1993, pp. 253-266.

[13] A. Brogi, E. Lamma, P. Mello, *Objects in a Logic Programming Framework*, Proc. of the 2nd Russian Conference on Logic Programming, 1991, pp. 102-113.

[14] A. Brogi, P. Mancarella, D. Pedreschi, F. Turini, *Meta for Modularising Logic Programming*, Proc. of the 3rd Intern. Workshop on Meta-Programming in Logic (META-92), 1992, pp. 105-119. Extended version in Technical Report, University of Pisa, 1992.

[15] B. Carre, J.M. Geib, *The Point of View notion for Multiple Inheritance*, Proc. of ECOOP/OOPSLA'90, 1990, pp. 312-321.

[16] K. L. Clark, *Negation as Failure*, Logic and Data Bases, Plenum Press, 1978, pp. 293-322.

[17] de Kleer, *An Assumption-Based Truth Maintenance System*, Artificial Intelligence, Elsevier Science Publishers, 28(2), 1986, pp. 127-162.

[18] J. Doyle, *A Truth Maintenance System*, Artificial Intelligence, Elsevier, 12(3), 1979, pp. 231-272.

[19] W. Dowling, J. Gallier, *Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulas*, Journal of Logic Programming, 3, (1984), pp. 267-284.

[20] P.M. Dung, P. Ruamviboonsuk, *Well-Founded Reasoning with Classical Negation*, A. Nerode, W. Marek, V.S. Subrahmanian (eds.), Proc. of the 1st Intern. Workshop on Logic Programming and Non-monotonic Reasoning, 1991, pp. 120-132.

[21] P.M. Dung, *An Argumentation Semantics for Logic Programs with Explicit Negation*, Proc. of the 10th Intern. Conference on Logic programming (ICLP'93), 1993, pp. 616-630.

[22] C. Elkan, *A Rational Reconstruction on Nonmonotonic Truth Maintenance System*, Artificial Intelligence, Elsevier Science Publishers, 43(2), 1990, pp. 219-234.

[23] M. Fitting, *A Kripke-Kleene Semantics for Logic Programs*, Journal of Logic Programming, 2(4), 1985, pp. 295-312.

[24] D. Gabbay, E. Laenens, D. Vermeir, *Credulous vs. Sceptical Semantics for Ordered Logic Programs*, J. Allen, R. Fikes, E. Sandewall (eds.), Proc. of the 2nd Intern. Conference on Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann, 1991, pp. 208-217.

[25] P. Geerts, D. Vermeir, *Credulous and autoepistemic reasoning using ordered logic*, A. Nerode, W. Marek, V.S. Subrahmanian (eds.), Proc. of the 1st Intern. Workshop on Logic Programming and Non-monotonic Reasoning, 1991, pp.21-36.

[26] M. Gelfond, V. Lifschitz, *The Stable Model Semantics for Logic Programming*, R. Kowalski, K. Bowen (eds.), Proc. of the 5th Symposium on Logic Programming, MIT Press, 1988, pp. 1070-1080.

[27] M. Gelfond, V. Lifschitz, *Classical Negation in Logic programs and Disjunctive Databases*, New Generation Computing, 9, OHMSHA, 1991, pp. 365-385.

[28] L. Giordano, A. Martelli, *General Stable Models, Truth Maintenance and Conflict Resolution*, Proc. of the 7th Intern. Conference on Logic Programming, 1990, pp. 427-441.

[29] L. Giordano, A. Martelli, *Truth Maintenance Systems and Belief Revision*, Proc. of the ECAI-90 Workshop on Truth Maintenance Systems, 1990, pp. 71-86.

[30] L. Giordano, A. Martelli, *A Modal Reconstruction of Blocks and Modules in Logic Programming*, Proc. of the 1991 Intern. Logic Programming Symposium (ILPS'91), pp. 239-253.

[31] M. Goldszmidt, J. Pearl, *Rank-based systems: A simple approach to belief revision, belief update and reasoning about evidence and actions*, Proc. of the 3rd Intern. Conference on Knowledge Representation and Reasoning (KR'92), 1992, pp. 661-672.

[32] A. Heuer, P. Sander, *Preserving and Generating Objects in the LIVING IN A LATTICE Rule Language*, IEEE Intern. Conference on Data Engineering, 1991, pp. 562-569.

[33] R. Hill, *Lush resolution and its completeness*, Technical Report DCL Memo 78, Department of Artificial Intelligence, University of Edinburgh, August 1974.

[34] J. F. Horty, R. H. Thomason, D. S. Touretzky, *The Skeptical Theory of Inheritance in Nonmonotonic Semantics Networks*, Journal of Artificial Intelligence, 42, 1990, pp. 311-348.

[35] A. Hunter, *A Conceptualization of Preferences in Non-monotonic Proof Theory*, D. Pearce and G. Wagner (eds). Logics in AI, LNCS 633, 1992, pp. 174-188.

