Jürgen Dix

Semantics of Logic Programs:

Their Intuitions and Formal Properties.

An Overview

1. Introduction

Historically, semantics for logic programs have been considered in the logic programming community for about 20 years. It began with Colmerauer et al. (1973), Kowalski (1974), van Emden and Kowalski (1976) and led to the definition and implementation of *PROLOG*, a by now theoretically well-understood programming language (at least the declarative part consisting of Horn-clauses: *pure PROLOG*). Extensions of PROLOG allowing negative literals have been also considered in this area: they rely on the idea of *negation-as-finite-failure*, we call them *Logic-Programming-semantics* (or shortly *LP-semantics*).

In parallel, starting at about 1980, *Non-monotonic Reasoning* entered into computer science and began to constitute a new field of active research. In recent years, independently of the research in logic programming, people interested in non-monotonic reasoning also tried to define declarative semantics for programs containing negative literals in their bodies (or even *disjunctions* in their heads). They defined various semantics by appealing to (different) intuitions they have about programs.

This second line of research started in 1986 with the *Workshop on the Foundations of Deductive Databases and logic programming* organized by Jack Minker: the revised papers of the proceedings were published in Minker (1988a). The *stratified* (or the similar *perfect*) semantics presented there can be seen as a splitting-point: it is still of interest for the logic programming community (see Cavedon and Lloyd (1989)) but its underlying intuitions were inspired by non-monotonic reasoning. Semantics of this kind leave the philosophy underlying classical logic programming in that their primary aim is not to model negation-as-finite-failure, Clark's completion or SLDNF-resolution, but to construct new, more powerful semantics

Logic, Action and Information,
André Fuhrmann and Hans Rott, eds.,
de Gruyter, Berlin–New York, 1995.

suitable for applications of some forms of non-monotonic reasoning. Let us call such semantics *NMR-semantics*.

The situation in deductive databases up to 1988 is summarized in the two complementary papers Minker (1988b) and Shepherdson (1988). While Minker (1988b) gives a historical, informal description of the developments in the last thirty years, sketching the main lines of developments, Shepherdson (1988) treats the problem of negation in logic programming in a very comprehensive and detailed way. All necessary notions are introduced in a formal and precise way, many results are discussed (both in depth and in breadth) and even proofs are given: the whole subject is presented fairly completely.

Nowadays, due to the work of Apt, Blair and Walker, Fitting, Lifschitz, Przymusinski and others, very close relationships between these two independent research lines became evident. Methods from logic programming, e.g. least fixpoints of certain operators, can be used successfully to define NMR-semantics. The NMR-semantics also shed new light on the understanding of the classical nonmonotonic logics such as *Default Logic* (DL), *Autoepistemic Logic* (AEL) and the various versions of *Circumscription* (CIRC). In addition, the investigation of possible semantics for logic programs seems to be useful because

1. parts of NMR's (which are usually defined for full predicate logic, or even contain additional (modal)-operators) may be "implemented" with the help of such programs,

2. non-monotonicity in these logics may be described with an appropriate treatment of negation in logic programs.

Recently two other interesting overviews about negation in logic programming have appeared: Minker's article Minker (1993) in the *Special Issue of the Journal of Logic Programming on Non-Monotonic Reasoning and Logic Programming* (a continuation and extended version of Minker (1988b, 1991)), and Apt and Bol's article Apt and Bol (1994) in the jubileum (10th anniversary) issue of the *Journal of Logic Programming*. While Minker's article gives an almost complete description of all the activities and different approaches in the field, Bol/Apt and we (in this article) also try to present recent research results in a comprehensive and detailed manner. Although no proofs are given, all important definitions and notions are formally introduced to illustrate the underlying ideas in a precise and strict fashion. Apt/Bol's paper and the present overview can be seen as complementary in a sense: while Apt and Bol concentrate more on *LP-semantics*, we are more concerned with *NMR-semantics*. This paper is somewhere in between Minker (1993) and a full-detailed paper in the style of Shepherdson (1988) that still has to be written.

The paper is organized as follows. In Section 2 we present and discuss LP-semantics, semantics for programs from the viewpoint of logic programming. The main intuition here is the idea of *negation-as-finite-failure*. We consider procedural mechanisms such as SLD- and SLDNF-resolution and suitable declarative semantics that were inspired by those: Clark's completion $comp(P)$ and its 3-valued variant $comp_3(P)$ due to Fitting. For both semantics, we discuss correctness and completeness results with respect to SLD and SLDNF. This section summarizes the main part of Shepherdson (1988).

In Section 3 we consider NMR-semantics, semantics for programs from the viewpoint of non-monotonic reasoning. After comparing the different intuitions underlying LP and NMR, we present the *supported* semantics M_P^{supp} due to Apt, Blair and Walker. This semantics is defined only for a subclass of the class of all programs with negation: the class of *stratified* programs. All semantics proposed so far in the literature coincide on this class with M_P^{supp}. We then consider the most important extensions of M_P^{supp}: the stable semantics STABLE and the wellfounded semantics WFS. After a brief review of some extensions of WFS recently proposed, we give some remarks concerning semantics of *disjunctive* programs. This area is still in its infancy: only recently some interesting proposals of semantics defined for all disjunctive programs came up. We end this section with some notes on the complexity of the semantics introduced so far.

Section 4 considers translations from logic programs into the by now well-known nonmonotonic logics such that semantics of a program correspond to the meaning of the translated program in the respective logic. In particular, we compare CWA with negation-as-finite-failure, CIRC with M_P^{supp}, DL with STABLE and CIRC, AEL_{cl} with WFS.

Finally, Section 5 presents our own approach of how all these different semantics presented in the former sections may be compared and classified. We show that any semantics induces a natural consequence relation. Abstract properties of this relation turn out to be very relevant here. In particular, (adapted versions of) the notions of *Cumulativity* and *Rationality*, known in the context of non-monotonic reasoning since a couple of years, play an important role. We refer to them as *strong* principles. In addition we introduce *weak* principles and argue that they should be true for any well-behaved semantics. Using these weak principles, we can show some interesting theorems: Theorems 5.26 and 5.27. We claim that by taking both sorts of principles together, *weak* and *strong* properties can be used to uniquely characterize certain semantics. Important representation theorems (Conjectures 5.29, 5.30 and 5.31) would follow. We also show how our approach can be applied to disjunctive programs by using a general *Partial Evaluation Property*. As a result, we get an elegant definition of a semantics D-WFS that is strongly related to the recent static semantics

proposed by Przymusinski.

1.1 Notation and Terminology.
A *general disjunctive* logic program consists of a finite number of *rules* that allow arbitrary *positive* clauses to appear in their heads:

$$A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_l \quad \text{where } n \geq 1 \ .$$

If $l = 0$, the program is *positive disjunctive*; if $n = 1$ the program is *normal*; if $n = 1$ and $l = 0$ the program is *positive* or *definite* (see Figure 2).

All the literals may contain free variables: the rules are viewed as shorthand notations for all possible instantiations. P_{inst} stands for the (infinite) sets of fully instantiated rules and facts (*ground* clauses). A program is, in addition, called *propositional* or *Datalog*, if it does not contain any function symbols, but only propositional variables or their negations.

When we use the notation $P \cup U$, U stands for an (infinite) set of atoms, *positive facts*, that we want to distinguish from the facts in P. That is, P contains rules and positive facts, but U only contains atoms.

A query Q is a conjunction of literals. A substitution Θ in a query Q, is a replacement of the variables in Q by certain terms: $Q\Theta$ stands for the resulting formula, $\forall\, Q\Theta$ for its *universal* closure and $\exists\, Q\Theta$ for its *existential* closure.

We denote the Herbrand base with respect to a program P by $B_{\mathcal{L}_P}$ or simply by B_P: the underlying language $\mathcal{L}_P$ is given by the symbols in P. Th(Φ) denotes the classical closure of the set of formulae Φ and *Fml* denotes the set of all formulae.

Finally, let MIN-MOD(T) denote the class of all *two-valued* minimal Herbrand-models of the theory T. A Herbrand model $\mathcal{A}$ of T is called *minimal*, if there is no other model $\mathcal{A}'$ of T such that for all atoms a of the Herbrand base B_T: $\mathcal{A}' \models a$ implies $\mathcal{A} \models a$.

We also need some notions from 3-valued logic. We use truth values **t** "true", **f** "false", **u** "undefined" and the Kleene connectives $\vee, \wedge, \neg$ and $\leftarrow$. $\leftarrow$ is the *weak* implication, where "$\mathbf{u} \leftarrow \mathbf{u}$" is considered to be *true*. Additionally, we can use two different orderings of the truth values: the lattice $\mathbf{3}_t$ defined by $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$ (*truth-ordering*) and the semi-lattice $\mathbf{3}_k$ defined by $\mathbf{u} \leq_k \mathbf{t}$, $\mathbf{u} \leq_k \mathbf{f}$ (*knowledge-ordering*): see Figure 1. We regard a three-valued interpretation $\mathcal{I}$ as a pair $\langle T; F \rangle$, consisting of the sets of *atoms* T (the *true* atoms) and F (the *false* atoms). We are using True($\mathcal{I}$) (resp. False($\mathcal{I}$)) to denote the atoms that are true (resp. false) in $\mathcal{I}$ and we will also represent $\mathcal{I}$ by just enumerating its ground literals: $\mathcal{I} = \mathit{True}(\mathcal{I}) \cup \{\neg x : \ x \in \mathit{False}(\mathcal{I})\}$. It is clear that an interpretation $\mathcal{I}$ on atoms can be uniquely extended to an interpretation $\hat{\mathcal{I}}$ of all sentences. The relations $\leq_t$ and $\leq_k$ also naturally extend. Thus, to define a mapping from the set of three-valued interpretations into itself, it suffices to consider

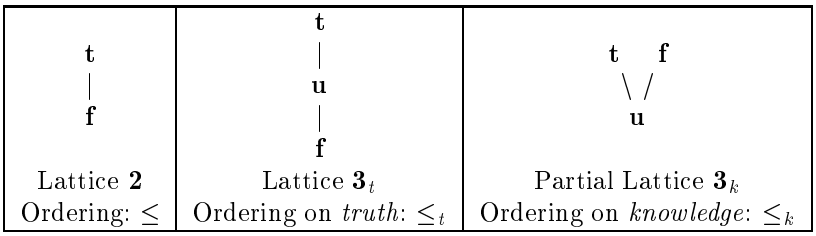

Figure 1: Important (Partial) Lattices of Truth-Values

mappings from $\mathbf{3}^{B_P}$ into itself. In the sequel, we write "$\mathbf{3}_k^{B_P} \rightarrow \mathbf{3}_k^{B_P}$" to indicate that we are interested in the $\leq_k$-ordering.

Given an operator Φ on a (semi-) lattice L, we denote by $\Phi\uparrow^\lambda$ its λ'th iteration from below (starting with the empty set $\emptyset$) and by $\Phi\downarrow^\lambda$ its λ'th iteration from above (starting with the whole set L). Its least (resp. greatest) fixpoint is $lfp(\Phi)$ (resp. $gfp(\Phi)$).

Proof-theoretically, semantics SEM_P of a program P can be defined as a set of literals that are derivable from P for a particular derivation mechanism (such as SLDNF-resolution).

Model-theoretically, all semantics we will be presenting are defined as subsets of MOD(P) (the set of all models of P). More precisely, they are subsets of the (three-valued) Herbrand-models of the underlying language $\mathcal{L}_P$

- $\text{SEM}_P \subseteq \text{MOD}_{3-val}^{Herb_{\mathcal{L}}P}(P)$.

While SEM_P stands for a certain set of models, we will also use SEM_P^{scept} (or simply SEM_P) to denote the three-valued model of P formed by all ground literals true in all elements of SEM_P (the precise definitions will be given in Section 5). With this notion, we are able to introduce two sorts of *extensions* of a semantics. When we say that a semantics SEM′ *extends* another semantics SEM, we have to distinguish between two different notions:

- SEM $\leq_k$ SEM′: this means that SEM′ classifies more atoms as true or false than SEM, or
- SEM′ is defined for a class of programs that strictly includes the class of programs for which SEM is defined and for all programs of this smaller class, the two semantics coincide.

The first notion also makes perfect sense for semantics defined for the same class of programs (see Figure 2).

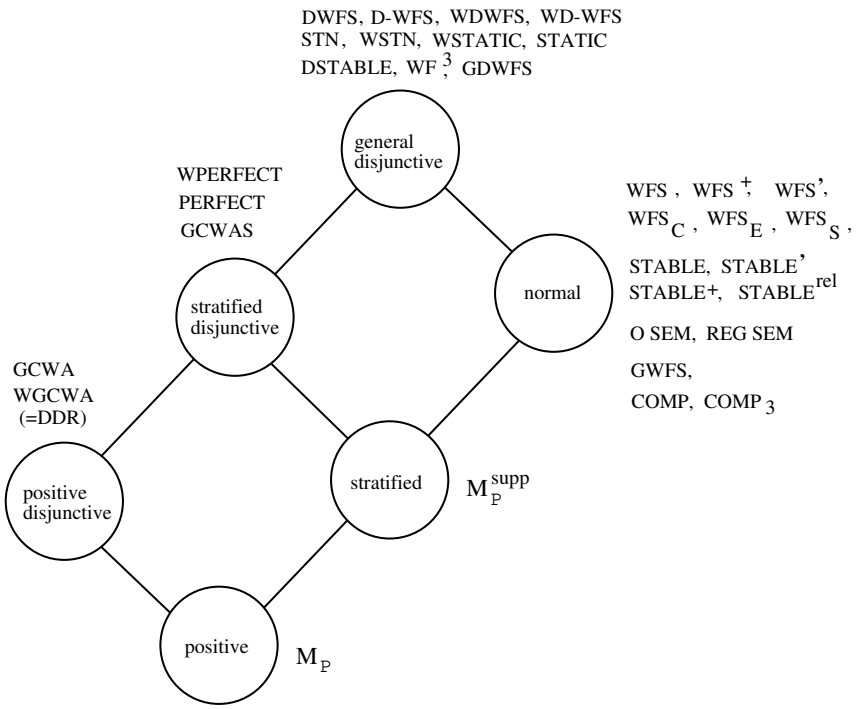

Figure 2: Classes of Programs

2. Logic Programming Semantics

Quite independently of the work in non-monotonic reasoning, there evolved interesting work in the logic programming community: proof and model theory for *normal* logic programs (programs, where negative literals are allowed in the body) were defined. The proof theory originated from the well-known SLD-resolution of definite programs; it is based on the notion of *negation-as-finite-failure*: SLDNF-resolution. The corresponding semantics COMP is defined by Clark's completion $comp(P)$ (see Clark (1978)). In logic programming, much work aimed at finding (syntactically defined) classes of programs, for which *soundness* and, more importantly, *completeness* results hold (see Decker and Cavedon (1990), Cavedon and Lloyd (1989)).

It turned out that both the SLDNF-resolution and the completion semantics have various drawbacks: inconsistency of $comp(P)$ and problems related to *floundering*. This section addresses these problems. For a more comprehensive presentation, the reader is referred to the textbooks Lloyd

(1987), Apt (1990) or to the recent Apt and Bol (1994).

The SLD-resolution in logic programming represents the top-down method of answering queries. In the field of deductive databases, there is also the bottom-up approach. This is an interesting alternative that led in recent years to the definition of *magic sets* and various proposals and technologies for answering queries relative to a given database. This approach is not dealt with here, rather we refer the reader to Ramakrishnan (1991), Ramakrishnan and Sudarshan (1991), Ullman (1989), Sagiv (1990), Brass and Lipeck (1992), Brass and Dix (1994c, 1994b), Brass (1994), Degerstedt and Nilsson (1995).

2.1 Definite Programs.

A *query* Q is a conjunction of atoms, possibly containing free variables. Asking a query relative to a program means finding instantiations for the variables in Q. The famous SLD-resolution constitutes the procedural semantics for definite programs. We give here only an example (see Figure 3) for illustrating the SLD-inference procedure.

Example 2.1 (SLD-Resolution).
Let the program P_{SLD} consist of the following three clauses

$$\begin{array}{lll} (1) & p(x,z) & \leftarrow \quad q(x,y), p(y,z) \\ (2) & p(x,x) & \\ (3) & q(a,b) & \end{array}$$

Figure 3 illustrates the behaviour of SLD-resolution: the selected atom is underlined, the number 1, 2 or 3 indicates the number of the clause which the selected atom is resolved against. Obviously, there are three different sorts of branches, namely

1. *infinite* branches,
2. branches that *end up with the empty clause*, and
3. branches that *end in a deadlock ("Failure")*: no rule can be applied.

In this example we always resolve with the *last* atom in the goal under consideration. If we choose always the *first* atom in the goal, we will obtain, at least in this example, a *finite* tree.

Apt and Pedreschi consider in Apt and Pedreschi (1993) the problem of *termination* of pure PROLOG programs. In Vieille (1989), Vieille describes an improvement of SLD-resolution where the occurrence of *similar* goals in a branch is tested. For Datalog-programs, his method always generates a *finite* tree and is strongly related to *tabulation-methods* (see Tamaki and Sato (1986), Chen and Warren (1992), Bol and Degerstedt (1993), Degerstedt and Nilsson (1995)).

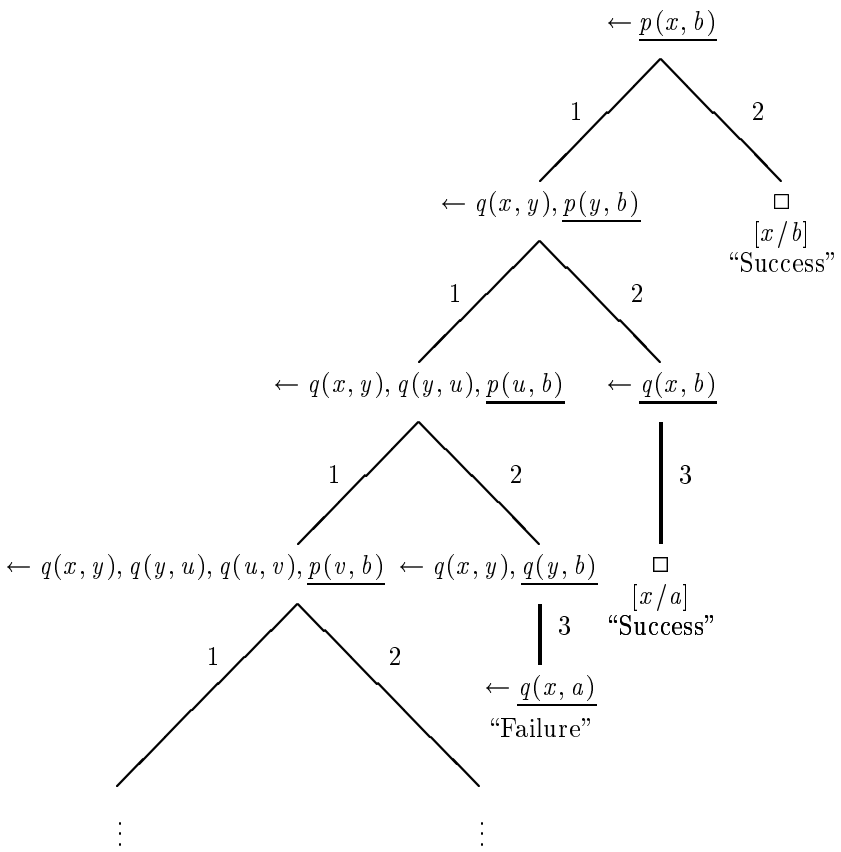

Figure 3: An Infinite SLD-Tree

It is well-known that definite programs always have a *least* Herbrand-model M_P. This model is related to SLD in the following way:

Theorem 2.2 (Soundness and Completeness of SLD).
$M_P \models \forall\ Q\Theta$ **iff** *SLD computes an answer* τ *that subsumes*[1] Θ *wrt* Q.

There is, however, a problem with this theorem. Consider the program $P := p(a)$, the query $Q := p(x)$ and the empty substitution $\Theta := \epsilon$. We have $M_P \models \forall x p(x)$ but SLD only computes the answer x/a. Przymusinski called this *the universal query problem*. There are essentially two solutions to avoid this behaviour: to use a language which is *rich enough* (i.e. contains sufficiently many terms, see the definition of $\mathcal{L}_{KU}$ in Sec-

[1] I.e. $\exists \sigma :\ Q\tau\sigma = Q\Theta$.

tion 2.4) or to consider arbitrary models, not only Herbrand models (again, see Section 2.4).

There is also a definition of M_P using the immediate consequence operator $T_P: \ 2^{B_P} \longmapsto 2^{B_P}; \ \mathcal{I} \longmapsto T_P(\mathcal{I})$ that turns out to be a useful tool in logic programming. For a definite program P, we define the operator $T_P: \ 2^{B_P} \longmapsto 2^{B_P}; \ \mathcal{I} \longmapsto T_P(\mathcal{I})$ as follows: (A is a ground atom)

$$T_P(\mathcal{I})(A) = \begin{cases} \mathbf{t}, \text{ if there is a clause } A \leftarrow L_1, \ldots, L_n \in P_{inst} \text{ with:} \\ \qquad - \ \forall i \leq n \text{ we have: } \mathcal{I}(L_i) = \mathbf{t}, \\ \mathbf{f}, \text{ otherwise.} \end{cases}$$

It turns out that T_P is monotone, continuous and $M_P = T_P{\uparrow}^{\omega} = \mathit{lfp}(T_P)$.

2.2 How to Obtain Negative Information.

Can we extend our framework and allow negative literals in a query? Up to now, we have no method to derive *negative* literals. We could, of course declare the least Herbrand-model M_P as the intended model. This gives us the well-known rule CWA for inferring negative information:

- CWA: Infer $\neg A$ $\underline{\text{iff}}$ $M_P \models \neg A$.

But M_P is only recursive enumerable and not recursive: note that the general halting problem for Turing machines can already be formalized within definite programs (see Tärnlund (1977)). Let us use from now on the following more general definition of CWA:

$$\text{CWA}(T) = \mathit{Th}(\ T \cup \{A : A \text{ is a ground atom with } T \not\models A\}).$$

The complexity of CWA has been proved in Apt and Blair (1990) to be *hyperarithmetical* even for *stratified* programs: see Section 3.6 for more complexity results.

To get a recursive enumerable subset of $B_P \setminus M_P$, we can try to derive $\neg A$ by *failing* the tree for A, i.e. by building, using SLD-resolution, a tree for A that is finite and has only failing branches. The axiomatic counterpart of this idea is due to Clark: Clark's completion $\mathit{comp}(P)$ defined in Clark (1978).

The idea of Clark was that a program P consists not only of the implications, but also of the information that *these are the only ones.* Roughly speaking, he argues that one should interpret the "$\leftarrow$"-arrows in rules as "$\leftrightarrow$"-arrows. We do not give the exact definitions here, as they are very complex; in the non-propositional case, a symbol for equality,

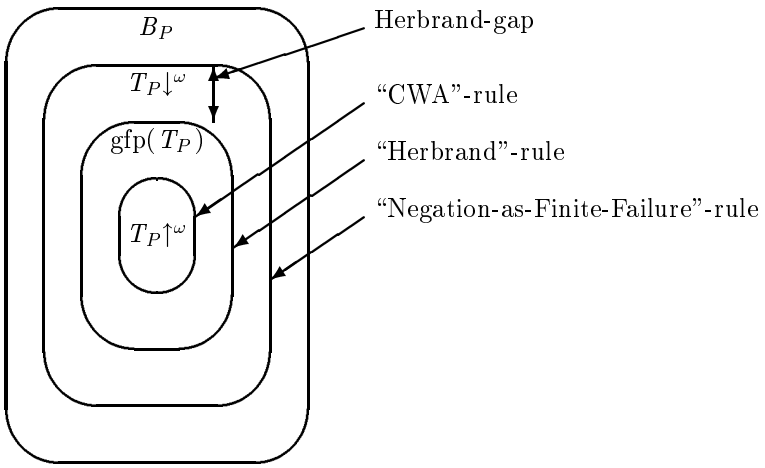

Figure 4: Rules to Infer Negative Information

together with axioms describing it[2], has to be introduced. However, for the propositional case, $comp(P)$ is obtained from P by just

1. collecting all given clauses with the same head into one new "clause" with this respective head and a disjunctive body (containing all bodies of the old clauses), and
2. replacing the implication-symbols "$\leftarrow$" by "$\leftrightarrow$".

The model-theoretic notion corresponding to $comp(P)$ is the notion of a *supported* model:

Definition 2.3 (Supported Model).
A Herbrand-model $M \subseteq B_P$ is supported*, if for every $A \in M$ there exists a clause $A \leftarrow L_1, L_2, \ldots, L_n \in P_{inst}$ such that $M \models L_1 \wedge L_2 \wedge \ldots \wedge L_n$.*

It is easy to see that

$$M \text{ is a supported model of } P \quad \underline{\text{iff}} \quad M \models comp(P).$$

Clark's completion gives us two new rules for inferring negative information, each of which defines a semantics SEM_P:

[2]CET: Clark's Equational Theory. CET($\mathcal{L}_P$) axiomatizes exactly the equality theory of all Herbrand($\mathcal{L}_P$)-models and is, under an additional assumption, complete (see Maher (1988)). See also Mancarella et al. (1988), Shepherdson (1988) for the problem of equality and the underlying language.

- *Herbrand*: Infer $\neg A$ iff $Herbrand(\mathcal{L}_P) - \mathrm{MOD}(comp(P)) \models \neg A$.
- *Neg.-as-F.-F.*: Infer $\neg A$ iff $\mathrm{MOD}(comp(P)) \models \neg A$.

Both rules, together with the before-mentioned CWA-rule are illustrated in Figure 4. Blair considered in Blair (1982) the complexity of *truth in all Herbrand-models of* $comp(P)$: this is shown to be Π^1_1-complete. While the Herbrand-rule is not recursive enumerable, the negation-as-finite-failure-rule is. For more on negation-as-failure, we refer the reader to Shepherdson (1984), Shepherdson (1985), Bruffaerts and Henin (1989).

To give a program where the behaviour of $comp(P)$ is not so obvious, we consider

Example 2.4 (COMP not cautious).

$$P_{comp}: \quad \begin{array}{lcl} a & \leftarrow & \neg p \\ a & \leftarrow & b \\ p & \leftarrow & a \\ p & \leftarrow & b \\ b & \leftarrow & b \end{array} \qquad comp(P_{comp}): \quad \begin{array}{lcl} a & \leftrightarrow & \neg p \vee b \\ p & \leftrightarrow & a \vee b \\ b & \leftrightarrow & b \end{array}$$

$comp(P_{comp}) = Th(\{\ a \leftrightarrow (\neg p \vee b),\ p \leftrightarrow (a \vee b),\ b \leftrightarrow b\ \}) = Th(\{a, b, p\})$. *But* $comp(P_{comp} \cup \{p\})$ *is the conjunction of* $a \leftrightarrow (\neg p \vee b)$ *and* $p \leftrightarrow (a \vee b \vee true)$, *so it is exactly* $Th(\{p,\ a \leftrightarrow b\})$. *In Section 5.2 we will see that this example shows the failure of Cautious Monotony for COMP.*

We have the following soundness and completeness result for definite Programs P and definite queries Q consisting of *positive* atoms: $Q = \bigwedge_i A_i$.

Theorem 2.5 (COMP and Fair FF-Trees).
$comp(P) \models \forall \neg Q$ **iff** *every fair SLD-tree for* P *wrt* Q *is finitely failed.*

We need the fairness assumption to ensure that the selection of atoms is reasonably well-behaving: we want that every literal occurring in the list of preliminary goals will eventually be selected.

Clark's completion applies literally to programs with negation: no changes are needed, since its definition does not depend on the definiteness of the program. For the program P_{strat} consisting of the two clauses $p \leftarrow \neg q$ and $q \leftarrow \neg b$ we get the intuitively correct result:

$$comp(P_{strat}) = Th(\{\ p \leftrightarrow \neg q,\ q \leftrightarrow \neg b,\ b \leftrightarrow false\ \}) = Th(\{\ \neg b,\ q,\ \neg p\ \}).$$

In Fitting (1989), negation is treated as *refutation* and a tableau calculus is presented (and has been implemented) as a suitable semantics.

Note that $comp(P)$ may become inconsistent: $comp(p \leftarrow \neg p) = Th(\{p \leftrightarrow \neg p\}) = Fml$. In Przymusinski (1990b), various problems with

comp concerning the treatment of loops were discussed. See Sato (1990), Fages (1993) for sufficient conditions under which $comp(P)$ has a Herbrand-model.

Another weakness of the completion approach is its inability to characterize the transitive closure correctly (see Przymusinski (1989b), Kunen (1990)). Indeed, Kunen has proved that there is no *strict* (see Definition 2.7) normal program without function symbols that correctly expresses the transitive closure. But note that the property of *"having a cycle"* is expressible using COMP, although it is not uniformly expressible in first-order-logic (see Kunen (1990) for more details and the exact definition of *being expressible with COMP*).

2.3 Procedural Inference: SLDNF.

The main idea in this section is to use the benefits of SLD and to try to extend it to a new procedure SLDNF by setting

SLDNF = SLD + Neg.-as-Finite-Failure: $\neg L$ succeeds, if L finitely fails.

But the precise definitions of SLDNF-*resolution*, *tree*, etc. are very complex: see Lloyd (1987), Apt (1990) for the exact definitions. Recently, Apt and Bol gave interesting improved versions of these notions: see (Apt and Bol, 1994, Section 3.2). In order to get an intuitive idea, it is sufficient to describe the following underlying principle:

Principle 2.6 (A "Naive" SLDNF-Resolution).
If in the construction of an SLDNF-tree a negative literal $\neg L_{ij}$ is selected in the list $\mathcal{L}_i = \{L_{i1}, L_{i2}, \ldots\}$, then we try to prove L_{ij}.
If this fails finitely *(it fails because the generated subtree is finite and failing), then we take $\neg L_{ij}$ as proved and we go on to prove $L_{i(j+1)}$.*
If L_{ij} succeeds, then $\neg L_{ij}$ fails and we have to backtrack to the list $\mathcal{L}_{i-1}$ of preliminary subgoals (the next rule is applied: "backtracking").

As an example, let us consider the program P consisting of the three facts $p(c,c)$, $q(b)$ and $r(f(c))$. We ask $? - p(x,c), \neg q(x), r(f(x))$, that is, we are interested in instantiations of x such that the query follows from the program. The situation is illustrated in Figure 5. The SLDNF-tree of this trivial example is linear and has three nodes[3]: the first node is the query itself $?-\underline{p(x,c)}, \neg q(x), r(f(x))$, the second node is $?-\underline{\neg q(c)}, r(f(c))$. Now, we enter the negation-as-failure mode and ask $? - \underline{q(c)}$. This query immediately fails (the generated tree exists, is finite and fails) so that we give back the answer "yes, the literal $\underline{\neg q(c)}$ succeeds and can be skipped from the list". The last node is $? - \underline{r(f(c))}$ which immediately succeeds.

[3]Let us suppose that we always select the first literal: it is underlined in the sequel.

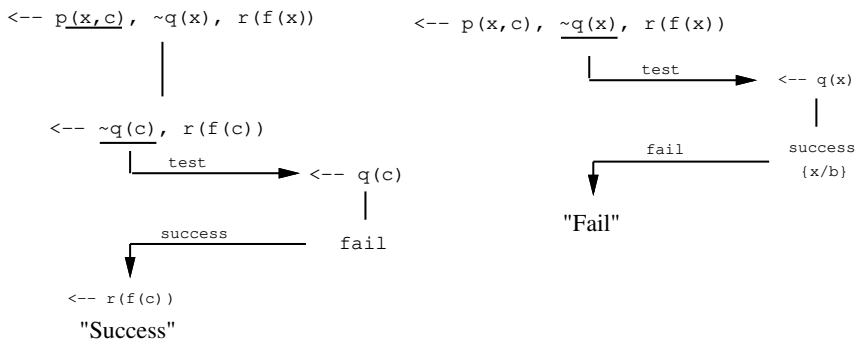

Figure 5: The Floundering-Problem

Note that in the last step, the test for $? - q(c)$ has to be finished before the tree can be extended. If we get no answer, the SLDNF-tree simply does not exist: this can not happen with SLD-trees.

A major problem is the treatment of free variables in negative literals: the *floundering problem* (see also Decker (1989)). If we select the second literal in the first step $? - p(x, c), \underline{\neg q(x)}, r(f(x))$, then we will enter the negation-as-failure mode with the query $? - \neg q(x)$. In this case, x may be instantiated to b so that we have to give back the answer "no, the literal $\neg q(x)$ fails" and the whole query will fail. This is because SLDNF treats the subgoal as "$\forall x \neg q(x)$" instead of "$\exists x \neg q(x)$" which is intended.

Chan's *constructive negation* (see Chan (1988), Chan and Wallace (1989), Ludäscher (1991), Vorbeck (1991), Drabent (1994)) attempts to solve the floundering problem by computing the answers for $q(x)$ and then giving back a set of constraints $(x \neq t_1(y), \ \ldots \ , x \neq t_n(y))$. These constraints have to be respected in the further SLDNF construction. See also Ginsberg (1991) for still another approach to deal with free variables. In the classical SLDNF-resolution however,

- negation-as-finite-failure is only a test, no bindings are produced.

On the one hand this may be considered a shortcoming, on the other hand, it makes the SLDNF procedure more tractable. Note that the problem to decide if a given program flounders is undecidable (shown in Börger (1987)). See also Shepherdson (1991b) for more unsolvable problems related to SLDNF.

2.4 Model-theoretical Semantics: Φ_P and $\mathbf{comp}_3(P)$.

It is not trivial to define a canonical model for programs containing negation: unlike in the case for definite programs, there is no *least* Herbrand-

model: in general there are various minimal supported ones. In addition, T_P is not monotone and COMP may become inconsistent. In the logic-programming community three-valued extensions of COMP were defined by Fitting and Kunen in order to solve the inconsistency problem: a shift from the well-known two-valued operator T_P to its three-valued analogue Φ_P. Some deep theorems showing the equivalences

- Herbrand($\mathcal{L}_\mathcal{P}$)-MOD($comp_3(P)$) $\cong$ $lfp(\Phi_P)$ *(Fitting's version)*,
- MOD($comp_3(P)$) $\cong$ $\{\Phi_P{\uparrow}^n$: for $n \in \mathbb{N}\}$ in a language $\mathcal{L}_{KU}$ *(Kunen's version)*, and finally
- MOD($comp_3(P)$) $\cong$ MOD($comp(P)$) for *strict* P *(Kunen)*

were established. Fitting's version can be seen as a three-valued analogue to the Herbrand-rule (Figure 3) while Kunen's version is a three-valued analogue of the negation-as-finite-failure-rule.

Fitting (1985) solved the inconsistency problem by considering a three-valued formulation $comp_3(P)$ of the completion: he showed that there always exists a unique $\leq_k$-least, minimal (three-valued) Herbrand-model of $comp_3(P)$. This model is the least fixpoint of a $\leq_k$-monotone operator Φ_P refining T_P. To be more precise, we define the operator $\Phi_P : \mathbf{3}_k{}^{B_P} \longmapsto \mathbf{3}_k{}^{B_P}$; $\mathcal{I} \longmapsto \Phi_P(\mathcal{I})$ as follows: (A is a ground atom)

$$\Phi_P(\mathcal{I})(A) = \begin{cases} \mathbf{t}, \text{ if there is a clause } A \leftarrow L_1, \ldots, L_n \text{ in } P_{inst} \text{ with:} \\ \quad - \ \forall i \leq n \text{ we have:} \\ \quad\quad * \ \mathcal{I}(L_i) = \mathbf{t}. \\ \mathbf{f}, \text{ if for all clauses } A \leftarrow L_1, \ldots, L_n \in P_{inst} \text{ we have:} \\ \quad - \ \exists i \leq n \text{ with:} \\ \quad\quad * \ \mathcal{I}(L_i) = \mathbf{f}. \\ \mathbf{u}, \text{ otherwise.} \end{cases}$$

For *definite* programs P, we have the following relationship with T_P:

$$\Phi_P{\uparrow}^\alpha \ = \ (T_P{\uparrow}^\alpha, B_P \setminus T_P{\downarrow}^\alpha).$$

As there are normal programs P with $lfp(\Phi_P) = \Phi_P{\uparrow}^\gamma$ (where $\gamma \geq \omega$), Φ_P is monotone, but <u>not</u> continuous. Thus, *Fitting's semantics* "$lfp(\Phi_P)$" is not recursive enumerable in general.

To get an enumerable semantics, Kunen (1987) considered the finite powers of Φ_P: $\bigcup_{n\in\mathbb{N}} \widehat{\Phi_P\uparrow^n}$ instead of "$lfp(\Phi_P)$". He also extended the underlying language "$\mathcal{L}_P$" by new function-symbols of any arity:

$$\mathcal{L}_{KU} =_{def} \mathcal{L}_P \cup \{f_j^i : f_j^i \text{ new i-ary function symbols}, i,j \in \mathbb{N}\}$$

and proved that for[4] $SEM_{Kunen} := \bigcup_{n\in\mathbb{N}} \widehat{\Phi_P\uparrow n}$, and Ψ a $\mathcal{L}_{KU}$-sentence:

Kunen's Theorem: $comp_3(P) \models \Psi$ **iff** $SEM_{Kunen} \models \Psi$.

Fitting's approach seems to be very elegant: just take the least fixpoint of Φ_P as the intended *canonical* three-valued model of P. This fixpoint is the $\leq_k$-least Herbrand-model of P (we use the ordering defined by $\mathbf{u} \leq_k \mathbf{t}$, $\mathbf{u} \leq_k \mathbf{f}$ on the truth values!). On the other hand, Kunen's variant is recursive enumerable: it corresponds to truth in all models (not only Herbrand-models) of $comp_3(P)$.

2.5 SLDNF versus COMP$_3$.

Kunen's theorem states that SEM_{Kunen} is sound and complete with respect to $comp_3(P)$. It turns out that in the propositional case we have

- SLDNF is sound and complete with respect to $comp_3(P)$.

In the predicate logic case, SLDNF is always correct (see (Shepherdson, 1988, Theorem 39)) with respect to $comp_3(P)$: given a query Q,

- if SLDNF succeeds with answer Θ, then $comp_3(P) \models_3 \forall Q\Theta$, and
- if SLDNF fails, then $comp_3(P) \models_3 \neg\exists Q$.

This correctness result is also the reason for the incompleteness of SLDNF with respect to two-valued $comp(P)$. It states that any formula derivable by SLDNF is a three-valued consequence of $comp_3(P)$. But, since there are two-valued consequences of a theory that are not three-valued ones (three-valued logic is weaker than two-valued logic), SLDNF can not be complete. In fact, Shepherdson has shown (see Shepherdson (1988), Shepherdson (1989)) that SLDNF is even correct with respect to a weaker logic than $\models_3$: a certain *intuitionistic* three-valued logic $\models_{3i}$.

Extensions of the above completeness result to certain subclasses of predicate logic programs require severe restrictions on the syntactic form of P. To define these syntactic restrictions, we need the notion of the dependency-graph:

[4] $\Phi_P \uparrow^n$ is a subset of the Herbrand base, $\widehat{\Phi_P\uparrow^n}$ is the unique extension to an interpretation of the whole language.

Definition 2.7 (Dependency-Graph $\mathcal{G}_P$).
For a normal logic program P, the dependency graph $\mathcal{G}_P$ *is a finite directed graph whose vertices are the predicate symbols from P. There is a* positive *(respectively* negative*) edge from R to R' :iff there is a clause in P with R in its head and R' occurring positively (respectively negative) in its body. We also say*

- R depends on R' *if there is a path in $\mathcal{G}_P$ from R to R' (by definition, R depends on itself),*
- R depends positively *(resp.* negatively*) on R' if there is a path in $\mathcal{G}_P$ from R to R' containing only* positive *edges (resp.* at least one negative *edge).*

  (by definition R depends positively on itself),
- R depends evenly *(resp.* oddly*) on R' if there is a path in $\mathcal{G}_P$ from R to R' containing an* even *(resp.* odd*) number of negative edges (by definition R depends evenly on itself).*

A generalization of $\mathcal{G}_P$ is the infinite instantiated dependency graph $\mathcal{G}_P^{inst}$ *whose vertices are the elements of $B_{\mathcal{L}_P}$. The edges are defined analogously: instead of P one takes P_{inst}.*

The following properties of a program P turn out to be very important:

Definition 2.8 (Stratified, Strict, Call-Consistent).
We call a program stratified *if no predicate depends negatively on itself (equivalently, if there are no cycles containing at least one edge).*
We call it strict *if there are no dependencies that are both even and odd.*
Finally, we call it call-consistent *if no predicate depends oddly on itself (equivalently, if there are no odd cycles).*

There is another very restrictive notion: *hierarchical* programs (defined in Lloyd (1987)). This class of programs is a subclass of the stratified programs: no form of recursion is allowed. Hierachical programs correspond to relational databases. We have obviously

- *hierarchical* implies *stratified*,
- *stratified* implies *call-consistent*,
- *strict* implies *call-consistent*.

These properties are clearly decidable. However, their generalizations (defined as above, but using P_{inst} instead of P) *locally* stratified, *locally* call-consistent, and *locally* strict are not (see Cholak (1989), Cholak and Blair (1994)).

Strictness together with the following property turn out to be the most important restrictions that imply completeness results for SLDNF:

Prog. P	**Query A**	**Semantics**	**Completeness**
allowed + hierarch.	allowed	$comp(P)$	yes, no recursion
allowed + strat.	allowed	$comp(P)$	yes, if strict[5]
allowed	allowed	$comp_3(P)$	yes, w.r.t. $\vdash_3$
allowed + call-cons.	allowed	$comp(P)$	yes, if strict: $comp(P) \vdash \forall A$ **iff** $comp_3(P) \vdash_3 \forall A$

Table 1: Completeness for SLDNF

allowedness: every variable occurring in a clause must occur in at least one positive literal of the body.

Allowedness excludes constructs of the form $equal(x, x) \leftarrow$, while *strictness* excludes situations of the form $p(x) \leftarrow q(x)$, $p(c) \leftarrow \neg q(f(c'))$. Allowedness solves the floundering-problem. In addition, we get the following completeness result (due to Kunen (1989))

- SLDNF is complete w.r.t. $comp_3(P)$ if $P \cup \{Q\}$ is allowed.

Strictness of P implies equivalence of $comp_3$ and $comp$ (see Kunen (1989))

- $comp_3(P) \models_3 \forall Q\Theta$ iff $comp(P) \models \forall Q\Theta$

Much work was done in LP (see Decker and Cavedon (1990) and Barbuti and Martelli (1986)) to find other syntactically characterizable classes, for which SLDNF is also *complete*: see Decker and Cavedon (1990). Recently, Stärk (1994) introduced an interesting notion (so called *input/output dependencies*) and defined a framework that contains most of these results as special instances.

There exist proposals to redefine the completion (e.g. Dhung and Kanchanasut (1989b), Drabent and Martelli (1991), Wallace (1993)) or to transform programs into a certain normalform (see Wallace (1989a, 1989b)) in order to obtain completeness results for SLDNF. There are also attempts of changing the notion of completion and the underlying logic simultaneously: intuitionistic and modal logics are considered for this purpose in Plaza (1991), Shepherdson (1989).

[5]More precisely: if $P \cup \{\leftarrow A\}$ is strict.

In Cerrito (1992), an interesting axiomatization of PROLOG's successes and failures (as opposed to SLDNF) is given in linear logic (see also Ümit Yalçinalp and Sterling (1988)). A literal A succeeds (resp. fails) iff A (resp. $\neg A$) is derivable from the linear theory associated to P.

2.6 Summary.

- *Procedural side:* augment SLD by *Neg.-as-F.-F.* to obtain SLDNF.
- *Declarative side:* use *completion* as an appropriate semantics.
- *Relationship:* Find classes of (syntactically defined) programs, for which soundness and completeness results hold.

3. Non-monotonic Reasoning Semantics

Besides the two problems of SLDNF mentioned in the beginning of Section 2 (*inconsistency* of COMP and *floundering*), a new problem arises from the viewpoint of non-monotonic reasoning: the existence of loops which should be detected as such. We address this problem in Example 3.1.

In Section 3.2, we consider the class of stratified programs with the semantics M_P^{supp}. Unfortunately, there are many programs that do not belong to this class but still have an intuitive meaning. Example 3.5 presents such a class of programs. The effort to define suitable semantics for these programs led to the definition of the stable semantics STABLE (Section 3.3) and the wellfounded semantics WFS (Section 3.4).

In Section 3.5 we shortly review some extensions of WFS, mainly motivated by the *reasoning-by-cases* problem. Finally, we consider in Section 3.6 disjunctive programs and illustrate the motivations underlying the most important semantics.

3.1 NMR-Intuition Compared with LP-Intuition.
The close relationship between COMP and SLDNF, for which COMP was invented, is best shown by the following:

Example 3.1 (COMP vs. NMR).

$$P_{NMR}: \quad p \leftarrow p \qquad\qquad P'_{NMR}: \quad p \leftarrow p$$
$$q \leftarrow \neg p \qquad\qquad q \leftarrow \neg p$$
$$r \leftarrow \neg r$$

$$comp(P_{NMR}): \quad p \leftrightarrow p \qquad\qquad comp(P'_{NMR}): \quad p \leftrightarrow p$$
$$q \leftrightarrow \neg p \qquad\qquad q \leftrightarrow \neg p$$
$$r \leftrightarrow \neg r$$

?-q: *No (COMP).* *Yes (NMR).* — *?-p*: *Yes (COMP).* *No (NMR).*

For both programs, the answers of the completion-semantics do not match our NMR-intuition! In the case of P_{NMR} we expect q to be derivable, since we expect $\neg p$ to be derivable: the only possibility to derive p is the rule $p \leftarrow p$ which, obviously, will never succeed. But $q \notin Th(\{q \leftrightarrow \neg p\}) = comp(P_{NMR})$! In the case of P'_{NMR} we expect p not to be derivable, for the same reason: the only possibility to derive p is the rule $p \leftarrow p$. But $p \in Fml = Th(\{r \leftrightarrow \neg r\}) = comp(P'_{NMR})$!

Note that the answers of the completion-semantics agree with the mechanism of SLDNF: $p \leftarrow p$ represents a loop. The completion of P' is inconsistent: this led Fitting to consider the three-valued version of $comp(P)$ introduced in the last section. This approach avoids the inconsistency (the query $? - p$ is not answered "yes") but it still does not answer "no" as we would like to have.

The last example motivates the need for a stronger semantics and shows the difficulties in obtaining one: loops should be detected. In general, this problem is undecidable because the halting problem reduces to it. This is also the reason why the stable or wellfounded semantics are, in the predicate logic case, not recursive (not even recursive enumerable, see Section 3.7). In addition, these two extensions of SLDNF do not agree on *which* loops should be detected. Let us underline the following idea:

- *NMR-Semantics = SLDNF + Loop-check*

3.2 M_P^{supp}: Semantics for Stratified Programs.

The problem in applying T_P to an arbitrary program is that at a stage i an atom A may still be false (so that it gives rise to derive new positive atoms via a clause of the form $A_{new} \leftarrow \neg A$) but that on a later stage $j > i$ the atom A may become true, so that all previously drawn inferences have to be rejected: this is exactly caused by the non-monotonicity of T_P! Let us consider

Example 3.2 (Positive and Negative Cycles).

$$P_{p \leftarrow p}: \quad p \leftarrow p, \qquad P_{p \leftarrow \neg p}: \quad p \leftarrow \neg p$$

In the first program, p depends positively on itself while in the second, p depends negatively on itself. Therefore the only reason to assume p in the first program is p itself (hence p should not be derivable).

In the second program, the assumption $\neg p$ implies a contradiction (hence $\neg p$ should not be derivable) but what should be assumed about p?

Note that $comp_3$ assigns p in the first program the value **u** (undefined). The problems with positive and negative cycles can be avoided for an interesting large class of programs (see Apt et al. (1988)). The idea is to

rule out *all* programs having a cycle (not only a direct negative link) with a negative edge in their dependency graph, as for example the program $P_{unstrat}$: $p \leftarrow \neg q$, $q \leftarrow \neg p$. Programs without such cycles, such as P_{strat}: $p \leftarrow \neg q$, $q \leftarrow \neg b$ (see Section 2.2), induce a natural priority ordering on their relation-symbols. If however, the clause $b \leftarrow p$ is added, the resulting program is no more stratified: there is a cycle with a negative edge between p and p. Equivalently, stratification may be formulated as follows

Definition 3.3 (Stratification).
A program P is called stratified, *if it is decomposable as $P = P_1 \cup \cdots \cup P_n$, such that the following holds (for i=1, ...,n):*

1. *If a relation-symbol* R *occurs* positively *in a clause in P_i, then all clauses containing* R *in their heads are contained in $\bigcup_{j \leq i} P_j$.*
2. *If a relation-symbol* R *occurs* negatively *in a clause in P_i, then all clauses containing* R *in their heads are contained in $\bigcup_{j < i} P_j$.*

We say: "P is stratified via $P_1 \cup \cdots \cup P_n$" and call P_i "stratum of P".

More recently, Fitting also defined notions related to stratification (see Fitting and Ben-Jacob (1990)). Programs describing the transitive closure, or even the difference of two transitive hulls, are stratified and hence handled correctly by the stratified semantics. But note that there are stratified programs P, such that M_P^{supp} is not recursive enumerable.