[36] M.H. Jamil, L.V.S. Lakshmanan, *Realizing Orlog in LDL*, I.S. Mumick (ed.), Proc. of the Workshop on Combining Declarative and Object-Oriented Databases (in cooperation with ACM SIGMOD), May 1993, pp.45-59.

[37] M. Kifer, G. Lausen, *F-logic: A higher-order language for reasoning about objects, inheritance and scheme*, J. Clifford, B. Lindsay, D. Maier (eds.), Proc. of the 1989 ACM SIGMOD Intern. Conference on the Management of Data, 1989, pp. 134-146.

[38] M. Kifer, G. Lausen, J. Wu, *Logical Foundations of Object-Oriented and Frame-Based Languages*, Technical Report 90/14 (2-nd revision), Dept. of Computer Science, SUNY at Stony Brook, August 1990. (To appear in the Journal of the ACM).

[39] R. Reiter, *On Closed World Data Bases*, Logic and Data Bases, Plenum Press, New York, 1978, pp. 55-76.

[40] R.A. Kowalski, D. Kuehner, *Linear Resolution with Selection Function*, Artificial Intelligence, 2, 1971, pp. 227-260.

[41] R.A. Kowalski, *Predicate Logic as a Programming Language*, Information Processing, North Holland, 1974, pp. 569-574.

[42] E. Laenens, D. Vermeir, *A Fixpoint Semantics for Ordered Logic*, Journal of Logic and Computation, 1(2), Oxford University Press, 1990, pp. 159-185.

[43] E. Laenens, D. Vermeir, *Assumption-free Semantics for Ordered Logic Programs: On the Relationship Between Well-founded and Stable Partial Models*, Journal of Logic and Computation, 2(2), Oxford University Press, 1992, pp. 133-172.

[44] E. Lamma, P. Mello, G. Rossi, *Parametric Composable Modules in a Logic Programming Language*, Computer Languages, 18(2), Pergamon Press, 1993, pp. 105-123.

[45] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, second edition, 1987.

[46] C.L. Mason, R.R. Johnson, *DATMS: A Framework for Distributed Assumption Based Reasoning*, Distributed Artificial Intelligence, 2, 1989, pp. 293-317.

[47] F.G. McCabe, *Logic and Objects*, Prentice Hall, 1992.

[48] D. Miller, *A Logical Analysis of Modules in Logic Programming*, Journal of Logic programming, 6, 1989, pp. 79-108.

[49] L. Monteiro, A. Porto, *Contextual Logic Programming*, Proc. of the 6th Intern. Conference on Logic Programming (ICLP' 89), Lisbon, 1989, pp. 284-299.

[50] L. Monteiro, A. Porto, *A Language for Contextual Logic Programming*, Logic Programming Languages: Constraints, Functions and Objects, MIT Press, 1993, pp. 115-147.

[51] A.H.H. Ngu, L. Wong, S. Widjojo, *On Canonical and Non-canonical Classifications*, C. Delobel, M.Kifer, Y. Masunaga (eds.), Proc. of the 2nd Intern. Conference on Deductive and Object-Oriented Databases (DOOD'91), 1991, pp. 371-390.

[52] L.M. Pereira, J.J. Alferes, J.N. Aparicio, *Contradiction Removal within Well Founded Semantics*, A. Nerode, W. Marek, V.S. Subrahmanian (eds.), Proc. of the 1st Intern. Workshop on Logic programming and Non-monotonic Reasoning, 1991, pp. 106-119.

[53] L.M. Pereira, J.N. Aparicio, J.J. Alferes, *Nonmonotonic Reasoning with Well Founded Semantics*, Proc. of the Intern. Conference on Logic Programming (ICLP',91), 1991, pp. 475-504.

[54] L.M. Pereira, J.J. Alferes, *Well Founded Semantics for Logic Programs with Explicit Negation*, B.Neumann (ed.), Proc. of the 10th European Conference on Artificial Intelligence, John Wiley &Sons, 1992, pp. 102-106. Also, Technical Report, AI Centre, Uninova, March 1992.

[55] L.M. Pereira, J.J. Alferes, J.N. Aparicio, *Contradiction Removal with Explicit Negation*, Proc. of the Applied Logic Conference, ILLC, 1992.

[56] L.M. Pereira, J.N. Aparicio, J.J. Alferes, *Nonmonotonic Reasoning with Logic Programming*, Journal of Logic Programming, Elsevier Science Publishing, 17, 1993, pp. 227-263.

[57] T.C. Przymusinski, *Perfect Model Semantics*, Proceedings of the 5th International Conference and Symposium on Logic Programming, R.A. Kowalski, K.A. Bowen (eds.), 1988, pp. 1081-1096.

[58] T. Przymusinski, *Every Logic program has a Natural Stratification and an Iterated Fixed Point Model*, Proc. of the 8th Symposium on Principles of Database Systems, ACM SIGACT-SIGMOD, 1989, pp. 11-21.