We defined in Subsection 2.2 the notion of a *supported* model (Definition 2.3). Unfortunately, not even for stratified programs does exist a *least* supported model: the program consisting of the two clauses "$p \leftarrow \neg q$" and "$q \leftarrow q$" has two minimal supported models, namely $\{q\}$ and $\{p\}$.

But it is possible to define *powers* of T_P and to use them for selecting a unique Herbrand-model M_P^{supp} among all minimal Herbrand-models of $comp(P)$: the supported semantics is defined by declaring M_P^{supp} (see Apt et al. (1988)) as the unique canonical or intended model:

Theorem 3.4 (Apt, Blair, Walker: M_P^{supp}).
For every stratified program P, a unique supported Herbrand-model M_P^{supp} of P can be constructed.

In particular, $comp(P)$ is *consistent* for stratified programs. The supported semantics treats positive loops $p \leftarrow p$ as we would expect: **f** (false) is assigned to p in this case. This means that *negation-as-finite-failure* should be enhanced by a mechanism that *detects* loops of this sort. Instead of only failing p, the program should be transformed such that p *finitely* fails. The idea of loop-detection is formally carried out and investigated at length in Bol (1991) for stratified programs. A method for implementing the supported semantics is discussed in Kemp and Topor (1988).

The question arises whether there are extensions of the supported semantics defined for all programs? Let us take a look at the following

Example 3.5 (Non-Strat. Program with Unique Semantics).

$$\begin{array}{lll} P_{default}: & v(a,b) & \\ & v(b,c) & \\ & e(c) & \\ & e(x) & \leftarrow \quad v(x,y), \neg e(y) \end{array}$$

$P_{default}$ is not even locally stratifiable, because of $e(a) \leftarrow v(a,a), \neg e(a)$. Nevertheless it has an intuitive model:

$$\mathcal{M}_{P_{default}} = \{v(a,b), v(b,c), e(a), e(c)\}.$$

The clause $e(x) \leftarrow v(x,y), \neg e(y)$ can be interpreted as the description of a game, where the moving player wins, when his opponent has no more move to make[6]*: the clause*

- *wins(x) ← move_from_to(x,y), ¬ wins(y)*

has to be read as

- *the situation x is won (for the moving player A), if he can lead over*[7] *to a situation y that can never be won for B.*

Note that cycles in the relation move_from_to(,), or in v(x,y) respectively, make difficulties. Such cycles are handled differently by STABLE and WFS, as we will see below.

In order to give such programs a meaning, the *stable* and the *wellfounded* semantics were defined. As already mentioned in Section 3.1, they can be seen as extensions of SLDNF by detecting loops.

3.3 STABLE: The Stable Semantics.

Let us refer to the last example in the previous section and consider the program P, consisting of the clause

$$wins(x) \leftarrow move_from_to(x,y), \ \neg\, wins(y)$$

together with the facts

$$move_from_to(a,b), \ move_from_to(b,a), move_from_to(b,c).$$

Intuitively, we would consider the model $\{\neg wins(c), wins(b), \ \neg wins(a)\}$ to be the intended one, because any atom has a definite reason to be true

[6] As in checkers; see van Gelder et al. (1988).

[7] With the help of a regular move, given by the relation $move_from_to(,)$.

or false. This idea was made explicit in Bidoit and Froidevaux (1991a, 1991b) and, independently, Gelfond and Lifschitz (1988). We use the latter terminology and introduce the Gelfond-Lifschitz transformation: for a program P and a model $N \subseteq B_P$ we define

$$P^N := \{rule^N : \ rule \in P\}$$

where

$$(A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m)^N := \begin{cases} A \leftarrow B_1, \ldots, B_n, & \text{if } \forall j : C_j \notin N, \\ \mathbf{t}, & \text{otherwise.} \end{cases}$$

Note that P^N is always a *definite* program. We can therefore compute its least Herbrand model M_{P^N} and check whether it coincides with the model N with which we started:

Definition 3.6 (STABLE).
N is called a stable *model of P, if $M_{P^N} = N$.*

Alternatively, stable models can also be defined as fixpoints of $T_P^{\mathcal{J}}$, an extension of the well-known T_P operator introduced in Section 2.1 (see Przymusinski (1991e)). It is easy to see that stable models are supported, and that they are minimal models of $comp(P)$ (see Marek and Subrahmanian (1989)). Unfortunately, stable models do not always exist:

Example 3.7 ((Non-) Existence of Stable Models).

$$P_{\neg\exists\, stab.}: \begin{array}{lll} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a \\ p & \leftarrow & \neg p \end{array} \qquad P_{stable}: \begin{array}{lll} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a \\ p & \leftarrow & \neg p \\ p & \leftarrow & a \end{array}$$

$P_{\neg\exists\, stab.}$ has no stable models. P_{stable} has the unique stable model $\{p, a\}$.

Sufficient criteria for the existence of stable models are contained in Dung (1992), Fages (1993).

Example 3.8 (Adding Irrelevant Clauses).

$$P_{stratified}: \ a \leftarrow \neg b \qquad P_{no_stable_model}: \begin{array}{lll} a & \leftarrow & \neg b \\ p & \leftarrow & \neg p \end{array}$$

The unique stable model of $P_{stratified}$ coincides with the supported model: therefore, a is derivable. If the clause $p \leftarrow \neg p$ is added, a is no longer derivable because no stable model exists.

The last example shows that the truthvalue of an atom a also depends on atoms that are totally unrelated with a: the principle of *Relevance* (see Section 5.3) is not satisfied. One might think that this problem can be easily solved by redefining STABLE in the case where no stable models exist. The easiest way is to take the wellfounded model WFS(P) (to be introduced in the next section) and to define

$$\text{STABLE}^*(P) := \begin{cases} \text{WFS}(P), & \text{if no stable model exists,} \\ \text{STABLE}(P), & \text{otherwise.} \end{cases}$$

Unfortunately this only solves the problem with the two particular programs above. More complex programs, showing that even STABLE* suffers from the same shortcomings, can be easily constructed. We will come back to this point in Section 5.3. In Dix and Müller (1994b) we define a version of STABLE where the truthvalue of any atom a only depends on those atoms that are *relevant* for a. We refer to this version as STABLE^{rel}.

The close relationship of STABLE with the theory of truth-maintenance (TMS) has been established in Elkan (1990) (see also Pimentel and Cuadrado (1989)).

For recent attempts to extend the stable semantics by *classical* or *explicit* negation and introducing the concept of *answer-sets*, see Gelfond and Lifschitz (1990), Gelfond (1991), Gelfond and Lifschitz (1991), Jonker (1993), Minker and Ruiz (1994), Costantini and Lanzarone (1995a). Lifschitz (1992) contains an interesting representation of STABLE using epistemic operators (see also Lifschitz (1991)).

In Subrahmanian et al. (1992) it is shown how stable models can be obtained from WFS by *Branch and Bound*.

In Brass and Dix (1994a, 1994b, 1995c, 1995b, 1995a) characterizations of the disjunctive stable semantics using the principle of *partial deduction* and *supported models* are given. They also derive interesting computational procedures for STABLE based on Clark's completion. David S. Warren's group is also working on implementations of STABLE using tabulation mechanisms.

Some complexity results about STABLE are contained in Marek and Subrahmanian (1989), Marek and Truszczyński (1993): the problem of deciding whether a given model is stable, is Π_2^0-hard.

3.4 WFS: The Wellfounded Semantics.

In van Gelder et al. (1988) for every program P a partial model was constructed using the notion of *greatest unfounded set*. This definition was reconstructed in 3-valued logic using $\Phi_P^{\mathcal{J}}$, a generalization of Φ_P (see Section 2.4), from Przymusinski in Przymusinski (1989a, 1991e, 1990a) (see also Sacca and Zaniolo (1991)).

Przymusinski showed that both the stable and the wellfounded semantics can be seen as fixpoints of the "same" extension of different operators: fixpoints of an extension of T_P in the case of STABLE and fixpoints of an extension of Φ_P in the case of WFS. The following schema characterizes the work of Przymusinski (1989a, 1991e, 1990a) (see also Dix (1991b)):

$$\begin{array}{ccc} \boldsymbol{T}_P & \leadsto & \boldsymbol{T}_P^{\mathcal{J}} \\ \textit{(two-valued)} & & \textit{(two-valued)} \\ P \text{ stratified: } M_P^{supp} & & P \text{ normal: STABLE} \end{array}$$

$$\begin{array}{ccc} \boldsymbol{\Phi}_P & \leadsto & \boldsymbol{\Phi}_P^{\mathcal{J}} \\ \textit{(three-valued)} & & \textit{(three-valued)} \\ P \text{ normal: } M_P^{3-val} & & P \text{ normal: WFS} \end{array}$$

There is an equivalent approach using alternating fixpoints due to van Gelder, see van Gelder (1989). Recently, a very elegant reconstruction in two-valued logic using minimal model reasoning (see Przymusinski (1991a)) and an extension using a 4-valued logic (or more generally, using the notion of a bilattice) were also proposed: see Fitting (1992).

While stable models are 2-valued, the wellfounded model is a 3-valued one defined using the following operator $\Phi_P^{\mathcal{J}}$:

$$\Phi_P^{\mathcal{J}}(\mathcal{I})(A) = \begin{cases} \mathbf{t}, \text{ if there is a clause } A \leftarrow L_1, \ldots, L_n \text{ in } P_{inst} \text{ with:} \\ \quad - \forall i \leq n \text{ we have:} \\ \quad\quad * \ \mathcal{J}(L_i) = \mathbf{t}, \\ \quad\quad \underline{\text{or}} \\ \quad\quad * \ L_i \text{ is positive, } \mathcal{J}(L_i) \neq \mathbf{f} \text{ and } \mathcal{I}(L_i) = \mathbf{t}, \\ \mathbf{f}, \text{ if for all clauses } A \leftarrow L_1, \ldots, L_n \in P_{inst} \text{ we have:} \\ \quad - \exists i \leq n \text{ with:} \\ \quad\quad * \ \mathcal{J}(L_i) = \mathbf{f} \\ \quad\quad \underline{\text{or}} \\ \quad\quad * \ L_i \text{ is positive, } \mathcal{J}(L_i) \neq \mathbf{t} \text{ and } \mathcal{I}(L_i) = \mathbf{f}, \\ \mathbf{u}, \text{ otherwise.} \end{cases}$$

$\mathcal{J}$ represents the *safe* knowledge: True($\mathcal{J}$) and False($\mathcal{J}$) never decrease. $\mathcal{I}$ stands for the current knowledge; applying $\Phi_P^{\mathcal{J}}$ to $\mathcal{I}$ and iterating this further gives us new information (False($\mathcal{I}$) decreases). Positive and negative

information is handled symmetrically[8].

Przymusinski proved that for all three-valued interpretations $\mathcal{J}$, the operator $\Phi_P^{\mathcal{J}}$ is $\leq_t$-*monotone* on ${\mathbf{3}_t}^{B_P}$ and has a *unique* $\leq_t$-*least fixpoint*:

$$\mathit{lfp}(\Phi_P^{\mathcal{J}}) = \Phi_P^{\mathcal{J}}{\uparrow}^{\omega}, \text{where } \Phi_P^{\mathcal{J}}{\uparrow}^0 = \langle \emptyset, B_P \rangle.$$

Note that $\Phi_P^{\mathcal{J}}$ is also $\leq_k$-monotone. He then defined the $\leq_k$-monotone operator Ω_P

$$\Omega_P : {\mathbf{3}_k}^{B_P} \longmapsto {\mathbf{3}_k}^{B_P}; \mathcal{J} \longmapsto \Phi_P^{\mathcal{J}}{\uparrow}^{\omega} .$$

Ω_P has a unique $\leq_k$-least fixpoint, which is exactly the wellfounded model from *van Gelder, Ross and Schlipf*:

$$\mathrm{WFS}(P) = \mathit{lfp}(\Omega_P).$$

This least fixpoint is a consistent three-valued model, because it is the iteration of a consistency preserving operator. WFS is closely related to STABLE:

- Every stable model $\mathcal{N}$ of P extends WFS(P): WFS(P) $\leq_k \mathcal{N}$.
- If WFS(P) is two-valued, WFS(P) is the unique stable model.

Let us again refer to Example 3.5 and consider the program P consisting of the clause

$$\mathit{wins}(x) \leftarrow \mathit{move_from_to}(x, y),\ \neg \mathit{wins}(y)$$

together with the facts

$$\mathit{move_from_to}(a, b),\ \mathit{move_from_to}(b, a), \tag{1}$$
$$\mathit{move_from_to}(b, c),\ \mathit{move_from_to}(c, d). \tag{2}$$

In this particular case we have two stable models, and

$$\mathrm{WFS}(P) = \bigcap_{\mathcal{N} \text{ a stable model of } P} \mathcal{N}$$

This is not always the case, as the program $P_{splitting}$ defined in the next section will show.

Note that depending on the interpretation of the nonexistence of stable models, WFS and STABLE might be incomparable with respect to our relation $\leq_k$: in Example 3.8 we can derive a from $P_{no_stable_model}$ using WFS but not using STABLE. Using our modification STABLE* however, we get a more regular behaviour: WFS(P) $\leq_k$ STABLE*(P). But we can

[8] As before, negative literals are only evaluated by $\mathcal{J}$!

also view the inconsistency of STABLE as *deriving anything*: under this interpretation we have

$$\mathrm{WFS}(P) \leq_k \mathrm{STABLE}^*(P) \leq_k \mathrm{STABLE}(P).$$

Using the notions of *strictness* and *call-consistency*[9] Dung proved in Dung (1992):

- If P is call-consistent then there exists a stable model.
- If P is strict then $\mathrm{WFS}(P) = \bigcap_{\mathcal{N} \text{ a stable model of } P} \mathcal{N}$.

There is also a procedural mechanism suitable for WFS, the so-called *SLS-resolution*: see Przymusinski (1989b), Ross (1992). While SLS-resolution is only a theoretical construct, there is an implementation in QUINTUS-PROLOG of the wellfounded semantics (which is complete for nonfloundering Datalog programs) based on an other method: the OLDT-algorithm of Tamaki and Sato (1986). This implementation is comprehensively presented in Warren (1991), Przymusinski and Warren (1992). A related approach is contained in Chen and Warren (1992, 1993), Chen et al. (1995), Bol and Degerstedt (1993). Another method using *magic sets* is presented in Kemp et al. (1991), Degerstedt and Nilsson (1995).

Extending the wellfounded semantics by *explicit* negation (as first proposed for STABLE in Gelfond and Lifschitz (1990)) is considered in Pereira et al. (1991), Dung and Ruamviboonsuk (1991), Alferes and Pereira (1992). The usefulness and expressiveness of extended programs is discussed in Gelfond and Lifschitz (1992), Jonker (1993), Minker and Ruiz (1994). The forthcoming book Alferes and Pereira (1995b) gives a comprehensive treatment of WFS for normal programs with explicit negation. They also describe procedural aspects and describe implementations in PROLOG. For more on the complexity of extended programs the reader may consult Marek et al. (1992).

3.5 Other Semantics.

Any sceptical semantics based on the intersection of two-valued models inherits disjunctions. Some people argue that a reasonable semantics should rely on this. The wellfounded model, being a three-valued one, does not:

Example 3.9 (Reasoning by cases).

$$\begin{array}{lllll} P_{splitting}: & a & \leftarrow & \neg b \\ & b & \leftarrow & \neg a \\ & p & \leftarrow & a \\ & p & \leftarrow & b \end{array}$$

[9] These notions are introduced in Section 2.5: the latter is implied by the former.

Although neither a, nor b can be derived in any semantics based on two-valued models (as STABLE for example), the disjunction $a \vee b$, thus also p, is true. In this way the example is handled by the completion semantics, too. WFS(P), however, is empty; if the WFS cannot decide between a or $\neg a$, then a is undefined.

Baral et al. (1990c) defined GWFS in order to solve the "reasoning by cases" problem: GWFS is an extension of WFS, in the sense that WFS(P) $\leq_k$ GWFS(P). In Dix (1992a) we also defined two extensions of WFS to solve this problem: WFS$^+$ and WFS$'$. Both semantics are related to GWFS but avoid some undesirable behaviour to be described below. Let us first illustrate the common idea underlying these definitions. The simplest way of extending WFS to allow reasoning-by-cases is by using minimal model reasoning. There are, however, various possibilities. The simplest one is first evaluating WFS and then adding all literals that are true in all minimal models extending WFS(P): this defines EWFS, a precursor of WFS$^+$ defined in Dix (1991b). Now, this procedure may add new literals in such a way that a further application of WFS still yields new literals: using the terminology introduced in Section 5.2, the *Cut* is not satisfied. In order to get some *"closure"* of this process, Baral, Lobo and Minker defined GWFS.

While GWFS satisfies *Cut* but not *Cautious Monotony*, EWFS is cautious but fails to satisfy *Cut*. GWFS has another strange behaviour: it does not satisfy the *principle of partial evaluation*. All these properties will be introduced below.

The exact realization of the intuitive idea of extending WFS by adding all literals true in all minimal models extending WFS is by using the operator

$$\Omega_P : {\mathbf{3}_k}^{B_P} \longrightarrow {\mathbf{3}_k}^{B_P}; \mathcal{J} \longmapsto \Phi_P^{\mathcal{J}} \uparrow^{\omega},$$

which is based on the definition of

$$\Phi_P^{\mathcal{J}} : {\mathbf{3}_t}^{B_P} \longrightarrow {\mathbf{3}_t}^{B_P};\ \mathcal{I} \longmapsto \Phi_P^{\mathcal{J}}(\mathcal{I})$$

as defined in Section 3.4. Note that WFS(P):= $lfp(\Omega_P)$. GWFS is then defined as follows:

Definition 3.10 (GWFS, T($\mathcal{J}$), F($\mathcal{J}$)). *For a three-valued interpretation $\mathcal{J}$, let $\mathcal{J}-$MIN-MOD(P) denote the class of all minimal two-valued Herbrand models of P that are consistent with $\mathcal{J}$. Furthermore, let*

- $\mathbf{T}(\mathcal{J}) :=$ *True($\mathcal{J}-$MIN-MOD(P)), and*
- $\mathbf{F}(\mathcal{J}) :=$ *False($\mathcal{J}-$MIN-MOD(P)).*

Here, for a set $\mathcal{S}$ of Herbrand models, True($\mathcal{S}$) (resp. False($\mathcal{S}$)) stands for the set of all ground atoms A, which are true (resp. the negations of which are true) in all models in $\mathcal{S}$[10].

We define

$$EWFS(P) := WFS(P) + \langle \mathbf{T}(WFS(P)), \mathbf{F}(WFS(P)) \rangle,$$

$$GWFS(P) := lfp(\Omega_P^G),$$

where the operator Ω_P^G is defined as follows:

$$\Omega_P^G : \mathbf{3}_k{}^{B_P} \longmapsto \mathbf{3}_k{}^{B_P}; \mathcal{J} \longmapsto \Phi_P^{\mathcal{J}} \uparrow^\omega + \langle \mathbf{T}(\mathcal{J}), \mathbf{F}(\mathcal{J}) \rangle.$$

EWFS is stronger than WFS (WFS $\leq_k$ EWFS) and it is also cautious (see Dix (1991a)). That it does not satisfy *Cut* can be seen with the following counterexample:

$$\begin{array}{lllll} P_{EWFS}: & a & \leftarrow & \neg a \\ & \beta & \leftarrow & \neg x, a \\ & y & \leftarrow & \neg \beta \\ & z & \leftarrow & \neg y \end{array}$$

We have WFS(P)−MIN-MOD(P_{EWFS})=$\{\{a, \beta, z\}, \{a, \beta, y\}\}$ because obviously WFS(P_{EWFS})=$\{\neg x\}$ Therefore we get EWFS(P_{EWFS})=$\{a,\ \beta,\ \neg x\}$ but WFS($P_{EWFS} \cup \{\beta\}$)=$\{\neg x, \neg y, z, \beta\}$ and thus EWFS($P_{EWFS} \cup \{\beta\}$) = $\{a,\ \beta,\ \neg x,\ \neg y,\ z\}$. Thus, by adding a derivable atom ("β"), we are able to derive more atoms than without: *Cut* is not satisfied.

We will now show that GWFS gives different results on programs that usually are considered to be equivalent (see also Section 5.3).

Example 3.11 (Extension-by-Definition).
We consider the following two programs:

$$\begin{array}{lllll} P_{GWFS}: & p & \leftarrow & \neg b \\ & a & \leftarrow & \neg b \\ & b & \leftarrow & c \\ & c & \leftarrow & p, \neg a \end{array} \qquad \begin{array}{lllll} P_{GWFS\,c}: & p & \leftarrow & \neg b \\ & a & \leftarrow & \neg b \\ & b & \leftarrow & p, \neg a \end{array}$$

GWFS(P_{GWFS}) entails $\neg c$, because MIN-MOD(P_{GWFS})=$\{\ \{p, a\},\ \{b\}\ \}$. Therefore GWFS also entails (by simple negation-as-failure reasoning) $\neg b$, p and a. But MIN-MOD($P_{GWFS} \cup \{p\}$)=$\{\ \{p, a\},\ \{p, c, b\}\ \}$, thus GWFS($P_{GWFS} \cup \{p\}$) does not entail $\neg c$.

[10] If $\mathcal{S} = \emptyset$, we define True($\mathcal{S}$) (resp. False($\mathcal{S}$)) to be equal to $\emptyset$.

$P_{GWFS\,c}$ *partial evaluates*[11] P_{GWFS}: the last two clauses were amalgamated in one single clause. Obviously, a semantics should assign the same meaning to these programs (see Section 5.3): unfortunately GWFS does not!

Therefore, GWFS adds too many literals. But there is the possibility to add only *positive* literals true in all minimal models[12] (after the computation of WFS) and iterating this further until no new literals can be derived. This yields our semantics WFS$^+$ (and a variant WFS$'$) that turns out to behave much more regular than GWFS.

The following program is due to Jack Minker. Carolina Ruiz noticed that it proves the incomparability of STABLE (and STABLE^{rel}) with GWFS:

Example 3.12 (STABLE $\not\leq_k$ GWFS).

$$
\begin{array}{lllll}
P_{STABLE/GWFS}: & a & \leftarrow & \neg b \\
 & b & \leftarrow & \neg a \\
 & p & \leftarrow & a, \neg c \\
 & p & \leftarrow & b \\
 & c & \leftarrow & \neg d \\
 & d & \leftarrow & a
\end{array}
$$

P has two stable models: $\{a, d, p\}$ and $\{b, c, p\}$, so that p is derivable. GWFS is not able to derive p, because $\{a, d, c\}$ is a possible minimal model (see Dix (1992a) for the details).

Baral, Lobo and Minker defined two further semantics for disjunctive programs (see the next section): GDWFS and WF^3. Our example also applies to these.

In Dix (1992a) the two above mentioned extensions of WFS were introduced: WFS$^+$ and WFS$'$ (we give the exact definitions of these semantics in Section 5.2). Both semantics are related to GWFS but avoid the undesirable behaviour just described. While WFS$^+$ assigns a to be true in the program $a \leftarrow \neg a$, WFS' is more cautious and assigns **u** (*undefined*) to a. Still a different extension of WFS was recently proposed by Pereira et al. : the O-Semantics O-SEM (see Pereira et al. (1992), Alferes and Pereira (1995a)):

Example 3.13 (O-SEM 1).

$$
\begin{array}{lllll}
P_{O_SEM\,1}: & a & \leftarrow & \neg a \\
 & c & \leftarrow & \neg a
\end{array}
$$

[11] See Section 5.4 for more on partial evaluation.

[12] This amounts obviously to adding all atoms true in $Th(P)$.

O-SEM assigns $P_{O_SEM\,1}$ the three-valued model $\{\neg c\}$, a is undefined. Therefore O-SEM is no three-valued model of $P_{O_SEM\,1}$: the clause $c \leftarrow \neg a$ is not assigned true! Adding a, we get $\{a, c\}$ as the associated model.

While $WFS^{+}(P_{O_SEM\,1}) = \{a, \neg c\}$, $WFS'(P_{O_SEM\,1})$ is empty!

Figure 6 illustrates the situation for *normal* programs. While there is no conflict about the right semantics for *positive* or *stratified* programs, there are various different semantics for the whole class of *normal* programs. In Section 5.2, we will see that WFS is, in a certain sense, the most regular semantics. WFS also turns out to be the weakest one (see Theorem 5.27): this is also illustrated in Figure 6, where all extensions of the supported semantics are $\leq_k$-extensions of WFS.

While most of the proposed semantics are indeed different (as can be shown by very simple programs) You and Yuan (1993) recently showed that some semantics coincide:

- You and Yuan's *regular models* (see You and Yuan (1990)),
- Sacca and Zaniolo's *partial models* (see Sacca and Zaniolo (1991)),
- Przymusinski's *$\leq_k$-maximal 3-valued stable models* (see Przymusinski (1991e, 1990a)), and
- Dung's *preferred extensions* (see Dung (1991)).