[59] W. Drabent, *SLS-resolution without floundering*, 2nd Intern. Workshop on Logic programming and Non-monotonic Reasoning, 1993, pp.82-98. Extended version *What is failure? An approach to constructive negation* submitted to *Acta Informatica*.

[60] T. Przymusinski, *Well-Founded Semantics Coincides with the Three-Valued Stable Semantics*, Fundamenta Informaticae XIII, IOS Press, 1990, pp. 445-463.

[61] H. Przymusinska, T. Przymusinski, *Semantics Issues in Deductive Databases and Logic Programs*, R.B. Banerji (ed.), Formal Techniques in Artificial Intelligence, A Sourcebook, Elsevier, Amsterdam, 1990, pp. 321-367.

[62] T.C. Przymusinski, *Semantics of Disjunctive Logic Programs and Deductive Databases*, C. Delobel, M.Kifer, Y. Masunaga (eds.), Proc. of the 2nd International Conference on Deductive and Object-Oriented Databases, Morgan Kaufmann, 1991, pp. 85-107.

[63] R. Reiter, *A Logic for Default Reasoning*, Artificial Intelligence, Elsevier Science Publishers, 13, 1980, pp. 81-132.

[64] N. Roos, *A Logic for Reasoning with Inconsistent Knowledge*, Artificial Intelligence, Elsevier Science Publishers, 57, 1992, pp. 69-103.

[65] M. Ryan, *Defaults and Revision in Structured Theories*, Proc. of the 6th IEEE Symposium on Logic in Computer Science (LICS), 1991, pp. 362-373.

[66] M. Ryan, *Representing Defaults as Sentences with Reduced Priority*, Proc. of the 3rd International Conference on Knowledge Representation and Reasoning (KR'92), 1992, pp. 649-660.

[67] C. Sakama, *Extended Well-Founded Semantics for Paraconsistent Logic Programs*, Proc. of the Intern. Conference on Fifth Generation Computer Systems (FGCS'94), ICOT, 1992, pp. 592-599.

[68] D.T. Sannella, L.A. Wallen, *A calculus for the construction of modular Prolog programs* Journal of Logic Programming, 12, 1992, pp. 147-177.

[69] J. S. Schlipf, *A Survey of Complexity and Undecidability Results in Logic Programming*, H. Blair, W. Marek, A. Nerode, J. Remmel (eds.), Proc. of the Workshop on Structural Complexity and Recursion-theoretic Methods in Logic Programming, 1992, pp. 143-165.

[70] L. A. Stein, *Skeptical Inheritance: Computing the Intersection of Credulous Extensions*, Proc. of the 11th Intern. Joint Conference on Artificial Intelligence, 1989, pp. 1153-1158.

[71] V.S. Subrahmanian, *Amalgamating Knowledge Bases*, to be published in ACM Transactions on Database Systems (TODS), 1994.

[72] F. Teusink, *A Proof Procedure for Extended Logic Programs*, Proc. of the 1993 Intern. Logic Programming Symposium (ILPS'93), pp. 235-249.

[73] D. S. Touretzky, *Implicit Ordering of Defaults in Inheritance Systems*, International Joint Conference on Artificial Intelligence (IJCAI-84), 1984, pp. 322-325.

[74] D. S. Touretzky, J. F. Horty, R. H. Thomason, *A Clash of Intuitions: The Current State of Nonmonotonic Multiple Inheritance Systems*, International Joint Conference on Artificial Intelligence (IJCAI-87), 1987, pp. 476-482.

[75] M. van Emden, R. Kowalski, *The Semantics of Predicate Logic as a Programming Language*, Journal of the ACM, 23(4), 1976, pp. 733-742.

[76] A. van Gelder, K.A. Ross, J.S. Schlipf, *The Well-Founded Semantics for General Logic Programs*, Journal of the Association for Computing Machinery, 38(3), July 1991, pp. 620-650.

[77] G. Wagner, *Reasoning with Inconsistency in Extended Deductive Databases*, L.M. Pereira and A. Nerode (eds.), Proc. of the 2nd Intern. Workshop on Logic Programming and Non-monotonic Reasoning, 1993, pp. 300-315.

[78] C. Witteveen, *Skeptical Reason Maintenance is Tractable*, Proc. of the 2nd Intern. Conference on Knowledge Representation and Reasoning (KR91), 1991, pp. 570-581.

[79] C. Witteveen, *Expanding Logic Programs*, D. Pearce and G. Wagner (eds). Logics in AI, LNCS 633, 1992, pp. 372-390.

[80] K. Yoshida, C. Smith, T. Kazic, G. Michaels, R. Taylor, D. Zawada, R. Hagstrom, R. Overbeek, *Toward a Human Genome Encyclopedia*, Proc. of the Intern. Conference on Fifth Generation Computer Systems, 1992, pp. 307-317.

[81] C. Zaniolo, *Object Identity and Inheritance in Deductive Databases - an Evolutionary Approach*, W. Kim (ed.), Proc. of the 1st Intern. Conference on Deductive and Object-Oriented Databases, 1989, pp.106-114.