We use the notation REG-SEM (the semantics based on regular models) as a representative for these four semantics. The exact definition is as follows (see You and Yuan (1990, 1993)):

Definition 3.14 (Justified and Regular Models).
A three-valued Herbrand model $\mathcal{M}$ is justified for a program P, if every $a \in True(\mathcal{M})$ is justified. The notion of justifiedness of an atom a is defined inductively: a is justified if

- *there is a clause $a \leftarrow \neg c_1, \ldots, \neg c_m$ in P_{inst} such that $\{c_1, \ldots, c_m\} \subseteq False(\mathcal{M})$, or*
- *there is a clause $a \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m$ in P_{inst} such that $b_1, \ldots, b_n$ are already justified and $\{c_1, \ldots, c_m\} \subseteq False(\mathcal{M})$.*

We define

$$\begin{aligned} \textit{JUST-MOD}(P) &:= \{\mathcal{M} :\ \mathcal{M} \in MOD_{3-val}^{Herb_{\mathcal{L}}P}(P)\ ,\mathcal{M} \textit{ is justified for } P\} \\ \textit{REG-MOD}(P) &:= \{\mathcal{M} :\ \mathcal{M} \textit{ is } \leq_k\textit{-maximal in JUST-MOD}(P)\} \end{aligned}$$

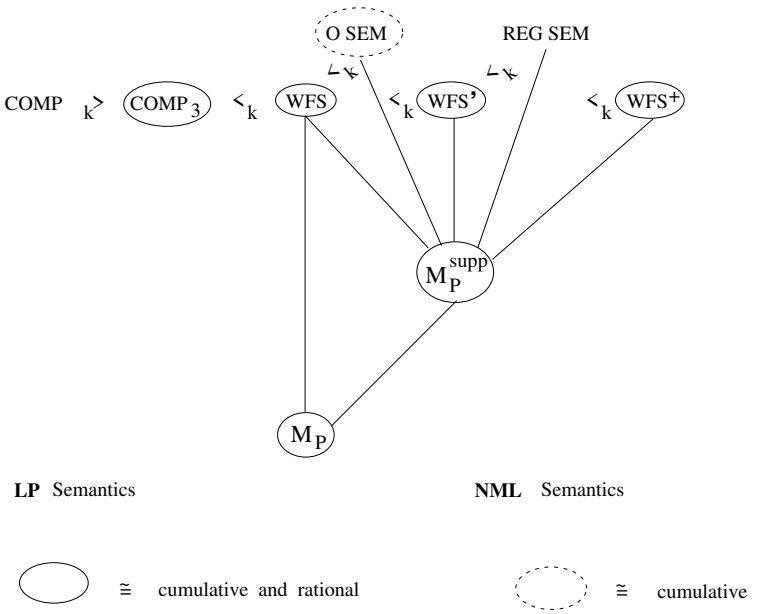

Figure 6: Semantics for Normal Programs

Przymusinski (1990a) proved that the induced consequence relation of all three-valued stable models coincides with WFS. The reader should note that all regular models are $\leq_k$-extensions of the wellfounded model and that, in addition, all stable models are regular. P_{stable} from Example 3.7 shows that REG-SEM is weaker than STABLE: REG-SEM($P_{\neg cum}$) = WFS($P_{\neg cum}$) = $\emptyset$. This is because $\langle\{a,p\};\{b\}\rangle$ and $\langle\{b\};\{a\}\rangle$ are the regular models. Therefore:

$$\text{WFS} \leq_k \text{REG-SEM} \leq_k \text{STABLE}.$$

Other extensions of WFS (WFS$_E$ and WFS$_S$) were defined in Chen and Kundu (1991), Hu and Yuan (1991). We will see in Section 5 that they do not satisfy the *Cut*. Schlipf defined in Schlipf (1990a, 1992b) a semantics WFS$_C$ that is equivalent to WFS$^+$ (see Dix (1995a, 1995b)). Two completely different but very interesting approaches were investigated in Geffner (1991, 1992) and in Baral and Subrahmanian (1992, 1991). While the work of Baral and Subrahmanian is more in the tradition of NMR-semantics (they assign to every program P a class of interpretations, not necessarily models, of P using Gelfond/Lifschitz operator $T_P^{\mathcal{J}}$), Geffner's approach was motivated by his work in causal and conditional theories:

Geffner (1989). His main aim is to develop a constructive proof theory based on the notion of an *argument*. This is strongly related to nonmonotonic TMS's. In contrast to all semantics presented so far, his theory is also applicable for programs containing negative literals in their heads.

Brewka, Makinson, and Schlechta (see Brewka et al. (1991)) defined a cumulative semantics for logic programs by adapting (thereby simplifying) the cumulative version of DL in Brewka (1991a) to logic programs. There is also the approach of Herre (1988, 1991) based on model-operators.

Finally, let us also mention some approaches using methods of *abduction* to assign a meaning to semantics to logic programs: Kakas and Mancarella (1991), Dung (1991), Eshgi and Kowalski (1989), Brewka (1993), Kakas et al. (1993).

3.6 Disjunctive Programs.
As already defined in Section 1.1, disjunctive logic programs are extensions of normal programs that allow arbitrary *positive* clauses to appear in the head of a rule:

$$A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_l$$

(see Lobo et al. (1989), Rajasekar et al. (1989a), Minker (1989)). All semantics for general disjunctive programs considered so far in the literature

- Baral, Lobo and Minker's GDWFS (Baral et al. (1990a, 1990b)), and WF^3 (Baral et al. (1991)),
- DSTABLE, the *disjunctive STABLE* semantics introduced independently in Gelfond and Lifschitz (1991) and Przymusinski (1991c), (we call it DSTABLE to distinguish it from STABLE for normal programs),
- Przymusinski's stationary and *static*-semantics $\mathcal{STN}$ and STATIC defined in Przymusinski (1991b, 1995), and
- DWFS of Dix (see Dix (1992b), Dix and Müller (1994a)) and D-WFS defined in Brass and Dix (1994c, 1995c),

coincide on the subclass of *positive* disjunctive programs: the respective semantics was first considered by Minker and is given by

$$SEM_P \;=\; \text{MIN-MOD}(P).$$

This minimal model semantics, called GCWA when applied to positive disjunctive programs, can also be seen as a weaker version of CWA that avoids the inconsistency-problem. For every program P, GCWA(P) is always consistent, which is not the case for CWA (see Section 4.1).

Chomicki and Subrahmanian (1990) show that GCWA is Π_2^0-complete so it lies between CWA (which is Π_1^0-complete, see Apt and Blair (1990) and general Circumscription (which is Σ_1^1-complete, see Cadoli et al. (1992)). For propositional programs we have to distinguish between deriving an *atom* or a *literal*. The first problem is co-NP-complete while the second is even Σ_2^P-complete (see Imielinski (1991)).

The motivation to define GDWFS or WF^3 are similar to the motivations underlying GWFS: finding an extension of WFS (for normal programs) in order to solve more complicated "reasoning by cases" problems. Let us illustrate this with an example of Baral et al. (1990a, 1990b): we consider the program P_{GDWFS}

$$\begin{array}{lllll} P_{GDWFS}: & a & \leftarrow & \neg b, c, d \\ & b & \leftarrow & \neg a \\ & c & \leftarrow & \neg d \\ & d & \leftarrow & \neg c \end{array}$$

Baral, Lobo and Minker's intuition about P_{GDWFS} is that a is false and b is true, because $c \wedge d$ is false in every minimal model (and therefore the only clause supporting a can not be applied). Besides the problem of *partial evaluation* inherited from GWFS, Chen showed that there is a bug in the definition of GDWFS (the resulting model may be inconsistent) and proposed a refinement in Chen (1992). See also Seipel and Argenton (1993), Seipel (1994) for more on implementation-issues of these semantics.

Analogously to the situation for programs without disjunction, there is also a notion of *stratified* disjunctive databases together with canonical models associated to it: the *perfect* models (see Przymusinski (1988)) defined by Przymusinski. In Figure 10, the semantics PERFECT is the counterpart of Apt, Blair and Walker's supported semantics. This is partly justified by Rajasekar and Minker (1989), where the methods of Apt et al. (1988) were successfully extended to disjunctive databases: the semantics GCWAS constructed there, however, is weaker than PERFECT, which corresponds to prioritized Circumscription (the priority relation is induced from the stratification of the program, see Section 4.2):

Example 3.15 (GCWAS $\leq_k$ PERFECT).

$$\begin{array}{lllll} P_{strat.\ excl.}: & x & \leftarrow & \neg a \\ & x & \leftarrow & \neg b \\ & a \vee b & & \end{array}$$

PERFECT derives x from $P_{strat.\ excl.}$: The perfect models are $\{a, \neg b, x\}$ and $\{\neg a, b, x\}$. But GCWAS does not derive x.

Przymusinski's stationary approach $\mathcal{STN}$ avoids the problems of GWFS, GDWFS and WF^3 (see Example 3.11) related to *partial evaluation*. This approach is very elegant: for normal programs, $\mathcal{STN}$ turns out to be equivalent to WFS. The advantage of this new construction is that it also applies to *arbitrary* disjunctive programs. There is, however, a problem: the straightforward generalization does not extend PERFECT.

Example 3.16 ($\mathcal{STN}$ without DIR).

$$
\begin{array}{lll}
P_{or}: & a & \leftarrow \; e, \neg g \\
 & a & \leftarrow \; f, \neg g \\
 & a & \leftarrow \; f, \neg e \\
 & a & \leftarrow \; g, \neg e \\
 & e \vee f \vee g &
\end{array}
$$

PERFECT derives a from P_{or}; $\mathcal{STN}$ without DIR does not achieve this. But if we replace the last clause $e \vee f \vee g$ with $e \vee f$ or with $f \vee g$, then a is derivable using $\mathcal{STN}$ without DIR.

Since it seems that PERFECT is the right semantics for stratified programs under a fully *exclusive* interpretation of "$\vee$", Przymusinski was forced to introduce a *disjunctive inference rule* to obtain an extension of PERFECT. In a preliminary draft, he used a different rule than in his final paper Przymusinski (1991d). In Dix (1992b) we have shown serious shortcoming of these semantics. We will illustrate these in Section 5.4.

Recently, however, Przymusinski seems not to rely on PERFECT: the *static semantics* introduced in Przymusinski (1995) does not extend PERFECT. It turns out that STATIC is very closely related with the semantics D-WFS introduced in Brass and Dix (1995a, 1995c) (see Section 5.4). We claim that any semantics of disjunctive programs under an exclusive interpretation of $\vee$ will be an $\leq_k$-extension of our semantics D-WFS (this semantics will be defined in Section 5.4, Definition 5.38, using our abstract framework).

While GDWFS, WF^3 and $\mathcal{STN}$ both extend the perfect semantics (in the sense that they coincide on the class of stratified disjunctive programs), only $\mathcal{STN}$ also extends WFS in this sense. In general, we have

$$\text{WFS}(P) \;\leq_k\; \text{GDWFS}(P) \;\leq_k\; \text{WF}^3(P)$$

where the first $\leq_k$ has only a meaning for programs P without disjunctions. Analogously, DSTABLE is both an extension of PERFECT and of STABLE.

Up to now we interpreted the disjunction "$\vee$" occurring in the heads of disjunctive programs *exclusive*. There is an alternative semantics for positive disjunctive programs, the *weak generalized* CWA: WGCWA (see Rajasekar et al. (1989b)). WGCWA is equivalent to the disjunctive database

rule DDR considered in Ross and Topor (1988) and has been considered as a more tractable (weaker) variant of GCWA (from the procedural point of view developed in Rajasekar et al. (1989b)). Indeed, for deriving negated literals $\neg A$, WGCWA is Π_1^0-complete (like CWA) and therefore "better" than GCWA (which is Π_2^0-complete). In the propositional case, WGCWA is polynomial while GCWA is Π_2^p-complete (both for the derivation of literals of the form $\neg A$). WGCWA (or DDR) interprets "$\vee$" *inclusive*. Let us clarify this point with

Example 3.17 (Inclusive versus Exclusive).

$$P_{incl/excl}: \quad \begin{array}{lll} a \vee b & & \\ c & \leftarrow & a, b \end{array}$$

Under an exclusive *interpretation,* $\neg c$ *should be derivable. Indeed, we have* $GCWA(P_{incl/excl}) = \{\neg c\}$.
Under an inclusive *interpretation however,* $\neg c$ *should not be derivable. This is the case for WGCWA:* $M_{P^*_{incl/excl}} = \{a, b, c\}$. *The set of* positive *derivable literals is in both cases the same! If we replace the first clause with* a *or* b, *then* $\neg c$ *is derivable.*

Originally, WGCWA was defined by modifying the set of derivable negated atoms. Instead of considering $HB_P \setminus \{\text{atoms in } can(T_P^{\mathcal{C}} \uparrow \omega(\emptyset)\}$ Minker and Rajasekar declared $HB_P \setminus \{\text{atoms in } T_P^{\mathcal{C}} \uparrow \omega(\emptyset)\}$ to be the set of derivable negated atoms under WGCWA. The set of derivable atoms is for both semantics the same. In Rajasekar et al. (1989b), they gave an equivalent and more intuitive definition of WGCWA. The idea is to restrict the inference of negative literals with respect to GCWA. Note that *only for the inference of* negative *literals* are the disjunctions interpreted *inclusively* rather than *exclusively*. This idea is formalized by transforming any clause $A_1(\underline{t_1}) \vee ... \vee A_n(\underline{t_n}) \leftarrow B_1(\underline{s_1}) \wedge ... \wedge B_m(\underline{s_m})$ into n clauses

$$A_1(\underline{t_1}) \leftarrow B_1(\underline{s_1}) \wedge ... \wedge B_m(\underline{s_m}), \tag{3}$$
$$A_2(\underline{t_2}) \leftarrow B_1(\underline{s_1}) \wedge ... \wedge B_m(\underline{s_m}), \tag{4}$$
$$\vdots \tag{5}$$
$$A_n(\underline{t_n}) \leftarrow B_1(\underline{s_1}) \wedge ... \wedge B_m(\underline{s_m}). \tag{6}$$

Using this transformation, any positive disjunctive program P can be transformed into a definite program P^*, which has a least Herbrand model M_{P^*}. WGCWA(P) can then be defined as the following set:

$$\text{MIN-MOD}(P) \cup \{M_{P^*}\}.$$

In Dix (1992b), we have introduced WPERFECT: a counterpart of PERFECT defined only for stratified programs but interpreting "$\vee$" inclusively.

The idea underlying WPERFECT is the same as the one used in the definition of WGCWA: we restrict the derivation of negative literals by first transforming P into a stratified program P^* and then adding $M_{P^*}^{supp}$ as an additional model:

$$\text{WPERFECT}(P) = \{\mathcal{M} : \mathcal{M} \text{ is a perfect model of } P \} \cup \{M_{P^*}^{supp}\}.$$

For the associated sceptical consequence-relation (consisting of all derivable literals) we can also write

$$\text{WPERFECT}(P) = \text{PERFECT}(P) \cap M_{P^*}^{supp}.$$

While both $\mathcal{STN}$ and STATIC are built on GCWA, we can also define $\mathcal{WSTN}$ and WSTATIC by using WGCWA instead: in this way we get versions of these semantics under an inclusive view. Without going into details, we just note that

$$\text{WSTATIC} \leq_k \mathcal{WSTN} \quad \text{and} \quad \text{STATIC} \leq_k \mathcal{STN}.$$

In addition WPERFECT is stronger than $\mathcal{WSTN}|_{strat}$, the restriction of $\mathcal{WSTN}$ to stratified programs: we claim that WPERFECT is the canonical semantics for stratified programs under the inclusive view. Recently this claim has been supported by work of Sakama and Inoue (1993) where they constructed with different methods the same semantics.

Example 3.18 ($\mathcal{WSTN} \leq$ WPERFECT).

$$\begin{array}{lll} P_{WPER}: & a \vee b & \\ & x \leftarrow & \neg a \\ & x \leftarrow & \neg b \\ & y \leftarrow & a, b \\ & z \leftarrow & x, \neg y \\ & z \leftarrow & y, \neg x \end{array}$$

WPERFECT derives z from P_{WPER}: PERFECT derives x and $\neg y$ (hence also z) and $M_{P^*_{WPER}}^{supp}$ is a model of $a, b, \neg x$, therefore also from z. Neither y nor $\neg y$ is derivable from WPERFECT. But adding y to P_{WPER}, z is no longer derivable. $\mathcal{WSTN}$ does not derive z.

Given a semantics SEM_{strat} for stratified programs (disjunctive or not, interpreting "$\vee$" exclusive or inclusive), we describe in Dix and Müller (1994a) how to construct a cumulative extension SEM to the whole class of arbitrary programs. In the case of normal programs (choosing SEM_{strat} to be Apt, Blair and Walker's supported semantics) we get WFS. But the construction generalizes to the class of disjunctive programs. Choosing SEM_{strat} to be PERFECT, we get a cumulative extension DWFS, which is weaker than $\mathcal{STN}$:

Example 3.19 (STATIC $\leq$ DWFS $\leq \mathcal{STN}$).

$$\begin{array}{lllll} P_{DWFS}: & a & \leftarrow & \neg d \\ & c & \leftarrow & \neg d \\ & c \vee d & \leftarrow & a \end{array}$$

The final version of $\mathcal{STN}$ derives $\neg d$ from P_{DWFS}. DWFS and STATIC are more cautious: $STATIC(P_{DWFS}) = DWFS(P_{DWFS}) = \emptyset$.

For an inclusive meaning of "$\vee$" we can choose SEM_{strat} to be WPERFECT and get an extension WDWFS, with $\mathcal{WSTN} \leq_k WDWFS$. In Brass and Dix (1995c) we also defined WD-WFS, the inclusive counterpart of D-WFS. Again we claim that any semantics of disjunctive programs under an inclusive interpretation of $\vee$ will be an $\leq_k$-extension of WD-WFS (this semantics will also be defined in Section 5.4, Definition 5.38, using our abstract framework).

There is also a semantics for general disjunctive programs defined in Ross (1989) which interprets "$\vee$" *inclusive*: the strong WFS. The reader is referred to the discussion in Dix (1992b), Dix and Müller (1994a), Brass and Dix (1995c, 1995a).

An implementation of different variants of Przymusinski's stationary semantics using *Integer-Linear-Programming methods* is presented in Müller (1992a). This approach was introduced by Bell, Nerode, Ng and Subrahmanian Bell et al. (1991, 1993), Nerode et al. (1991). The reader is also referred to Müller (1992b), Müller and Dix (1993), Dix and Müller (1992) for more on implementation-theoretic issues. An approach to compute STATIC is contained in Costantini and Lanzarone (1995b). Fernandez and Minker present in Fernández and Minker (1995) an interesting idea to compute perfect models for stratified databases.

Figure 2 illustrates important classes of programs and various semantics defined for them. Obviously, a semantics defined for a certain class of programs is also defined for any smaller class. But note that it does not necessarily coincide on this smaller class with another semantics for this particular class. An illustration of these properties is Figure 10 (Section 5.4).

The semantics GWFS, GDWFS, WF^3 and al. are comprehensively discussed in the recent book Lobo et al. (1992).

Efficient query-evaluation is also discussed in Brass (1993), Brass and Lipeck (1993, 1992), where a bottom-up prover for prioritized circumscription is defined: this prover can be used to implement the perfect semantics for stratified disjunctive programs.

Recently, Brass and Dix discussed in Brass and Dix (1994c, 1994b)

	Complexity	
	1. ord. prog. *(with functions)*	*prop. prog.* *(no variables)*
$\mathbf{M_P}$ *(P is Horn)*	A: $\boldsymbol{\Sigma_1^0}$-compl. $\neg A$: $\boldsymbol{\Pi_1^0}$-compl.	**linear** in $\|P\|$
$\mathbf{M_P^{supp}}$ *(P is stratified)*	**arithm.**-compl. (M_P^{supp} is $\boldsymbol{\Sigma_n^0}$)	**linear** in $\|P\|$
COMP	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	**co-NP**-comp.
$\mathbf{COMP_3}$	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	**linear** in $\|P\|$
STABLE	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	**co-NP**-compl.
REG-SEM	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	**co-NP**-compl.
WFS	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	**linear** in $\#\mathrm{At} \times \|P\|$
WFS′	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	**co-NP**-compl.
$\mathbf{WFS^+}$	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	**co-NP**-compl.

Table 2: Complexity of Non-Disjunctive Semantics

the notion of *partial deduction*[13] (they called it GPPE, see Section 5.3, after Definition 5.15 and see Definition 5.35 for the disjunctive case) and its relation to semantics of logic programs. They showed how a program can be assigned a certain normalform and that the disjunctive stable semantics can be uniquely characterized by GPPE (Brass and Dix (1995b)).

3.7 Complexity.

In this section we collect some complexity results for the semantics considered so far. For the exact terminology, definitions and results we refer the interested reader to the following interesting overviews Schlipf (1990b, 1992a), Cadoli and Schaerf (1993). Further results are contained in Eiter et al. (1993), Sacca (1993), Chomicki and Subrahmanian (1990), Eiter and Gottlob (1993). While Table 2 and Table 4 are considered with semantics of normal logic programs, Table 3 and Table 5 treat the same questions for some of the disjunctive semantics. Some explanations are appropriate:

Table 2 and 3: We consider the complexity of deciding if a *given ground literal is contained in the respective sceptical semantics* (i.e. if it is true in all *intended* models).

[13] Also known as *unfolding*.

	Complexity	
	1. ord. disj. prog. *(with functions)*	*prop. disj. prog.* *(no variables)*
GCWA *(P is positive)*	A: $\boldsymbol{\Sigma_1^0}$-compl. $\neg A$: $\boldsymbol{\Pi_2^0}$-compl.	A: **co-NP**-compl. $\neg A$: $\boldsymbol{\Pi_2^P}$-compl.
WGCWA *(P is positive)*	A: $\boldsymbol{\Sigma_1^0}$-compl. $\neg A$: $\boldsymbol{\Pi_1^0}$-compl.	A: **co-NP**-compl. $\neg A$: **linear** in $\|P\|$
PERFECT *(P is stratified)*	**arithm.**-compl.	$\boldsymbol{\Pi_2^P}$-compl.
WPERFECT *(P is stratified)*	**arithm.**-compl.	$\boldsymbol{\Pi_2^P}$-compl.
DSTABLE	$\boldsymbol{\Pi_1^1}$-compl. over $\mathbb{N}$	$\boldsymbol{\Pi_2^P}$-compl.

Table 3: Complexity of Disjunctive Semantics

For the 1. column, we consider arbitrary first-order programs with function symbols. We thus get undecidability results of varying strength. Since we restrict to Herbrand models, we can assume (by standard recursive encoding techniques, like *Gödel-numberings*) that all models have universes which are subsets of the natural numbers $\mathbb{N}$. The completeness results mean that for every set of the respective complexity class there is a program that defines this set under the respective sceptical semantics. Unless indicated otherwise, there is no difference between deciding ground *atoms* or ground *negated atoms*.

For the 2. column, we consider *propositional* programs. Hence we get decidable problems of various degrees. We denote by $|P|$ the total length of the program and by #At the number of distinct proposition letters in P. See also Ben-Eliahu and Dechter (1992), Schlipf (1992a), Imielinski (1991), Marek et al. (1992), Witteveen (1991), Jonker and de Lavalette (1992) for more results on the complexity of propositional programs.

Table 4 and 5: Here we consider the *expressibility* (or expressive power) of first order programs *without* function symbols. The idea is to distinguish between EDB-relations (relations that do not appear in the head of a program) and IDB-relations (which are contained in some heads). For a given program P we can view any instance $\mathcal{D}$ of the (finite) EDB-relations as an *input* argument and then compute the (finite) IDB-relations (the *output*) under the respective sceptical semantics. So we are asking:

What are the relations expressible with logic programs under certain semantics?

	Expressibility *1. ord. prog. (no functions)*
$\mathbf{M_P}$ *(P is Horn)*	$\subset$ **IND** (thus $\subset$ **P**)
$\mathbf{M_P^{supp}}$ *(P is stratified)*	$\subset$ **IND** (thus $\subset$ **P**)
COMP	= **co-NP**
$\mathbf{COMP_3}$	= **IND** (thus $\subset$ **P**)
STABLE	= **co-NP**
REG-SEM	$= \boldsymbol{\Pi_2^P}$
WFS	= **IND** (thus $\subset$ **P**)
WFS′	= **IND** (thus $\subset$ **P**)
$\mathbf{WFS^+}$	= **IND** (thus $\subset$ **P**)

Table 4: Expressibility of Non-Disjunctive Semantics

Roughly speaking, a relation R over finite EDB's $\mathcal{D}$ (i.e. for every finite $\mathcal{D}$ is associated a relation $R^{\mathcal{D}}$ on $\mathcal{D}$) is *expressible* if there is a program P containing an IDB-symbol r s. t. for every relational database $\mathcal{D}$ and tuple $\underline{t}$ corresponding to r:

$$r(\underline{t}) \in SEM^{scept}(P + \mathcal{D}) \text{ if and only if } R(\underline{t}) \text{ holds in } \mathcal{D}.$$

This is the classical notion of *expressibility* (see Schlipf (1990b), Eiter et al. (1993)). Note that expressibility defined that way does not take into account the derived *negative literals*. E.g. in the case of GCWA (and WGCWA) expressibility only tells us about positive derivable atoms – therefore the expressive power of GCWA and WGCWA is identical. The same applies to PERFECT and WPERFECT.

We are in particular interested to express all relations of some complexity class (note that the complexity is always with respect to the finite relational database as input, the program is fixed). It is well-known that the relations *inductively definable* over $\mathcal{D}$, we denote them by $\mathbf{IND}(\mathcal{D})$

(or simply **IND** to avoid the explicit occurrence of the EDB), is a strict subclass of the relations that are *polynomial* over $\mathcal{D}$ (see Barwise (1975), Moschovakis (1974), Gurevich (1988)).

	Expressibility *1. ord. disj. prog. (no functions)*
GCWA *(P is positive)*	$\subset \boldsymbol{\Pi}_2^{\mathrm{P}}$
WGCWA *(P is positive)*	$\subset \boldsymbol{\Pi}_2^{\mathrm{P}}$
PERFECT *(P is stratified)*	$= \boldsymbol{\Pi}_2^{\mathrm{P}}$
WPERFECT *(P is stratified)*	$= \boldsymbol{\Pi}_2^{\mathrm{P}}$
DSTABLE	$= \boldsymbol{\Pi}_2^{\mathrm{P}}$

Table 5: Expressibility of Disjunctive Semantics

3.8 Summary.

- NMR-intuition differs from LP-intuition: infinite branches should be regarded as *failed*.
- For *stratified* programs, there is an agreement about the *right* semantics: M_P^{supp}.
- For *disjunctive* programs, there is the possibility to interpret "$\vee$" *inclusive* or *exclusive*, which leads already for *positive disjunctive* programs to two different semantics: GCWA and WGCWA. For stratified programs we have PERFECT as the canonical semantics for fully *exclusive* "$\vee$" and WPERFECT for *inclusive* "$\vee$".
- There is still no general agreement about the right semantics for the whole class of all disjunctive programs. However, it seems that D-WFS (resp. WD-WFS) are the weakest candidates. They coincide with the flat part of Przymusinski's STATIC semantics (resp. an appropriate variant based on WGCWA). These semantics are weaker than PERFECT (resp. WPERFECT).

4. Relations between NMR and LP

The close relationship of all these semantics to DL, AEL, and CIRC was proved by translating normal programs into these logics and showing that the semantics of a program corresponds to the meaning of the translated program in the respective logic. The first such approach is due to Przymusinski, who showed that the stratified semantics is in this sense equivalent to DL, AEL and CIRC. This result was extended in Marek and Truszczyński (1989b) to the stable semantics; they considered several translations and proved that there are one-to-one correspondences between the stable models of a program and the default extensions of the translated program (which is a default-theory). The recent Przymusinska and Przymusinski (1993) contains the converse translation: nonmonotonic theories are mapped into logic programs.

We refer the interested reader to the recent textbook Marek and Truszczyński (1993) as an comprehensive study of the most important nonmonotonic logics.

4.1 CWA versus Negation-as-Finite-Failure.

For programs P with negation, neither the consistency of CWA(P), nor of $comp(P)$ implies the other. We have, for example CWA($\{p \leftarrow \neg p\}$) = $Th(\{p\})$ but

$$comp(\{p \leftarrow \neg p\}) = Th(\{p \leftrightarrow \neg p\}) = Fml$$

and CWA($\{q \leftarrow \neg p\}$)=CWA($\{q \vee p\}$) = Fml but

$$comp(\{q \leftarrow \neg p\}) = Th(\{q \leftrightarrow \neg p\}).$$

For definite programs, however, Shepherdson proved that the union of CWA and *comp* is consistent:

If P is definite, then $comp(P) \cup \mathrm{CWA}(P)$ is consistent.

Finally, we mention Gelfond et al. (1990), where the relationship between CWA, minimal models and minimal Herbrand-models is discussed.

4.2 Circumscription versus M_P^{supp}.

The stratification $P = P_1 \cup \cdots \cup P_n$ of a program P induces an ordering $Pred_1, \ldots, Pred_n$ of the predicates ($Pred_i$ are sets of predicate-symbols) in an obvious way. Note, that this ordering is in general not unique. However, Przymusinski proved Przymusinski (1990b, 1989b),

M_P^{supp} is the only Herbrand-model of $CIRC(P; Pred_1 > \ldots > Pred_n)$

Here, $CIRC(P; Pred_1 > \ldots > Pred_n)$ stands for prioritized circumscription: predicates belonging to $Pred_i$ have a higher priority to be minimized

than predicates in $Pred_j$, for $i > j$. The same result is true for stratified disjunctive programs under the *perfect* semantics (see Przymusinski (1989b)). A similar characterization holds for pointwise circumscription (see Lifschitz (1988)).

4.3 DL and AEL versus STABLE.
Bidoit/Froidevaux as well as Marek/Truszczyński considered translations from logic programs into default logic and proved that stable models of a program P correspond to Reiter's extensions of the translation (which is a default theory). Let us consider the following natural mapping:

$$tr: \ A \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_r \ \longmapsto \begin{cases} B_1 \wedge \ldots \wedge B_k \Rightarrow A, & \text{if } r = 0, \\ \frac{b_1 \wedge \ldots \wedge b_k : \neg c_1, \ldots, \neg c_r}{a}, & \text{otherwise.} \end{cases}$$

Marek and Truszczyński proved (see Marek and Truszczyński (1993) for an excellent exposition of results of this kind)

Theorem 4.1 (Marek/Truszczyński).
Let $M \subseteq B_P$ and $tr(P) = (W, \mathcal{D})$.
M is a stable model of P **iff** *M is a maximal set of atoms, s. t. $Th(W \cup M)$ is an extension of $(W,\mathcal{D})$.*

Marek/Truszczyński proved similar results for other translations, e.g. for

$$tr'(A \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_r) := \frac{\emptyset : \neg c_1, \ldots, \neg c_r}{b_1 \wedge \ldots \wedge b_k \Rightarrow a}.$$

or for $tr''(A \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_r) := \frac{b_1 \wedge \ldots \wedge b_k : \neg c_1, \ldots, \neg c_r}{a}$.
Their result for tr' is:

M is stable **iff** *$Th(P^M)$ is an extension of $(W, tr'(P))$.*

Due to the close relationship between Default and Autoepistemic Logic, there are similar correspondences between stable models of a program and expansions of a suitably associated autoepistemic theory. Gelfond considered the following translation tr_{Gel}: the clause "rule" defined by rule $= A \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_r$, is translated to

$$tr_{Gel}(\text{rule}) := B_1 \wedge \ldots \wedge B_k \wedge \neg L(C_1) \wedge \ldots \wedge \neg L(C_r) \Rightarrow A$$

Using Marek/Truszczyński's result concerning tr' and the equivalence between extensions for prerequisite-free default theories and expansions of autoepistemic theories (see Marek and Truszczyński (1989a) and also Konolige (1988)), we immediately get[14]:

[14] Weak extensions correspond to ordinary expansions, but for prerequiste-free defaults (which is the case due to our translation tr') weak extensions coincide with extensions.

Theorem 4.2.
Let $M \subseteq B_P$ and $tr_{Gel}(P)$ be Gelfond's translation from P into AEL.
M is a stable model of P **iff** *$E(Th(P^M))$ is an expansion of $tr_{Gel}(P)$.*

Here, for a set S of formulae not involving the operator L, $E(S)$ is the unique stable (in the sense of AEL) theory T, such that $T \cap \mathcal{L} = \mathrm{Th}(S)$. $E(S)$ may be constructively described (Marek and Truszczyński (1989a)).

There are still other translations from logic programs into autoepistemic logic. In Marek and Subrahmanian (1989), a more complicated, nonmodular translation $ET(P)$ is given. Using $ET(P)$, they show a correspondence between expansions of $ET(P)$ and supported Herbrand models of P.

Finally, there is the translation of Truszczyński (1991): the clause rule $= A \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_r$ is translated to

$$tr_{Tr}(\text{rule}) = LB_1 \wedge \ldots \wedge LB_k \wedge L(\neg LC_1) \wedge \ldots \wedge L(\neg LC_r) \Rightarrow A$$

that also gives a correspondence between stable models of P and expansions of its translation. This translation, however, also works for extended programs: programs containing classical negation in the style of Gelfond and Lifschitz (1990). In addition, this translation is not sensitive with respect to the underlying modal theory.

See also Casanova et al. (1991) for an attempt to explain SLDNF within default logic.

Recently, Barback and Lobo (1995) relate the problem of determining if a given formula is true in *all* (sceptical entailment) or in *some* (credulous entailment) extensions of a default theory to logic programming. They define a sound and complete proof-procedure that is closely related to SLDNF and abduction-like procedures for logic programs.

4.4 CIRC and $\mathbf{AEL}_{cl}$ versus WFS.

In Przymusinski (1989a) it is proved that every program induces a *dynamic stratification* $(P_\alpha)_{\alpha \leq \lambda}$ on the Herbrand base. Using the straightforward extension of prioritized circumscription to an infinite set of priorities, he showed that

- $CIRC(P; P_1 > \ldots > Pred_\alpha > Pred_{\alpha+1} > \ldots)$ has WFS(P) as its only 3-valued Herbrand-model.

He also considered the Gelfond translation from logic programs into autoepistemic logic and showed, that via this translation, WFS(P) corresponds exactly to $tr_{Gel}(P)$ in AEL_{cl}, a new version of autoepistemic logic (see Przymusinski (1991a)).

Lifschitz (1992) considered still another version of autoepistemic logic and its relation to negation-as-failure.

In Gelfond and Lifschitz (1989) translations from circumscriptive theories into logic programs were considered: the conditions under which such a translation works well are, however, very strong (see also Gelfond et al. (1989)).

4.5 Summary.

- Stratified programs induce a natural priority relation that can be used to define the canonical model using prioritized circumscription.
- The stable semantics is very strongly related to Default Logic and also to different versions of Autoepistemic Logic.
- Przymusinski's new version of a AEL is strongly related to the well-founded semantics and also gives a possibility to define a semantics for *general disjunctive* programs: $\mathcal{STN}$ and STATIC.

5. Classifying and Characterizing Semantics

Our aim in this section is to look from a more structural point of view at the various semantics for logic programs just presented. These semantics induce nonmonotonic entailment relations " $\mid\!\sim$ "; it seems reasonable to ask for the properties they satisfy and try to distinguish between these semantics by looking at the requirements they meet. The interested reader may consult Dix (1991b, 1995a, 1995b), Dix and Müller (1994a) for a more detailed exposition containing full proofs. Note that we do not restrict to propositional programs. But we can assume w.l.o.g. that all our programs are fully instantiated, i.e. infinite propositional programs.

In Section 5.1, we show how all semantics induce in a uniform way a sceptical consequence relation. In Section 5.2 we list some of the by now well-known abstract properties of general nonmonotonic inference relations $\mid\!\sim$ and adapt them to the case of logic programs. Section 5.3 presents a different sort of abstract properties, especially tailored for logic programs and their underlying intuitions. We argue that any semantics should be checked against these principles. We also show that our list is in a certain sense complete (Theorems 5.26 and 5.27).

Finally we end with Section 5.4 where we sketch how to extend our methods to disjunctive programs.

5.1 Associating a Sceptical Consequence Relation.

We recall that all semantics considered so far can be defined as a certain set $MOD_3(P)$ of three-valued models of a program P. More precisely

Definition 5.1 (SEM).
A semantics SEM is a mapping from the class of all programs into the powerset of the set of all 3-valued Herbrand structures. SEM assigns to every program P a set of 3-valued Herbrand models of P:

$$SEM_P \subseteq MOD_{3-val}^{Herb_{\mathcal{L}}P}(P).$$

We will from now on use the notation $P \cup U$, where U is a set of *atoms* (no nontrivial program clauses are contained in U). P still may contain clauses with empty bodies. We also use $SEM_P(U)$ instead of $SEM_{P \cup U}$ or simply SEM(P) if $U = \emptyset$. This definition already indicates a fundamental difference to the general " $\sim$ "-framework in non-monotonic reasoning: our $\sim_P$ is not defined between *arbitrary* program clauses.

Our last definition gives rise to the following sceptical consequence relation:

Definition 5.2 (Sceptical entailment relation $\sim_P$).
Let P be a program and U a set of atoms. Any semantics SEM induces a sceptical entailment relation SEM^{scept} as follows:

$$SEM_P^{scept}(U) := \bigcap_{\mathcal{M} \in \mathrm{SEM}_P(U)} \{L : L \text{ is a pos. or neg. literal with: } \mathcal{M} \models L\}$$

Comparing with the " $\sim$ " framework of Kraus, Lehmann and Magidor, we can equivalently define a " $\sim$ "-relation between sets of atoms *U (positive literals) on the left hand side, and sets of arbitrary* literals *X on the right hand side:*

- $u_1 \wedge \ldots \wedge u_n \ \sim \ x_1 \wedge \ldots \wedge x_m$ **:iff** $\{x_1, \ldots x_m\} \subseteq SEM_P^{scept}(\{u_1, \ldots u_n\})$

5.2 Strong Principles.
The following structural properties for an entailment relation " $\sim$ " *between single formulae* were considered by Kraus, Lehmann and Magidor:

Cautious Mon.:	$\alpha \sim \beta$	*and*	$\alpha \sim \gamma$	*imply*	$\alpha \wedge \beta \sim \gamma$.
Cut:	$\alpha \sim \beta$	*and*	$\alpha \wedge \beta \sim \gamma$	*imply*	$\alpha \sim \gamma$.
Rationality:	*not* $\alpha \sim \neg\beta$	*and*	$\alpha \sim \gamma$	*imply*	$\alpha \wedge \beta \sim \gamma$.
Disjunct. Rat.:	$\alpha \vee \beta \sim \gamma$	*implies*	$\alpha \sim \gamma$	*or*	$\beta \sim \gamma$.
Or:	$\alpha \sim \gamma$	*and*	$\beta \sim \gamma$	*imply*	$\alpha \vee \beta \sim \gamma$.

The property *Cut* is uncontroversial: it is satisfied by any known non-monotonic system. *Cautious Monotony* combined with *Cut* is the following condition, called *cumulativity*:

$$\text{If } \alpha \sim \beta \text{ then: } \ \alpha \sim \gamma \ \text{ iff } \ (\alpha \wedge \beta) \sim \gamma.$$

It states that whenever a fomula β has been derived from α, then β can be used as a "lemma" and does not affect the set of formulae derivable from α alone. If this condition is not valid, intermediate lemmas are of no use: this indicates that reasoning in non-cumulative systems may be more expensive.

Under some weak conditions (that are always satisfied for the relations $\sim$ considered in this paper) *Rationality* implies both *Cautious Monotony* and *Disjunctive Rationality*. Since *Cut* is always satisfied, we will in the sequel use the terms *cautious* and *cumulative* interchangeably.

The property *Or* will turn out useful to distinguish between an *exclusive* and an *inclusive* meaning of "$\vee$" in disjunctive programs. In particular its converse

$$(\alpha \vee \beta) \sim \gamma \text{ implies } \beta \sim \gamma \text{ and } \alpha \sim \gamma$$

will turn out to describe semantics interpreting "$\vee$" inclusive.

Another difference to the KLM-framework is that we interpret the rules as *infinitistic*: our definition of entailment is a relation between *infinite* sets of atoms and literals

$$\sim_P \subseteq 2^{\text{Atoms}} \times 2^{\text{Literals}}.$$

Cumulativity: If $U_1 \subseteq U_2 \subseteq \text{SEM}_P^{scept}(U_1)$, then

$$\text{SEM}_P^{scept}(U_1) = \text{SEM}_P^{scept}(U_2).$$

Rationality: If $U_1 \subseteq U_2$, $U_2 \cap \{A : \text{SEM}_P^{scept}(U_1) \models \neg A\} = \emptyset$, then

$$\text{SEM}_P^{scept}(U_1) \subseteq \text{SEM}_P^{scept}(U_2).$$

Remark 5.3 (Finite versus Infinite). *It turns out that for our abstract properties* Cautious Monotony, Cut *or* Rationality *there is no difference between formulating it for* infinite, finite *or even* one element *sets U_2. This is obvious by the structure of the rule: if* Cautious Monotony *(resp.* Cut*) holds for* one element *sets U_2 then it also holds for* finite *sets U_2. The step from finite to infinite sets is not so obvious but all semantics justify it. For* Rationality *even the step from* one element *sets to finite sets is not so obvious. But again there are no counterexamples.*

While both two-valued semantics COMP and STABLE are not cumulative monotone[15], their three-valued extensions COMP_3 and the wellfounded semantics WFS are. The supported semantics is also cumulative (see Dix (1991b)) and rational.

[15] See P_{comp} in Section 2.2 and P_{stable} in Section 3.3: for both programs P we have: $\emptyset \sim_P p$ and $\emptyset \sim_P a$ but "not $p \sim_P a$".

Theorem 5.4 (Prop. of $\mathbf{COMP}_3$, $\mathbf{M}_P^{supp}$, WFS, WFS′, $\mathbf{WFS}^+$). *The semantics $COMP_3$, M_P^{supp}, WFS, WFS′ and WFS^+ are cumulative and rational.*

Neither of these two properties is satisfied for Baral, Lobo and Minker's extensions GWFS, GDWFS or WF^3 of WFS.

We now give the exact definition of WFS^+. As a further motivation let us consider the following problem suggested by H. Herre. *Given a semantics SEM that does not satisfy Cut, how can we extend SEM to a semantics SEM^+ satisfying Cut?* If SEM is *rational* the answer is given by the following

Theorem 5.5 ($\mathbf{SEM}^+$). *Let SEM be a rational semantics and define an increasing sequence of sets of atoms M_i by setting $M_0 := \emptyset$, $M_{i+1} := \{a :$ a an atom with $a \in SEM(P \cup M_i)\}$. We have*

$$SEM(P \cup M_i) \leq_k SEM(P \cup M_{i+1}).$$

There exists therefore a γ such that $M_\gamma = M_{\gamma+1}$. We define

$$SEM^+(P) := SEM(P \cup M_\gamma).$$

SEM^+ is the weakest extension of SEM satisfying Cut. *In addition, SEM^+ is also rational.*

The proof is obvious: we only add those atoms that have to be added in order to satisfy *Cut*. We need *Rationality* to prove inductively the relation $\mathrm{SEM}(P \cup M_i) \leq_k \mathrm{SEM}(P \cup M_{i+1})$[16] and to reach a limit M_γ. Otherwise the sequence M_i could be oscillating.

The semantics WFS^+ can be seen as an attempt to construct an extension of WFS that satisfies the property of *Supraclassicality* (called *classical completeness* in Schlipf (1992b)):

Supraclassicality: if $P \models a$ then $a \in \mathrm{WFS}^+(P)$ (for atoms a).

Definition 5.6 ($\mathbf{WFS}^+$). *Let P be a normal program, and define*

$$SEM(P) := WFS(P) \cup \{a :\ a \text{ an atom with: } P \models a\}.$$

The assumptions of Theorem 5.5 are satisfied: we get a cumulative and rational semantics $WFS^+ := SEM^+$.

In Dix (1995a) we show that the wellfounded-by-case semantics WFS_C due to Schlipf (1992b) and WFS^+ coincide. Indeed we can show the following more elegant and interesting characterization:

[16] The definition of M_{i+1} ensures that if $x \in M_{i+1}$ then $\neg x \notin \mathrm{SEM}(P \cup M_i)$ so that *Rationality* can be applied.

Theorem 5.7 (WFS$^+$ as $\leq_k$-weakest extension of WFS).
WFS$^+$ is the weakest extension of WFS that satisfies Cut and Supraclassicality. In addition WFS$^+$ is rational and satisfies the following property (stronger than Supraclassicality):

$$\textit{If } P \cup \textit{WFS}^+(P) \models a \textit{ then } a \in \textit{WFS}^+(P).$$

Obviously, supraclassical semantics derive p in programs containing the clause $p \leftarrow \neg p$ (classically, this clause is equivalent to p). One might think therefore that a supraclassical extension of STABLE removes sometimes the inconsistency problem. Schlipf (1992b) defined in Schlipf (1992b) to this end not only a supraclassical extension of WFS but also a supraclassical version of STABLE: STABLE$_C$ (STABLE-BY-CASE). Using our principles, we can give a more elegant definition:

Definition 5.8 (STABLE′).

$$\textit{STABLE}'(P) := \begin{cases} \textit{STABLE}(P \cup \{a : \ P \models a\}), & \textit{if stable models exist,} \\ \textit{WFS}^+(P), & \textit{otherwise.} \end{cases}$$

Lemma 5.9 (STABLE′ $\leq_k$ STABLE).
STABLE′ is a supraclassical semantics below STABLE satisfying Cut.

Why is STABLE′ weaker than STABLE? This is just because STABLE is not cumulative and some inconsistencies are removed: for the program "$p \leftarrow \neg p$" STABLE is inconsistent while STABLE′ is not. The introduction of new atoms in general *decreases* the number of stable models.

Note that while STABLE$_C$ might still be inconsistent for some programs P, our STABLE′ is always consistent (by construction). Schlipf's STABLE$_C$ is still weaker than STABLE′: we refer the reader to Dix and Müller (1994b) where we comment on this and where we defined (beside other things) a semantics STABLE$^+$ that coincides with the consistent part of STABLE$_C$.

Note that REG-SEM (the semantics based on three-valued stable models) is incomparable with STABLE$_C$: REG-SEM derives x from the program "$a \leftarrow \neg b$, $b \leftarrow \neg a$, $x \leftarrow \neg a$, $x \leftarrow \neg b$" while STABLE$_C$ does not.

Pereira's O-SEM is cumulative but not rational:

Example 5.10 (O-SEM 2).

$$P_{O_SEM\,2}: \quad \begin{array}{lcl} a & \leftarrow & \neg a \\ p & \leftarrow & a \\ q & \leftarrow & \neg p \end{array}$$

O-SEM associates $\{q, \neg p\}$ to $P_{O_SEM\,2}$: a is undefined. Therefore, $\neg a$ is not derivable. But the associated model for $P_{O_SEM\,2} \cup \{a\}$ is $\{p, \neg q\}$: p is now derivable. Hence rationality does not hold.

5.3 Weak Principles.
As already noted in the beginning of this section, all the known semantics (see Figure 6) are defined by declaring some of the (three-valued) models of P as the intended models. It turns out that the supported model, the stable models (which are two-valued) and also the wellfounded model satisfy the following conditions (this was noted by L. Pereira and explicitly stated in Pereira et al. (1992)): (let $\mathcal{A}$ be such a model)

C1: If $a \leftarrow rhs \in P$ and $\mathcal{A} \models rhs$, then $\mathcal{A} \models a$.

C2: If $\mathcal{A} \models a$ then there is $a \leftarrow rhs \in P$ with $\mathcal{A} \models rhs$.

C3: If all rule bodies for a are false in $\mathcal{A}$ then $\mathcal{A} \models \neg a$.

C4: If $\mathcal{A} \models \neg a$ then all rule bodies for a are false in $\mathcal{A}$.

Note that we are always interested in the associated sceptical semantics SEM^{scept} which is given by the $\sqcap_k$-intersection of all intended models. We now want to list reasonable properties of SEM^{scept} in order to describe particular semantics as uniquely determined by these properties. Any sceptical semantics SEM^{scept} such that SEM is based on models satisfying C1 - C4, also satisfies C1 and C3 but not necessarily C2 and C4. We only consider C1 as a reasonable property that should be satisfied. Depending on the application, C3 might also be useful. As simple examples show, neither C2 nor C4 are reasonable. WFS^+ (and also GWFS) do not satisfy C3 because $\text{WFS}^+(\{a \leftarrow \neg a\}) = \{a\}$. WFS′ however satisfies C3.

The failure of *Cautious Monotony* for GWFS can be illustrated with the program P_{GWFS} of Example 3.11: $\text{GWFS}(P_{GWFS})$ entails $\neg c$, because $\text{MIN-MOD}(P_{GWFS})$ consists of $\{ \{p, a\}, \{b\} \}$ and thus also $\neg b$, p and a. But $\text{MIN-MOD}(P_{GWFS} \cup \{p\}) = \{ \{p, a\}, \{p, c, b\} \}$, thus $\text{GWFS}(P_{GWFS} \cup \{p\})$ does not entail $\neg c$.

Note that c occurs only positively in P_{GWFS} and all occurrences have been replaced by the rule defining it, but yet the semantics of P_{GWFS} and $P_{GWFS\,c}$ differ: P_{GWFS} derives p and a while $P_{GWFS\,c}$ does not.

Our principle of partial evaluation, PPE, states that this should not happen: any semantics should assign the same meaning to a program P and a partial evaluation of it. The formal definition is as follows:

Definition 5.11 (PPE, Weak Version).
Let P be an instantiated program and let the atom c occur only positively in P. Let $c \leftarrow rhs_1, \ \ldots \ , \ c \leftarrow rhs_n$ be all the rules of P with c in their heads. Assume further that $rhs_1, \ldots, rhs_n$ do not contain c.

We denote by P_c the program obtained from P by deleting all rules with c in their heads and replacing each rule "head $\leftarrow c$, body" containing c by the rules:

$$\begin{array}{lcl} head & \leftarrow & rhs_1, body \\ & \vdots & \\ head & \leftarrow & rhs_n, body \end{array}$$

The weak PPE *states that* $SEM(P_c) = SEM(P) \setminus \{c, \neg c\}$.

PPE can significantly decrease the complexity of computing a semantics, since it allows us to *reduce* the program by making the underlying Herbrand base smaller: in general, less assignments of elements of B_P have to be taken into account. We argue that PPE should be satisfied by any reasonable semantics.

There are still other interesting properties that should be satisfied. To state these conditions, we need the notion of the Dependency-Graph (Definition 2.7) and the two definitions

- $dependencies_of(X) := \{A : X \text{ depends on } A\}$, and
- $rel_rul(P, X)$ is the set of *relevant rules* of P with respect to X, i.e. the set of rules that contain an $A \in dependencies_of(X)$ in their head.

Given any semantics SEM and a program P, it is perfectly reasonable that the truthvalue of a literal L, with respect to SEM(P), only depends on the subprogram formed from the *relevant rules* of P with respect to L.[17] This idea is formalized by:

Definition 5.12 (Relevance).
The principle of Relevance states: $SEM(P)(L) = SEM(rel_rul(P, L))(L)$.

Note that the set of relevant rules of a program P with respect to a literal L contains all rules, that could ever contribute to L's derivation (or to its nonderivability). In general, L depends on a large set of atoms: $dependencies_of(L) := \{A : L \text{ depends on } A\}$. But rules that do not contain these atoms in their heads, will never contribute to their derivation or nonderivation. Therefore, these rules should not affect the meaning of L in P. STABLE does not satisfy this principle. This is due to the nonexistence of stable models by adding a clause "$c \leftarrow \neg c$" to a program: see Example 3.8.

We consider this a very undesirable feature of STABLE. This can be repaired be modifying STABLE and defining a version STABLE^{rel} that focusses on the relevant rules of P with respect to a literal L:

$$\text{STABLE}^{rel}(P)(L) := \begin{cases} \text{STABLE}(rel_rul(P, L))(L), & \text{if } \exists \text{ stable models,} \\ \text{WFS}(rel_rul(P, L))(L), & \text{otherwise.} \end{cases}$$

[17] $dependencies_of(\neg X) := dependencies_of(X)$, $rel_rul(P, \neg X) := rel_rul(P, X)$.

Although this definition seems to be very natural and satisfies *Relevance*, it has another serious shortcoming (the Cut C_1 is not satisfied, see Dix (1992a)) and needs therefore further refinement (see Dix (1992d, 1995a), Dix and Müller (1994b)).

There is another natural property satisfied for all semantics in the literature. Suppose we add a set N of atoms to a program P. Independently of whether some of the atoms or their negations are derivable from P alone or not, the semantics of $P \cup N$ should be the same as the semantics of "P reduced by N", augmented by N. This idea can be generalized in two directions:

1. we can allow N to contain negative literals. We define P to be equal to $P \cup \{\neg A\}$ just in case A does not appear in any head of P. Thus, $P \cup \{\neg A\}$ is just an abbreviation for the program obtained from P by removing all clauses with A in their heads.

2. we can allow to replace any positive occurrence of an atom A by the associated defining rules.

Before giving the precise notions, we need the definition of "P reduced by M", where M is a set of literals.

Definition 5.13 (P reduced by M).
Let P be an instantiated program and M a set of literals. We define "P reduced by M" as the following program

$$P^M := \{rule^M : \ rule \in P\},$$

where $(A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m)^M$ is defined to be

$$\begin{cases} \mathbf{t}, & \text{if } \exists j : C_j \in M \text{ or } \neg B_j \in M, \\ \mathbf{t}, & \text{if } A \in M \text{ or } \neg A \in M, \\ rule', & \text{otherwise.} \end{cases}$$

Here, $rule'$ stands for the clause "$A \leftarrow B'_1, \ldots, B'_{n'}, \neg C'_1, \ldots, \neg C'_{m'}$", where the set $\{B'_i : i \in I'\}$ (resp. $\{\neg C'_i : i \in I'\}$) is just an enumeration of the set $\{B_i : B_i \in I\} \setminus M$ (resp. $\{\neg C_i : \neg C_i \in I\} \setminus M$).

We define the associated language $\mathcal{L}_{PM}$ for P^M as being $\mathcal{L}_P \setminus M$.

This notion of reduction resembles the Gelfond-Lifschitz transformation to define stable models but it will be used totally differently:

- Our reduced program still contains negation: the Gelfond-Lifschitz transformation is a positive program.

- While Gelfond/Lifschitz reduce with a set N of *atoms* and also implicitly assume that all atoms not in N are false, we are more cautious and reduce only those literals that are explicitly contained in M. We *remove* all atoms that are already known to be true or false: once **t** or **f** is assigned, it is fixed. We use this information to simplify the program: our reduction decreases the Herbrand base (see Müller and Dix (1993)). Literals contained in M do no longer occur in P!

Our remark about the underlying language is very important. It may happen that we reduce a program "$a \leftarrow b$" with $\{\neg b\}$, so that we obtain the empty program that gives us no information about a (a does not appear in the underlying language). In our definition we also obtain the empty program but with respect to the language containing a: this allows us to deduce $\neg a$. The formal statement for condition 1. above is

Definition 5.14 (Reduction).
Let a set of literals $M \subseteq B_P \cup \neg B_P$ be given.
The principle of Reduction states that $SEM(P \cup M) = SEM(P^M) \cup M$.

It is important to take a closer look at this property to make sure that it is indeed a very weak property. It simply says that whenever we have an explicit fact "$a \leftarrow$ " then we can replace every occurrence of a by "**t**" (dually: every occurrence of $\neg a$ by "**f**"). Note our convention of using "$P \cup \{\neg x\}$": it is just an abbreviation for the program obtained from P by deleting all clauses with x in their heads and keeping the language of P ($\mathcal{L}_{P \cup \{\neg x\}} := \mathcal{L}_P$). Therefore *Reduction* assures that atoms that do not occur in the heads of a program are always assigned "**f**". This property should definitely be among the minimal requirements for semantics of logic programs: it is nothing else than a very weak form of the *negation-as-failure*-idea.

The formal statement for condition 2. above is a generalization of the (weak version of) PPE introduced above:

Definition 5.15 (PPE, Strong Version).
Let P be an instantiated program and let the atom c occur positively in P. Let $c \leftarrow rhs_1, \ \ldots \ , \ c \leftarrow rhs_n$ be all the rules of P with c in their heads.

Any program clause of the form "$head \leftarrow c, body$" can be replaced by the rules

$$\begin{array}{lcl} head & \leftarrow & rhs_1, body \\ & \vdots & \\ head & \leftarrow & rhs_n, body \end{array}$$

Note that the rules $c \leftarrow rhs_1, \ \ldots \ c \leftarrow rhs_n$ are not removed (in contrast to the weak version of PPE). We call the program obtained in this way P'.

The principle of partial evaluation is: $SEM(P') = SEM(P)$.

Obviously, the weak version of PPE is deducible:

Lemma 5.16 (Weak PPE is Deducible).
Relevance *and* PPE *imply* Weak PPE.

Although it seems very intuitive to weaken the assumption that c *only occurs positively* in the strong version of PPE (let us denote this generalized version by GPPE, we give a precise definition including the disjunctive case in Definition 5.35), this is at least at odds with the semantics WFS_C (or, equivalently, WFS^+):

Example 5.17 (Generalized PPE fails for $\mathbf{WFS}_C$).

$$
\begin{array}{lllll}
P_{splitting}: & a & \leftarrow & \neg b & \\
& b & \leftarrow & \neg a & \\
& p & \leftarrow & a & \\
& p & \leftarrow & b &
\end{array}
\qquad
\begin{array}{lllll}
P'_{splitting}: & a & \leftarrow & \neg b & \\
& b & \leftarrow & \neg a & \\
& p & \leftarrow & \neg b & \\
& p & \leftarrow & \neg a &
\end{array}
$$

The second program results from the first by just evaluating a (by $\neg b$) and evaluating b (by $\neg a$). While WFS_C derives p from $P_{splitting}$, it does not derive p from $P'_{splitting}$. Note that if we make GPPE to be true (i.e. p is declared to be derivable from $P'_{splitting}$), then Rationality *fails: adding both a and b makes not only p nonderivable, but it makes even $\neg p$ derivable!*

As already mentioned at the end of Section 5.2 REG-SEM derives p from $P'_{splitting}$. Indeed, GPPE holds for REG-SEM (see Dix and Müller (1994b, 1994c) and Figure 9). John Schlipf noted that GPPE might be a principle formalizing some sort of *abductive reasoning* (see Kakas et al. (1993)). See also Brass and Dix (1995b) where various characterizations of the stable semantics (also for disjunctive logic programs) using GPPE are given.

The next principle, *Modularity*, has some similarities with PPE. It enables us to compute a semantics by modularizing it into certain "subprograms" (formed of the relevant rules). The semantics of these modules can be computed first and the semantics of the whole program can be determined by reducing this program with literals that were already determined.

Definition 5.18 (Weak Modularity).
Let P be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \{A\}$ and $P_2 = rel_rul(P, A)$. Then: $SEM(P) = SEM(P_1{}^{SEM(P_2)} \cup P_2)$.

We showed in Dix (1992a) that the stationary approach of Przymusinski does not satisfy Weak Modularity. We do not know of any semantics for *normal* programs that fails to satisfy Weak Modularity. It is useful to strengthen this condition by weakening the assumptions "$B_{P_1} \cap B_{P_2} = \{A\}$ and $P_2 = rel_rul(P, A)$":

Definition 5.19 (Modularity).
Let P be instantiated, $P = P_1 \cup P_2$ and for $A \in B_{P_2}$: $rel_rul(P, A) \subseteq P_2$.
The principle of Modularity is: $SEM(P) = SEM({P_1}^{SEM(P_2)} \cup P_2)$.

Modularity is a restricted form of *Cautious Monotony*:

Lemma 5.20 (Cumulativity implies Modularity).
From Reduction, Relevance *and* Cumulativity *follows* Modularity.

Note that in the last definition it is **not** possible to replace $SEM(P) = SEM({P_1}^{SEM(P_2)} \cup P_2)$ by

$$SEM(P) = SEM({P_1}^{SEM(P_2)}) \cup SEM(P_2)$$

or by

$$SEM(P) = SEM({P_1}^{SEM(P_2)} \cup SEM(P_2)).$$

This would exclude reasoning-by-cases phenomena: let $P_1 = \{x \leftarrow a,\ x \leftarrow b\}$ and $P_2 = \{a \leftarrow \neg b,\ b \leftarrow \neg a\}$. The modified definition would not allow us to deduce x. But for our first version of Modularity, Weak Modularity, this modification makes sense and follows from our principles:

Lemma 5.21 (Deducible Variant of Modularity).
Let P be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \{A\}$ and $P_2 = rel_rul(P, A)$.
Using Modularity, Relevance *and* Reduction *we have:*

- *if $A \in SEM(P_2)$ then $SEM(P) = SEM(P_1 \cup \{A\}) \cup SEM(P_2)$,*
- *if $\neg A \in SEM(P_2)$ then $SEM(P) = SEM(P_1 \cup \{\neg A\}) \cup SEM(P_2)$.*

We still need a condition to ensure that *equivalent* programs are assigned the same semantics. But this notion of equivalence depends on the underlying s+emantics: we therefore have to model a very weak form of equivalence, true for all possible semantics. Consider two programs P and Q. If Q is just a renaming of the constants in P, then P and Q should obviously be equivalent, i.e. the renaming of $SEM(Q)$ should equal $SEM(P)$.

The next condition, *Isomorphy*, formalizes the minimal intuition:

Definition 5.22 (Isomorphy).
Let $P \cong Q$, i.e. there is an isomorphism $I : B_{\mathcal{L}_P} \cong B_{\mathcal{L}_Q}$ s.t. $I(P) = Q$.
The principle of Isomorphy states that $I(SEM(P)) = SEM(Q)$.

Together with the property *Relevance* and our PPE-principle, *Isomorphy* also applies to more complex programs that need not be isomorphic but consist of isomorphic subprograms and thus can also be proved to be equivalent.

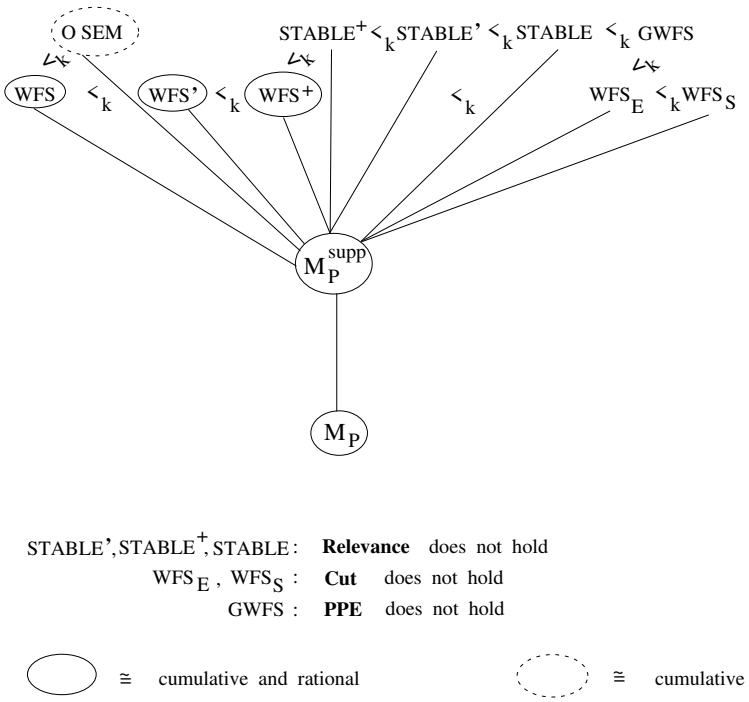

Figure 7: NMR-Semantics for Normal Programs

But there are still other *intuitively correct* equivalences between different programs possible that can not be deduced using all our principles. Let P be a program. If we add $p \leftarrow p$ we certainly get an equivalent program (independently of the truth, falsity or the undefinedness of p). But it is not clear how we can establish the equivalence $SEM(P) = SEM(P')$ using our principles. We therefore introduce our last two principles, which state that certain transformations (to be defined below) do not change a semantics.

The first principle is neccessary for infinite programs consisting of the fact $P(c)$ and the rule $P(x) \leftarrow P(s(x))$. For such programs we need the explicit requirement that for definite programs any semantics should coincide with the least Herbrand model. This property is formalized by the following definition:

Definition 5.23 (M_P-extension).
If P is a definite program, then SEM(P)=$\{l :$ l a literal with $M_P \models l\}$.

Our next principle is motivated by the program

$$\begin{array}{lcl} a & \leftarrow & b, \neg a \\ b & \leftarrow & a, \neg b \end{array}$$

where we want obviously that both $\neg a$ and $\neg b$ are derivable in order to get at least the well-founded semantics. The following principle models the idea that negative literals true in the positive transform P^+ as well as positive literals true in the negative transform P^- of a program should be derivable:

Definition 5.24 (Transformation).
Let P be a program and P^+ (resp. P^-) be the program obtained from P by cancelling all negative literals *(resp.* cancelling all clauses containing a negative literal*). A semantics SEM satisfies the principle of Transformation if the following holds:*

- $\neg x \in SEM(P^+) \implies SEM(P \cup \{\neg x\}) \subseteq SEM(P)$,
- $x \in SEM(P^-) \implies x \in SEM(P)$.

Note that in the first condition, we not only require that $\neg x \in \mathrm{SEM}(P)$ but that a special instance of the *extended Cut* can be applied.

Together with the principles introduced before, *Transformation* will be powerful enough to simulate WFS.

We are now in a position to define what a well-behaved semantics is:

Definition 5.25 (Well-behaved Semantics).
A well-behaved semantics SEM is a mapping

$$\{P : P \text{ a program}\} \longrightarrow 2^{Lit}$$

such that the following conditions are satisfied: Relevance, Reduction, Modularity, Isomorphy, M_P-extension, Transformation and Cut.

It is easy to see that *Cut* together with *Reduction* and *PPE* ensures that C_1 is satisfied for $SEM(P)$. For *finite* propositional programs, the rules *M_P-extension* and *Transformation* can be simulated by *PPE*, *Reduction* and *Equivalence*, a very weak rule defined in Dix (1995b) that allows to remove tautologies (it is essentially Definition 5.34 applied to normal programs).

Further work is needed to derive other properties that are implied by our conditions. Our idea is that fixing a semantics on a *reasonable* class of programs (for example axiomatically requiring $SEM(\{a \leftarrow \neg a) = \emptyset$, $SEM(\{a \leftarrow \neg b, b \leftarrow \neg a\}) = \emptyset$ and further axioms) already determines these semantics using our conditions. It is, however, possible that we still need more conditions (for different axiomatisations).

Using the principles we have introduced so far, we are able to prove (using only *Reduction*, *Modularity*, *M_P-extension*):

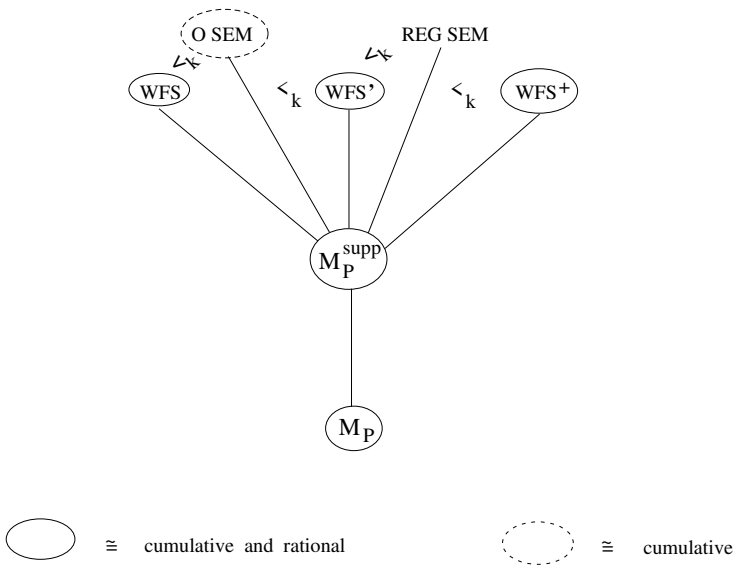

Figure 8: Well-Behaved Semantics for Normal Programs

Theorem 5.26 (Well-behaved Semantics Extend M_P^{supp}).
Any well-behaved semantics coincides for stratified programs with M_P^{supp}.

For the next theorem, we need *M_P-extension*, *Transformation*, *Reduction* and *Cut*:

Theorem 5.27 (WFS is the Weakest Extension of M_P^{supp}).
Any well-behaved semantics is a $\leq_k$-extension of WFS.

Lemma 5.28 (Well-behavedness of Some Semantics).
The semantics WFS, WFS′, WFS$^+$, O-SEM and REG-SEM are well-behaved.

Figure 8 illustrates the relationship between the well-behaved semantics. Figure 9 illustrates approximations of the stable semantics that are consistent, relevant, and have some more properties. It shows that we can either keep the *Rationality* of WFS (and obtain WFS′, WFS$^+$) or the *GPPE* (and obtain REG-SEM or STABLE^{rel}).

We claim that our reasonable conditions together with the property of rationality exactly describe the semantics WFS, WFS$^+$ and WFS′. In particular, we have the following conjectures:

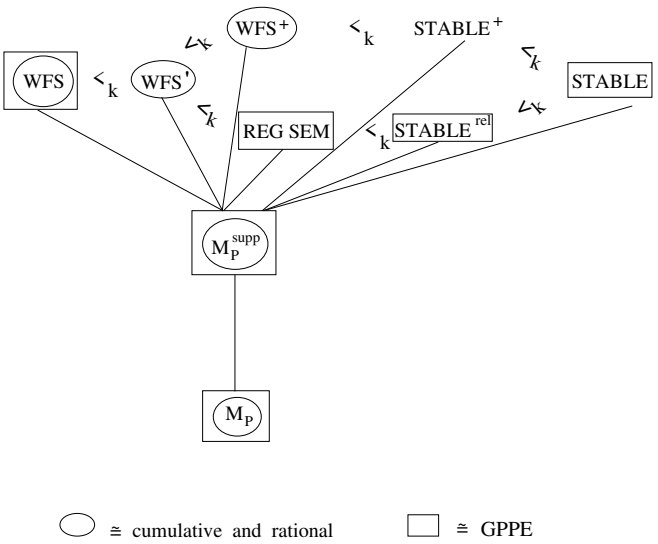

Figure 9: Approximations of STABLE

Conjecture 5.29 (Characterization of WFS, WFS$^+$ and WFS'). *There are no well-behaved and rational semantics other than WFS, WFS$^+$ and WFS$'$.*

Conjecture 5.30 (WFS$^+$). *WFS$^+$ is the only well-behaved and rational semantics satisfying Supra-classicality.*

Conjecture 5.31 (Characterization of WFS and WFS$'$). *There are no well-behaved and rational semantics satisfying C_3 other than WFS and WFS$'$.*

In Dix (1992a), we also presented a semantical framework which is related to the notion of *preferential model* in non-monotonic reasoning (see Kraus et al. (1990), Makinson (1994), Dix and Makinson (1992)). A somewhere related approach was recently proposed in Brown et al. (1992).

In Müller and Dix (1993), we show how we can use our weak principles successfully to reduce a given program and compute its semantics more efficiently. We end this section with Figures 8 and 9 to summarize the properties of the NMR-semantics considered so far.

5.4 Disjunctive Programs.
As already mentioned, the situation for disjunctive programs is more complicated than for normal programs: we have to consider extensions of both semantics GCWA and WGCWA to the whole class of programs[18].

The *exclusive* interpretation of "$\vee$" seems to be related to the *Or*-rule: if "$\alpha \mid\!\sim \gamma$" and "$\beta \mid\!\sim \gamma$" then "$(\alpha \vee \beta) \mid\!\sim \gamma$". All the semantics on the right hand side of Figure 10 satisfy this rule, while all semantics on the left hand side do not.

Example 5.32 (GCWA is not disjunctively rational).

$$\begin{array}{lll} P_{DR}: & x_1 \vee x_2 & \\ & x_1 \vee a \leftarrow & b \\ & x_2 \vee b \leftarrow & a \\ & c \leftarrow & a, b \end{array}$$

While MIN-MOD$(P_{DR} \cup \{a \vee b\}) = \{ \{a, x_2\}, \{b, x_1\} \}$*, the minimal models of* $P_{DR} \cup \{a\}$ *are* $\{a, x_2\}$ *and* $\{a, b, x_1, c\}$*. Those of* $P_{DR} \cup \{b\}$ *are* $\{b, x_1\}$ *and* $\{a, b, x_2, c\}$*. Setting* $\alpha := a$*,* $\beta := b$*, and* $\gamma := \neg c$ *we get a counterexample to* disjunctive Rationality*: we have* $\neg c \in GCWA(P_{DR} \cup \{a \vee b\})$ *but neither* $\neg c \in GCWA(P_{DR} \cup \{a\})$ *nor* $\neg c \in GCWA(P_{DR} \cup \{b\})$.

While GCWA is obviously cumulative (even full Circumscription is), the positive disjunctive program $P: \; p \vee b, \; b \leftarrow c, \; c \vee a \leftarrow p$ shows that *Rationality* can not be achieved for GCWA: $\emptyset \mid\!\sim_P \neg c$ and "not $\emptyset \mid\!\sim_P \neg p$" but "not $p \mid\!\sim_P \neg c$"! Indeed, the last example shows that GCWA is not even disjunctively rational (a weaker property).

WGCWA, however, is rational (therefore cumulative) and $\mathcal{WSTN}$, WD-WFS (semantics based on WGCWA instead on GCWA) still are cumulative and satisfy the converse of *Or*, as do all semantics on the left hand side of Figure 10 (see Dix (1992b), Dix and Müller (1994a)). The converse of *Or* can be seen as a restricted version of rationality.

It turns out that the perfect semantics PERFECT is cumulative, but, as an extension of GCWA, not rational.

Among the existing semantics for the whole class of disjunctive programs, the stationary $\mathcal{STN}$ and the static semantics STATIC of Przymusinski are very interesting. As pointed out by Przymusinski, $\mathcal{STN}$ not only extends PERFECT (as do GDWFS and WF^3) but it also extends WFS. We show, however that $\mathcal{STN}$ not only fails to be cumulative, it also does not satisfy the weaker property of *weak Modularity*.

[18] Recently a new semantics which is for some programs stronger than GCWA has been proposed in Dix et al. (1994), Schaerf (1993): the advantage of this semantics is its low complexity.

WPERFECT still is cumulative, but not rational. For normal logic programs $\mathcal{WSTN}$ and WD-WFS coincide with WFS.

In contrast to GDWFS and WF^3 Przymusinski's approaches $\mathcal{STN}$ and STATIC satisfy PPE (they even satisfy the stronger version GPPE). In a preliminary draft, he used a different rule than in the final paper Przymusinski (1991d). The following two programs show, that neither versions are cumulative and that his final version even fails to satisfy *Modularity*. Let us compare the program

$$\begin{array}{lllll} P_{STN}: & b & \leftarrow & y \\ & y \vee x & & \\ & z \vee y & & \\ & m & \leftarrow & x, z, b \\ & y & \leftarrow & \neg m \end{array}$$

with the union of the following two programs

$$\begin{array}{llllll} P'_{STN}: & b & \leftarrow & y & P_{or}: & a & \leftarrow & e, \neg g \\ & y \vee x & \leftarrow & a & & a & \leftarrow & f, \neg g \\ & z \vee y & \leftarrow & a & & a & \leftarrow & f, \neg e \\ & m & \leftarrow & x, z, b & & a & \leftarrow & g, \neg e \\ & y & \leftarrow & \neg m & & e \vee f \vee g & & \end{array}$$

P_{or} is a stratified program that derives a under PERFECT. Concerning P_{STN}, different intuitions seem possible. One can argue that $\neg m$ should be derivable, since the only way to derive m is by using the fourth clause, which means deriving b, which means deriving y which excludes deriving x or z. This is the way, P_{STN} is handled by the first version of $\mathcal{STN}$. The second (final) version $\mathcal{STN}$ does not derive $\neg m$. But if we apply $\mathcal{STN}$ to $P'_{STN} \cup P_{or}$, then $\neg m$ is derivable. This shows that *weak Modularity* is not satisfied: we consider this to be a serious shortcoming. The disjunctive version DSTABLE of the stable semantics is an extension of either of the two versions of STN. Again DSTABLE is inconsistent for many programs. In Dix and Müller (1994b), we discuss a consistent version DSTABLE^{rel} extending STABLE^{rel}.

During an e-mail discussion with Teodor Przymusinski, the simpler program P_{DWFS} presented in Example 3.19 was found: P_{DWFS} shows the failure of *cautious Monotony* for $\mathcal{STN}$.

In this paper we did neither give precise definitions of disjunctive semantics defined on the whole class of all disjunctive programs, nor did we consider our weak properties for this class. In the following, we sketch how this could be done and be used to give easy and elegant definitions of our semantics D-WFS and WD-WFS. First we need to introduce some terminology to facilitate our description. We identify a rule

$$A_1 \vee \wedge \cdots \wedge \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \neg C_1 \wedge \cdots \wedge \neg C_n,$$

with the triple consisting of the following sets of atoms $\mathcal{A} := \{A_1, \ldots, A_k\}$, $\mathcal{B}^+ := \{B_1, \ldots, B_m\}$, $\mathcal{B}^- := \{C_1, \ldots, C_n\}$, and write it as

$$\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg\, \mathcal{B}^-.$$

Given a semantics SEM, we are interested in particular program transformations, namely equivalence transformations:

Definition 5.33 (Equivalence Transformation).
A program transformation $\mapsto$ is a SEM-equivalence transformation if for all P, P' with $P \mapsto P'$: $SEM(P) = SEM(P')$.

A typical transformation of this kind is the elimination of tautological rules like $p \leftarrow p$, or, more general, $p \vee q \leftarrow p \wedge r \wedge \neg\, s$. If a semantics is amendable to bottom-up evaluation, it should not depend on such tautologies, because a tautology can never yield anything new if applied from body to head.

Definition 5.34 (Elimination of Tautologies).
A semantics SEM allows elimination of tautologies, iff the transformation

$$\text{Delete a rule } \mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg\, \mathcal{B}^- \text{ with } \mathcal{A} \cap \mathcal{B}^+ \neq \emptyset.$$

is a SEM-equivalence transformation.

Another, especially important transformation is partial evaluation in the sense of the "unfolding" operation. It is the "Generalized Principle of Partial Evaluation (GPPE)" of Dix and Müller (1994b), Brass and Dix (1994c), Sakama and Seki (1995). In the case of non-disjunctive programs, the idea is simply to replace a positive body literal B by the bodies of all matching rules. This is very natural because the rules with B in their head are usually considered as the only possibilities to derive B. In the disjunctive case, we simply add the disjunctive context to the head of the resulting rule. For instance, if we apply this transformation to q in the body of $p \leftarrow q \wedge r$, and there is the rule $q \vee s \leftarrow t$, then the resulting rule is $p \vee s \leftarrow t \wedge r$.

Definition 5.35 (Generalized Principle of Partial Evaluation).
A semantics SEM satisfies GPPE iff the following transformation on instantiated logic programs is a SEM-equivalence transformation:

Replace a rule $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg\, \mathcal{B}^-$, where $\mathcal{B}^+$ contains an atom B, by the *following n rules* $\mathcal{A} \cup (\mathcal{A}_i \setminus \{B\}) \leftarrow (\mathcal{B}^+ \setminus \{B\}) \cup \mathcal{B}_i^+ \wedge \neg\, (\mathcal{B}^- \cup \mathcal{B}_i^-)$ where $\mathcal{A}_i \leftarrow \mathcal{B}_i^+ \wedge \neg\, \mathcal{B}_i^-$ $(i = 1, \ldots, n)$ are all rules with $B \in \mathcal{A}_i$.

The following transformations express that we can evaluate negative body literals if their truth value is trivial:

Definition 5.36 (Reductions).
A semantics SEM allows positive/negative reduction iff the following are equivalence transformations on instantiated programs:

- *Replace a rule* $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg\, \mathcal{B}^-$ *by* $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg\, (\mathcal{B}^- \setminus \{C\})$ *where* C *appears in no rule head.*
- *Delete a rule* $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg \mathcal{B}^-$ *if there is a disjunctive fact* $\mathcal{A}' \leftarrow \mathit{true}$ *with* $\mathcal{A}' \subseteq \mathcal{B}^-$.

Second, many semantics allow to eliminate non-minimal rules, because they are logically weaker than other rules. However, this does not hold for semantics having an inclusive disjunction:

Definition 5.37 (Elimination of Non-Minimal Rules).
A semantics SEM allows elimination of non-minimal rules iff the following is a SEM-equivalence transformation:

Delete a rule $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg\, \mathcal{B}^-$ s.t.	there is a rule $\mathcal{A}' \leftarrow \mathcal{B}'^+ \wedge \neg\, \mathcal{B}'^-$ with $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{B}'^+ \subseteq \mathcal{B}^+$, $\mathcal{B}'^- \subseteq \mathcal{B}^-$ (where at least one $\subseteq$ is proper).

Definition 5.38 (D-WFS, WD-WFS).
There exists the weakest semantics satisfying positive and negative Reduction, GPPE, Elimination of Tautologies and non-minimal Rules. We call this semantics D-WFS.

Equivalently, D-WFS is the weakest semantics for which the program transformation $P \mapsto \hat{P}$ *(defined by applying all the above defined transformations to* P*) is an equivalence transformation. If we modify this transformation and only apply* GPPE *and* Positive/Negative Reduction*, we get a weaker transformation* $P \mapsto \tilde{P}$*. We call the weakest semantics for which* $P \mapsto \tilde{P}$ *is an equivalence transformation WD-WFS.*

We also mentioned that D-WFS is closely related with the flat part of Przymusinski's static semantics. A problem in comparing both semantics is that Przymusinski's STATIC assigns to any program a certain completion, i.e. a first order theory in an extended language using the belief operator $\mathcal{B}$. Literals $\neg A$ are mapped to $\mathcal{B}\neg A$. But we have the following[19]

Theorem 5.39 (STATIC extends D-WFS).
Let P *be a program and* $A, A_1, \ldots, A_n$ *ground atoms. Then:*

a) *If* $P \mid\!\sim_{\mathrm{D-WFS}} A_1 \vee \ldots \vee A_n$ *then* $\mathit{STATIC}(P) \vdash A_1 \vee \ldots \vee A_n$.

b) *If* $P \mid\!\sim_{\mathrm{D-WFS}} \neg A$ *then* $\mathit{STATIC}(P) \vdash \mathcal{B}(\neg A)$.

c) *STATIC satisfies GPPE, Elimination of Tautologies and Non-minimal Rules and Positive and Negative Reduction.*

[19] A similar result holds for WD-WFS and WSTATIC.

Note that STATIC derives $\mathcal{B}(\neg A_1) \vee \ldots \vee \mathcal{B}(\neg A_n)$ only if it already derives some $\mathcal{B}(\neg A_i)$. Therefore $\neg p \vee \neg q$ can not be considered to be derivable by STATIC or D-WFS (while it is for $\mathcal{STN}$ or PERFECT).

Let us conclude this section with an overview of the properties of some semantics mentioned above (see also Brass and Dix (1995b) for interesting characterizations of DSTABLE):

Properties of Logic-Programming Semantics					
Semantics	Domain	El. Taut.	GPPE	P./N. Red.	Nonmin.
Clark's comp	Nondis.	—	•	•	•
GCWA	Pos.	•	•	(trivial)	•
WGCWA	Pos.	—	•	(trivial)	—
DSTABLE	Dis.	•	•	•	•
WFS	Nondis.	•	•	•	•
$\mathcal{STN}$	Dis.	•	•	•	•
STATIC	Dis.	•	•	•	•
D-WFS	Dis.	•	•	•	•
DWFS	Dis.	•	•	•	•
Strong WFS	Dis.	—	—	•	—
WD-WFS	Dis.	—	•	•	•
WDWFS	Dis.	—	•	•	•

Figure 10 illustrates how the semantics discussed so far are related. Semantics on the same level are defined for the same class of programs:

positive: M_P,

positive disjunctive: WGCWA, GCWA,

stratified: M_P^{supp},

stratified disjunctive: GCWAS, PERFECT, WPERFECT,

normal: WFS, WFS′, WFS$^+$ (=WFS$_C$), REG-SEM, GWFS
STABLE, STABLE′, STABLE$^+$, STABLE^{rel},

general disjunctive: $\mathcal{WSTN}$, $\mathcal{STN}$, WSTATIC, STATIC,
D-WFS, WD-WFS, DWFS, WDWFS,
GDWFS, WF^3, and DSTABLE.

A line between two semantics indicates that one semantics is an extension of the other (in the sense that it coincides with the semantics on the lower level). As already mentioned in the introduction, this notion of an extension is different from $\leq_k$: $SEM_1 \leq_k SEM_2$ simply means that SEM_2 classifies more atoms as true or false than SEM_1. $SEM_1 \leq_k SEM_2$ does not imply that SEM_2 extends SEM_1 in the former sense.

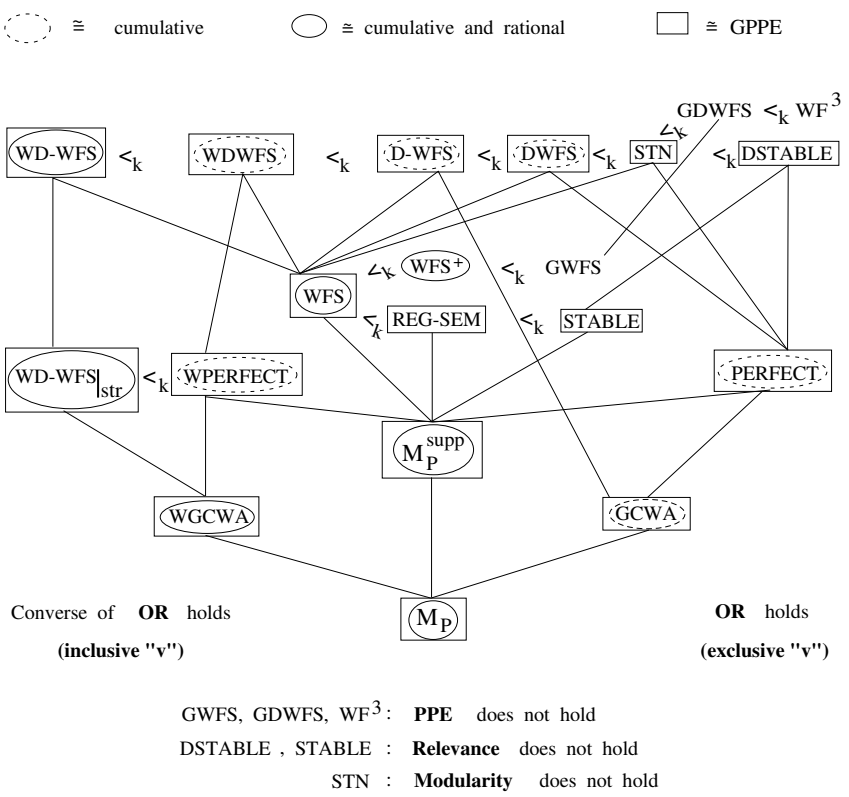

Figure 10: Semantics for Disjunctive Programs

We end this section with some ideas of how to extend our idea of sceptical consequence relation so that it better fits to the disjunctive framework. These ideas will be worked out in detail in forthcoming papers. Note that up to now we considered a semantics as associating to any program P the set of all sceptically derivable *literals*. When we consider disjunctive programs it would be more appropriate to look at the set of all *pure* disjunctions (consisting solely of positive or negative literals), i.e. to the set of all *Herbrand States*. The *sceptical* notion of entailment $\mathrm{SEM}_P^{scept}(U)$ for arbitrary pure disjunctions $a_1 \vee \ldots \vee a_n$ therefore is as follows

$$\bigcap_{\mathcal{M} \in SEM_P(U)} \{a_1 \vee \ldots \vee a_n : a_1 \vee \ldots \vee a_n \text{ is pure with: } \mathcal{M} \models a_1 \vee \ldots \vee a_n\}$$

All our previously introduced weak principles can be extended to the disjunctive case under this broader view. If we restrict to normal programs,

we get back our original *reasonable principles* introduced above (see Dix (1992b), Dix and Müller (1994a)).

5.5 Summary.

- Associate to any semantics a *sceptical consequence relation* and study their abstract properties.
- *Strong* properties, known in the context of classical nonmonotonic logics, can be used to *classify* the existing semantics.
- *Weak* properties are helpful to avoid pathological behaviour of semantics. Semantics should be checked against these properties. *Weak* properties are also important for the computation of a semantics.
- Taken together, *weak* and *strong* properties may characterize a particular semantics *uniquely*.

6. Conclusions

We have seen in Sections 2 and 3 that the intuitions underlying semantics of logic programs differ and we have therefore distinguished between LP-semantics (related to SLDNF) and NMR-semantics (related to loop-detection).

Nevertheless, many techniques from *logic programming*, like using fixpoints of monotone operators and three-valued logic turned out to be useful tools. In addition, *minimal model-reasoning* (Circumscription) played a dominant role in Przymusinski's versions of WFS, $\mathcal{STN}$ and STATIC. The results of Section 4 underline the very strong relationship of NMR-semantics with nonmonotonic logics.

We have also tried (in Section 5) to use another branch of nonmonotonic logics in order to get a classification of the various different semantics: the approach, pioneered by Makinson, Kraus, Lehmann and Magidor, of inspecting abstract properties like *Cumulativity* and *Rationality*. In addition, we presented new principles, specifically tailored to be used in our framework of logic programs. This approach can be seen as an axiomatic framework and attempts to characterize the semantics uniquely by their abstract properties. It turned out that all these properties can also be used to reduce the computational complexity and compute a semantics more efficiently (see Dix and Müller (1993)).

In particular, our abstract approach using GPPE, the generalized principle of partial deduction, gives us a nice semantics D-WFS, which coincides with Przymusinski's STATIC semantics. Moreover, our approach immediately provides us with efficient bottom-up query evaluation methods (Brass and Dix (1995a)) and is therefore close to implementation.

Acknowledgements

Many thanks to Martin Müller who proofread parts of this paper and made a lot of helpful remarks during many talks. Discussions with and useful suggestions of Roland Bol, Thomas Eiter, Heinrich Herre and John Schlipf helped to improve this paper a lot. Roland Bol also suggested to consider $\leq_k$-maximal stable models and Michael Gelfond turned the author's attention to You and Yuan's work. E-mail discussions with Li-Yan Yuan, Jia-Huai You and Teodor Przymusinski helped to correct various mistakes. Hans Rott read the final manuscript very carefully and still discovered many inaccuracies. Special thanks to Michael Kühn who processed the bibliography according to the nonstandard standards of de Gruyter.

The main part of this paper was finished in August 1994, while the author was visiting the Department of Computer Science at Université de Provence, Marseille. I appreciated very much the discussions with the group there, especially with Karl Schlechta and Pierre Siegel.

References

Alferes, J. J. and L. M. Pereira (1992), On Logic Program Semantics with Two Kinds of Negation, in K. Apt (ed.), *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, MIT Press, Cambridge, Mass., 574–589.

——— (1994), Belief, provability, and logic programs, in C. McNish, D. Pearce and L. Pereira (eds.), *JELIA '94, LNAI 838*, Springer, 106–121.

——— (1995a), An argumentation theoretic semantics based on non-refutable falsity, in J. Dix, L. Pereira and T. Przymusinski (eds.), *Nonmonotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin, 3–22.

——— (1995b), Reasoning with Logic Programming, LNCS, Springer, Berlin, to appear.

Apt, K. and D. Pedreschi (1993), Reasoning about termination of pure Prolog programs, *Information and Computation* **106(1)**, 109–157.

Apt, K. R. (1990), Logic programming, in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Vol. B*, Elsevier Science Publishers, 493–574.

Apt, K. R., H. Blair and A. Walker (1988), Towards a theory of declarative knowledge, in J. Minker (ed.), *Foundations of Deductive Databases*, Morgan Kaufmann, 89–148.

Apt, K. R. and H. A. Blair (1990), Arithmetic Classification of perfect Models of stratified Programs, *Fundamenta Informaticae* **XIII**, 1–17.

Apt, K. R. and R. N. Bol (1994), Logic Programming and Negation: A Survey, *Journal of Logic Programming* **19-20**, 9–71.

Balkenius, C. and P. Gärdenfors (1991), Nonmonotonic Inferences in Neural Networks, in J. Allen, R. Fikes and B. Sandewall (eds.), *Proceedings of the second Conference on Principles of Knowledge Representation and Reasoning, Cambridge, Massachusetts*, Morgan Kaufmann, 32–39.

Baral, C. (1992), Generalized Negation As Failure and Semantics of Normal Logic Programs, in A. Nerode and M. Taitslin (eds.), *Logical Foundations of Computer Science – Tver '92*, LNCS 620, Springer, Berlin, 309–319.

Baral, C., J. Lobo and J. Minker (1990a), Generalized Disjunctive Well-founded Semantics for Logic Programs: Declarative Semantics, in Z. Ras, M. Zemankova and M. Emrich (eds.), *Proceedings of the 5th Int. Symp. on Methodologies for Intelligent Systems, Knoxville, TN, October 1990*, North-Holland, 465–473.

——— (1990b), Generalized Disjunctive Well-founded Semantics for Logic Programs: Procedural Semantics, in Z. Ras, M. Zemankova and M. Emrich (eds.), *Proceedings of the 5th Int. Symp. on Methodologies for Intelligent Systems, Knoxville, TN, October 1990*, North-Holland, 456–464.

——— (1990c), Generalized Well-founded Semantics for Logic Programs, in M. E. Stickel (ed.), *10th International Conference on Automated Deduction*, LNAI 449, Springer, Berlin, 102–116.

——— (1991), WF^3: A Semantics for Negation in Normal Disjunctive Logic Programs, in Z. Ras and M. Zemankova (eds.), *Methodologies for Intelligent Systems*, LNAI 542, Springer, Berlin, 459–468.

Baral, C. and V. Subrahmanian (1991), Dualities between Alternative Semantics for Logic Programming and Non-monotonic Reasoning, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 69–86.

——— (1992), Stable and Extension Class Theory for Logic Programs and Default Logics, *Journal of Automated Reasoning* **8, No. 3**, 345–366.

Barback, M. and J. Lobo (1995), A Resolution-based Procedure for Default Theories with Extensions, in J. Dix, L. Pereira and T. Przymusinski (eds.), *Nonmonotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin, 101–126.

BARBUTI, R. and M. MARTELLI (1986), Negation as Failure. Completeness of the Query Evaluation Process for Horn Clause Programs with Recursive Definition, *Journal of Automated Reasoning* **2**, 155–170.

BARWISE, J. (1975), Admissible Sets and Structures, Springer.

BELL, C., A. NERODE, R. T. NG and V. SUBRAHMANIAN (1991), Computation and Implementation of Non-Monotonic Deductive Databases, Tech. Rep. TR-91-158, University of Maryland.

——— (1993), Implementing Stable Semantics by Linear Programming, in L. M. Pereira and A. Nerode (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Second International Workshop*, MIT Press, Cambridge, Mass., 23–42.

BEN-ELIAHU, R. and R. DECHTER (1992), Propositional Semantics for Disjunctive Logic Programs, in K. Apt (ed.), *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, MIT Press, Cambridge, Mass.

BIDOIT, N. and C. FROIDEVAUX (1991a), General logical Databases and Programs: Default Logic Semantics and Stratification, *Information and Computation* **91**, 15–54.

——— (1991b), Negation by Default and unstratifiable logic Programs, *Theoretical Computer Science* **78**, 85–112.

BLAIR, H. A. (1982), The Recursion-Theoretic Complexity of the Semantics of Predicate Logic as a Progr. Lang., *Information and Control* **54**, 25–47.

BOL, R. N. (1991), Loop Checking in Logic Programming, Ph.D. thesis, University of Amsterdam.

BOL, R. N. and L. DEGERSTEDT (1993), Tabulated resolution for well–founded semantics, in *Proc. Int. Logic Programming Symposium'93*, MIT Press, Cambridge, Mass.

BÖRGER, E. (1987), Unsolvable Decision Problems For Prolog Programs, in E. Börger (ed.), *Computation Theory and Logic*, LNCS 270, Springer, Berlin, 37–47.

BRASS, S. (1992), Defaults in Deduktiven Datenbanken, Ph.D. thesis, University of Hannover.

——— (1993), Efficient query evaluation in disjunctive deductive databases, Research report, Institut für Informatik, Universität Hannover.

——— (1994), An Optimization to Bottom-Up Query Evaluation for Disjunctive Databases, in *Proc. of the 13t World Computer Congress '94, IFIP,* GI-Workshop W2 (Disjunctive Logic Programming and Disjunctive Databases).

Brass, S. and J. Dix (1994a), A Characterization of the Stable Semantics by Partial Evaluation, in *Proc. of the 10th Workshop on Logic Programming, Zuerich, October 1994*.

——— (1994b), Computing Disjunctive Stable Semantics based on Clark's Completed Database, in *Proc. of the 6th GI-Workshop "Grundlagen von Datenbanken", Bad Helmstedt, September 1994*, 30–34.

——— (1994c), A disjunctive semantics based on unfolding and bottom-up evaluation, in B. Wolfinger (ed.), *Innovationen bei Rechen- und Kommunikationssystemen,* (IFIP '94-Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases), Springer, Berlin, 83–91.

——— (1995a), A General Approach to Bottom-Up Computation of Disjunctive Semantics, in J. Dix, L. Pereira and T. Przymusinski (eds.), *Nonmonotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin, 127–155.

——— (1995b), Caracterizations of the Stable Semantics by Partial Evaluation, in A. Nerode, W. Marek and M. Truszczyński (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Third International Conference*, LNCS, Springer, Berlin.

——— (1995c), Disjunctive Semantics based upon Partial and Bottom-Up Evaluation, in L. Sterling (ed.), *Proceedings of the 12th Int. Conf. on Logic Programming, Tokyo*, MIT Press.

Brass, S. and U. W. Lipeck (1992), Generalized bottom-up query evaluation, in A. Pirotte, C. Delobel and G. Gottlob (eds.), *Advances in Database Technology — EDBT'92, 3rd Int. Conf.*, LNCS 580, Springer, Berlin, 88–103.

——— (1993), Bottom-up query evaluation with partially ordered defaults, in S. Ceri, K. Tanaka and S. Tsur (eds.), *Deductive and Object-Oriented Databases, Third Int. Conf., (DOOD'93)*, LNCS 760, Springer, Berlin, 253–266.

Brewka, G. (1991a), Cumulative Default Logic: In Defense of Nonmonotonic Inference Rules, *Artificial Intelligence* **50**, 183–205.

——— (1991b), Nonmonotonic Reasoning: Logical Foundations of Commonsense, 12 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press.

——— (1993), An Abductive Framework for Generalized Logic Programs, in L. M. Pereira and A. Nerode (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Second International Workshop*, MIT Press, Cambridge, Mass., 349–365.

Brewka, G., J. Dix and K. Konolige (1993), A Tutorial on Nonmonotonic Reasoning, in G. Brewka, K. P. Jantke and P. H. Schmitt (eds.), *Nonmonotonic and Inductive Logic, Proceedings of the Second International Workshop, Reinhardsbrunn Castle, Germany, December 1991*, LNCS 659, Springer, Berlin, 1–88.

Brewka, G., D. Makinson and K. Schlechta (1991), A Cumulative Inference Relation for JTMS and Logic Programming, in J. Dix, K. P. Jantke and P. H. Schmitt (eds.), *Proceedings of the First Workshop on Nonmonotonic and Inductive Logic, Karlsruhe, 1990*, LNCS 543, Springer, Berlin, 1–12.

Brown, A. L., S. Mantha and T. Wakayama (1992), Preference Logics and Non-Monotonicity in Logic Programming, in A. Nerode and M. Taitslin (eds.), *Logical Foundations of Computer Science – Tver '92*, LNCS 620, Springer, Berlin.

Bruffaerts, A. and E. Henin (1989), Negation as Failure: Proofs, Inference Rules and Meta Interpreters, MIT Press, Cambridge, Mass., 170–190.

Cadoli, M., T. Eiter and G. Gottlob (1992), An efficient method for eliminating varying predicates from a circumscription, *Artificial Intelligence Journal* **54**, 397–410.

Cadoli, M. and M. Schaerf (1993), A Survey of Complexity Results for Non-Monotonic Logics, *Journal of Logic Programming* **17**, 127–160.

Casanova, M. A., A. Hemerly and R. Guerreiro (1991), Explaining SLDNF Resolution with Non-Normal Defaults, in K. Furukawa (ed.), *Proceedings of the 8th Int. Conf. on Logic Programming, Paris*, MIT, 18–31.

Cavedon, L. and J. Lloyd (1989), A Completeness Theorem for SLDNF-Resolution, *Journal of Logic Programming* **7**, 177–191.

Ceri, S., G. Gottlob and L. Tanca (1990), Logic Programming and Databases, Surveys in Computer Science, Springer.

Cerrito, S. (1992), A Linear Axiomatization of Negation As Failure, *Journal of Logic Programming* **12**, 1–24.

Chan, D. (1988), Constructive negation based on the completed database, in *Proc. 1988 Conf. and Symp. on Logic Programming*, 111–125.

Chan, D. and M. Wallace (1989), An Experiment with programming using pure Negation, Tech. Rep. TR, ECRC.

Chen, J. (1992), A Refined Semantics for Disjunctive Logic Programs, in B. Neumann (ed.), *Proceedings ECAI*, John Wiley & Sons, 97–101.

Chen, J. and S. Kundu (1991), The Strong Semantics For Logic Programs, in Z. Ras and M. Zemankova (eds.), *Proceedings of the 6th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1991*, LNAI 542, Springer, Berlin, 490–499.

Chen, W., T. Swift and D. S. Warren (1995), Efficient Top-Down Computation of Queries under the Well-Founded Semantics, *Journal of Logic Programming* forthcoming.

Chen, W. and D. Warren (1992), A Goal-Oriented Approach to Computing Well Founded Semantics, in K. Apt (ed.), *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, MIT Press, Cambridge, Mass., 589–606.

Chen, W. and D. S. Warren (1993), A Goal Oriented Approach to Computing The Well-founded Semantics, *Journal of Logic Programming* **17**, 279–300.

Cholak, P. (1989), Post correspondence problem and prolog programs, Tech. rep., Dept. of Mathematics Wisconsin.

Cholak, P. and H. Blair (1994), The Complexity of Local Stratification, *Fundamenta Informaticae* **XXI(4)**, 333–344.

Chomicki, J. and V. Subrahmanian (1990), Generalized Closed World Assumption is Π^0_2-Complete, *Information Processing Letters* **34**, 289–291.

Clark, K. L. (1978), Negation as Failure, in H. Gallaire and J. Minker (eds.), *Logic and Data-Bases*, Plenum, New York, 293–322.

Colmerauer, A., H. Kanoui, R. Pasero and P. Roussel (1973), Un système de communication homme-machine en français, Tech. rep., Groupe de Intelligence Artificielle Universite de Aix-Marseille II.

Costantini, S. and G. Lanzarone (1995a), Computing stable and partial stable models of extended disjunctive logic programs, in J. Dix, L. Pereira and T. Przymusinski (eds.), *Nonmonotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin, 205–229.

——— (1995b), Static Semantics as Program Transformation and Well-founded Computation, in J. Dix, L. Pereira and T. Przymusinski (eds.), *Nonmonotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin, 156–180.

Decker, H. (1989), The range Form of deductive Databases and Queries or: how to avoid floundering, in *Proc. 5. Österr. AI Tagung*, Informatik Fachberichte 208, Springer, Berlin.

Decker, H. and L. Cavedon (1990), Generalizing syntactic properties which ensure that SLDNF-Resolution is complete and flounder-free, Tech. rep., ECRC Munich.

DEGERSTEDT, L. and U. NILSSON (1995), Magic Computation of Well-founded Semantics, in J. Dix, L. Pereira and T. Przymusinski (eds.), *Nonmonotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin, 181–204.

DHUNG, P. M. and K. KANCHANASUT (1989a), A Fixpoint Approach to Declarative Semantics of Logic Programs, in E. L. Lusk and R. A. Overbeek (eds.), *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, Mass.

——— (1989b), On the Generalized Predicate Completion of Non-Horn Programs, in E. L. Lusk and R. A. Overbeek (eds.), *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, Mass., 587–603.

DIX, J. (1989), Some Tendencies in Non Monotonic Reasoning: I. The Main Approaches, in B. I. Dahn and H. Wolter (eds.), *Proceedings of the 7th Easter Conference on Model-Theory*, 1086 Berlin, PSF 1297, DDR, 81–105.

——— (1991a), Classifying Semantics of Logic Programs, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 166–180.

——— (1991b), Cumulativity and Rationality in Semantics of Normal Logic Programs, in J. Dix, K. P. Jantke and P. H. Schmitt (eds.), *Proceedings of the first Workshop on Nonmonotonic and Inductive Logic 1990 in Karlsruhe*, LNCS 543, Springer, Berlin, 13–37.

——— (1992a), A Framework for Representing and Characterizing Semantics of Logic Programs, in B. Nebel, C. Rich and W. Swartout (eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, Morgan Kaufmann, 591–602.

——— (1992b), Classifying Semantics of Disjunctive Logic Programs, in K. Apt (ed.), *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, MIT Press, Cambridge, Mass., 798–812.

——— (1992c), Default Theories of Poole-Type and a Method for Constructing Cumulative Versions of Default Logic, in B. Neumann (ed.), *Proceedings ECAI*, John Wiley & Sons, 289–293.

——— (1992d), Nichtmonotones Schließen und dessen Anwendung auf Semantiken logischer Programme, Ph.D. thesis, University of Karlsruhe.

——— (1995a), A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties, *Fundamenta Informaticae* **XXII(3)**, 227–255.

——— (1995b), A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties, *Fundamenta Informaticae* **XXII(3)**, 257–288.

——— (1995c), Detailed Report on the First lp & nmr retreat, *The European Journal on Artificial Intelligence* **Vol. 8, No. 1**, 39–43, Also in Computational Logic, Vol. 2, 1995, pages 34–38.

DIX, J., G. GOTTLOB and V. MAREK (1994), Causal Models for Disjunctive Logic Programs, in P. V. Hentenryck (ed.), *Proceedings of the 11th Int. Conf. on Logic Programming, S. Margherita Ligure*, MIT, 290–302.

DIX, J. and M. KUMMER (1994), Herbrand's Theorem for Nonstandard Inference Operations, *Journal of Logic and Computation* **Vol. 4, No. 2**, 143–156.

DIX, J. and D. MAKINSON (1992), The Relationship between KLM and MAK models for Nonmonotonic Inference Operations, *Journal of Logic, Language and Information* **Volume 1, No. 2**, 131–140.

DIX, J. and M. MÜLLER (1992), Abstract Properties and Computational Complexity of Semantics for Disjunctive Logic Programs, in *Proc. of the Workshop W1, Structural Complexity and Recursion-theoretic Methods in Logic Programming, following the JICSLP '92*, H. Blair and W. Marek and A. Nerode and J. Remmel, 15–28, also available as Technical Report 13/93, University of Koblenz, Department of Computer Science.

——— (1993), Implementing Semantics for Disjunctive Logic Programs Using Fringes and Abstract Properties, in L. M. Pereira and A. Nerode (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Second International Workshop*, MIT Press, Cambridge, Mass., 43–59.

——— (1994a), An Axiomatic Framework for Representing and Characterizing Semantics of Disjunctive Logic Programs, in P. V. Hentenryck (ed.), *Proceedings of the 11th Int. Conf. on Logic Programming, S. Margherita Ligure*, MIT, 303–322.

——— (1994b), Partial Evaluation and Relevance for Approximations of the Stable Semantics, in Z. Ras and M. Zemankova (eds.), *Proceedings of the 8th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1994*, LNAI 869, Springer, Berlin, 511–520.

——— (1994c), The Stable Semantics and its Variants: A Comparison of Recent Approaches, in L. Dreschler-Fischer and B. Nebel (eds.), *Proceedings of the 18th German Annual Conference on Artificial Intelligence (KI '94), Saarbrücken, Germany*, LNAI 861, Springer, Berlin, 82–93.

DIX, J., L. PEREIRA and T. PRZYMUSINSKI (eds.) (1995), Non-Monotonic Extensions of Logic Programming, LNAI 927, Springer, Berlin.

Dix, J. and P. Schmitt (1993), Nichtmonotones Schließen: Wieviel Nichtmonotonie ist nötig?, *Kognitionswissenschaft* **3**, 53–69.

——— (1995), Preferential and Rational Closure: A Comparative Study, in *Proceedings of the Colloquium Standardannahmen in Wissens- und Sprachverarbeitung, Bonn, April '92*, LNCS, Springer, Berlin, forthcoming.

Drabent, W. (1994), What is failure? A constructive approach to negation., *Acta Informatica* forthcoming.

Drabent, W. and M. Martelli (1991), Strict completion of Logic Programs, *New Generation Computing* **9(1)**, 69–79.

Dung, P. M. (1991), Negation as Hypotheses: An Abductive Foundation for Logic Programming, in K. Furukawa (ed.), *Proceedings of the 8th Int. Conf. on Logic Programming, Paris*, MIT Press, Cambridge, Mass.

——— (1992), On the relations between stable and wellfounded semantics of logic programs, *Theoretical Computer Science* **105**, 7–25.

Dung, P. M. and P. Ruamviboonsuk (1991), Well-Founded Reasoning with Classical Negation, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 120–133.

Eiter, T. and G. Gottlob (1993), Propositional Circumscription and Extended Closed World Reasoning are Π_2^P-complete, *Theoretical Computer Science* **144**, 231–245, Addendum: vol. 118, p. 315, 1993.

Eiter, T., G. Gottlob and H. Mannila (1993), Expressive Power and Complexity of Disjunctive DATALOG, in *Proceedings of Workshop on Logic Programming with Incomplete Information, Vancouver Oct. 1993, following ILPS' 93*, 59–79.

Elkan, C. (1990), A Rational Reconstruction of Nonmonotonic Truth Maintenance Systems, *Artificial Intelligence* **43**, 219–234.

Eshgi, K. and R. A. Kowalski (1989), Abduction Compared with Negation by Failure, in G. Levi and M. Martelli (eds.), *Proceedings of the 6th Int. Conf. on Logic Programming*, MIT, 234–255.

Fages, F. (1993), Consistency of Clark's completion and existence of stable models, *Methods of Logic in Computer Science* **2**.

Fernández, J. A. and J. Minker (1995), Computing Perfect Models of Disjunctive Stratified Databases, *Annals of Mathematics and Artificial Intelligence* to appear.

Fernández, J. A., J. Minker and A. Yahya (1994a), Computing Perfect and Stable Models Using Ordered Model Trees, *Journal of Computational Intelligence* .

——— (1994b), Ordered Model Trees: A Normalform for Disjunctive Deductive Databases, *Journal of Automated Reasoning* **13(1)**, 117–143.

FITTING, M. C. (1985), A Kripke-Kleene Semantics of logic Programs, *Journal of Logic Programming* **4**, 295–312.

——— (1989), Negation as Refutation, in *Proc. of 4th Annual Symposion on Logic in Computer Science (LICS)*, 63–70.

——— (1991), Bilattices and the Semantics of Logic Programming, *Journal of Logic Programming* **11**, 91–116.

——— (1992), Well-Founded Semantics, Generalized, *Journal of Logic Programming* **1**, 71–84.

FITTING, M. C. and M. BEN-JACOB (1990), Stratified, weak stratified and three valued Semantics, *Fundamenta Informaticae* **XIII**, 19–33.

GABBAY, D. (1984), Theoretical Foundations for Non-Monotonic Reasoning in Expert- Systems, in K. Apt (ed.), *Logics and Models of Concurrent Systems*, Springer, Berlin, 439–458.

——— (1991), Theoretical Foundations for Non-Monotonic Reasoning, Part 2, Structured Non-Monotonic Theories, in *Proceedings of the Third Scandinavian Conference on AI, SCAI '91*, IOS-Press, Amsterdam, 19–40.

GEFFNER, H. (1989), Default Reasoning: Causal and Conditional Theories, Ph.D. thesis, Cognitive Science Laboratory, Dept. of Comp. Science, UCLA, Los Angeles.

——— (1991), Beyond Negation as Failure, in J. Allen, R. Fikes and B. Sandewall (eds.), *Proceedings of the second Conference on Principles of Knowledge Representation and Reasoning, Cambridge, Massachusetts*, Morgan Kaufmann, 218–229.

——— (1992), Reasoning with Defaults in a Logic Programming Framework, Tech. Rep. TR, IBM T. J. Watson Research Center, forthcoming.

GELFOND, M. (1991), Strong Introspection, in *Proceedings 12th International Joint Conference on Artificial Intelligence*, 386–391.

GELFOND, M. and V. LIFSCHITZ (1988), The Stable Model Semantics for Logic Programming, in R. Kowalski and K. Bowen (eds.), *5th Conference on Logic Programming*, MIT Press, 1070–1080.

——— (1989), Compiling Circumscriptive Theories into Logic Programs, in Reinfrank, de Kleer, Ginsberg and Sandewall (eds.), *Non-Monotonic Reasoning*, LNAI 346, Springer, Berlin, 74–99.

——— (1990), Logic Program with Classical Negation, in D. H. Warren and P. Szeredi (eds.), *Proceedings of the 7th Int. Conf. on Logic Programming*, MIT, 579–597.

——— (1991), Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing* **9**, 365–387, (Extended abstract appeared in: Logic Programs with Classical Negation. *Proceedings of the 7-th International Logic Programming Conference, Jerusalem*, pages 579-597, 1990. MIT Press.).

——— (1992), Representing Actions in Extended Logic Programming, in K. Apt (ed.), *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, MIT Press, Cambridge, Mass., 559–573.

Gelfond, M., H. Przymusinska and T. Przymusinski (1989), On the Relationship between Circumscription and Negation as Failure, *Artificial Intelligence* **38**, 75–94.

Gelfond, M., H. Przymusinska and T. Przymusinski (1990), On the Relationship between CWA, minimal model and minimal Herbrand model Semantics, *International Journal of Intelligent Systems* **5**, 549–564.

Ginsberg, M. L. (1991), Negative Subgoals With Free Variables, *Journal of Logic Programming* **11**, 271–293.

Gurevich, Y. (1988), Logic and the Challenge of Computer Science, in E. Börger (ed.), *Trends in Theoretical Computer Science*, Computer Science Press.

Harland, J. (1989), A Kripke-like Model for Negation-as-Failure, in E. L. Lusk and R. A. Overbeek (eds.), *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, Mass., 626–642.

Herre, H. (1988), Negation and Constructivity in Logic Programming, *New Generation Computing* **1**, 295–305.

——— (1991), Nonmonotonic Reasoning and Logic Programs, in J. Dix, K. P. Jantke and P. H. Schmitt (eds.), *Proceedings of the first Workshop on Nonmonotonic and Inductive Logic 1990 in Karlsruhe*, LNCS 543, Springer, Berlin, 38–58.

Hu, Y. and L. Y. Yuan (1991), Extended Well-Founded Model Semantics for General Logic Programs, in K. Furukawa (ed.), *Proceedings of the 8th Int. Conf. on Logic Programming, Paris*, MIT, 412–425.

Imielinski, T. (1991), Incomplete Deductive Databases, *Annals of Mathematics and Artificial Intelligence* **3**, 259–294.

Jaffar, J. and P. J. Stuckey (1986), Canonical logic programs, *Journal of Logic Programming* **3**, 143–155.

Jonker, C. (1993), Implicit and explicit negation in logic programming, in *Dutch/German Workshop on Nonmonotonic Reasoning, Aachen.*

Jonker, C. and G. de Lavalette (1992), A tractable algorithm for the wellfounded model, Tech. Rep. Logic Group Preprint Series No. 746, Utrecht University, Dept. of CS.

Jonker, C. and C. Witteveen (1993), Ambiguity and Undefinedness in Nonmonotonic Semantics, in *Dutch/German Workshop on Nonmonotonic Reasoning, Aachen.*

Kakas, A. C., R. A. Kowalski and F. Toni (1993), Abductive Logic Programming, *Journal of Logic and Computation* **2(6)**, 719–770.

Kakas, A. C. and P. Mancarella (1991), Negation as Stable Hypotheses, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 275–288.

Kemp, D. B., P. J. Stuckey and D. Srivastava (1991), Magic Sets and Bottom-Up Evaluation of Well-Founded Models, in V. Saraswat and K. Ueda (eds.), *Proceedings of the 1991 Int. Symposium on Logic Programming*, MIT, 337–351.

Kemp, D. B. and R. Topor (1988), Completeness of a Top-down Query Evaluation Procedure for Stratified Databases, in *Proceedings of the 5th Int. Conf. and Symposium on Logic Programming, Seattle, USA*, 178–194.

Konolige, K. (1988), On the Relation between Default and Autoepistemic Logic, *Artificial Intelligence* **35**, 343–382.

Kowalski, R. (1974), Predicate logic as a programming language, in *Proceeedings IFIP' 74*, North Holland Publishing Company, 569–574.

Kraus, S., D. Lehmann and M. Magidor (1990), Nonmonotonic Reasoning, Preferential Models and Cumulative Logics, *Artificial Intelligence* **44**, 167–207.

Kunen, K. (1987), Negation in Logic Programming, *Journal of Logic Programming* **4**, 289–308.

——— (1989), Signed Data Dependencies, *Journal of Logic Programming* **7**, 231–245.

——— (1990), Some Remarks on the completed Database, *Fundamenta Informaticae* **XIII**, 35–49.

Lehmann, D. (1989), What does a conditional Knowledge Base entail?, in *Proceedings of the first KR-Conference, Toronto*, 212–222.

——— (1992), Plausibility Logic, in E. Börger, G. Jäger, H. Kleine-Büning and M. Richter (eds.), *Computer Science Logic, 5th Workshop, CSL '91, Berne, Switzerland*, LNCS 626, Springer, Berlin, 227–241.

LEHMANN, D. and M. MAGIDOR (1992), What does a conditional knowledge base entail?, *Artificial Intelligence* **55**, 1–60.

LI, L.-W. and J.-H. YOU (1991), Making Default Inferences from Logic Programs, *Journal of Computational Intelligence* **7**, 142–153.

LIFSCHITZ, V. (1988), On the declarative semantics of logic programs with Negation, in J. Minker (ed.), *Foundations of Deductive Databases*, Morgan Kaufmann, 177–192.

——— (1991), Nonmonotonic Databases and Epistemic Queries, in *Proceedings 9th National Conference on Artificial Intelligence*, 381–386.

——— (1992), Minimal Belief and Negation as Failure, Tech. Rep. Draft, University of Texas at Austin.

LLOYD, J. W. (1987), Foundations of Logic Programming, Springer, Berlin.

LLOYD, J. W. and J. C. SHEPHERDSON (1991), Partial Evaluation in Logic Programming, *Journal of Logic Programming* **11**, 217–242.

LOBO, J., J. MINKER and A. RAJASEKAR (1988), Weak Completion Theory of Non-Horn Programs, in R. Kowalski and K. Bowen (eds.), *5th Conference on Logic Programming*, MIT Press, 828–842.

——— (1989), Extending the semantics of logic programs to disjunctive logic programs, in G. Levi and M. Martelli (eds.), *Proceedings of the 6th Int. Conf. on Logic Programming*, MIT, 255–267.

——— (1992), Foundations of Disjunctive Logic Programming, MIT-Press.

LUDÄSCHER, B. (1991), CNF-Prolog: A Meta-Interpreter for Chan's Constructive Negation, Implementation, Tech. rep., Master Thesis, Karlsruhe University (in german).

MAHER, M. (1988), Complete Axiomatizations of the Algebras of finite, infinite and rational Trees, in *Proc. third Annual Symposium on Logic in Computer Science*, Computer Society Press, 348–359.

MAKINSON, D. (1994), General Patterns in Nonmonotonic Reasoning, in D. Gabbay, C. Hogger and J. Robinson (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 2, Nonmonotonic and Uncertain Reasoning*, Oxford University Press, 35–110.

MANCARELLA, P., S. MARTINI and D. PEDRESCHI (1988), Complete logic Programs with domain-closure Axiom, *Journal of Logic Programming* **5**, 263–276.

MAREK, V. and M. TRUSZCZYŃSKI (1993), Nonmonotonic Logics; Context-Dependent Reasoning, Springer, Berlin.

Marek, W., A. Rajasekar and M. Truszczyński (1992), Complexity of Computing with Extended Propositional Logic Programs, in *Proc. of the Workshop W1, Structural Complexity and Recursion-theoretic Methods in Logic Programming, following the IJCSLP '92*, H. Blair and W. Marek and A. Nerode and J. Remmel, 93–102.

Marek, W. and V. Subrahmanian (1989), The relationship between logic program semantics and non-monotonic reasoning, in G. Levi and M. Martelli (eds.), *Proc. of the 6. Int. Conf on Logic Programming*, MIT, 600–617.

Marek, W. and M. Truszczyński (1989a), Relating autoepistemic and default logics, in R. Brachman, H. Levesque and R. Reiter (eds.), *Proc. of the 1. Int. Conf. on Principles of Knowledge Representation and Reasoning, Toronto, Ontario*, Morgan Kaufmann, 276–288.

——— (1989b), Stable semantics for logic programs and default theories, Tech. rep., Dept. CS. Kentucky.

Minker, J. (1982), On indefinite databases and the closed world assumption, in *Proceedings of the 6th Conference on Automated Deduction, New York*, Springer, Berlin, 292–308.

——— (1988a), Foundations of Deductive Databases, Morgan Kaufmann, 95 First Street, Los Altos, CA 94022.

——— (1988b), Perspectives in deductive Databases, *Journal of Logic Programming* **5**, 33–60.

——— (1989), Toward a Foundation of Disjunctive Logic Programming, in E. L. Lusk and R. A. Overbeek (eds.), *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, Mass., 1215–1235.

——— (1991), An Overview of Nonmonotonic Reasoning and Logic Programming, CS-TR 2736, Computer Science Dept., Univ. Maryland, University of Maryland, College Park, Maryland, 20742, USA.

——— (1993), An Overview of Nonmonotonic Reasoning and Logic Programming, *Journal of Logic Programming, Special Issue* **17**.

Minker, J. and C. Ruiz (1994), Semantics for Disjunctive Logic Programs with Explicit and Default Negation, *Fundamenta Informaticae* **XX(3/4)**.

Moschovakis, Y. N. (1974), Elementary Induction on Abstract Structures, North-Holland.

Müller, M. (1992a), Disjunctive Logic Programs: Characterization and Implementation, Master's thesis, Universität Karlsruhe, (in German).

——— (1992b), Examples and Run-Time Data from KORF.

MÜLLER, M. and J. DIX (1993), Implementing Semantics for Disjunctive Logic Programs Using Fringes and Abstract Properties, in L. M. Pereira and A. Nerode (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Second International Workshop*, MIT Press, Cambridge, Mass., 43–59.

NERODE, A., R. T. NG and V. SUBRAHMANIAN (1991), Computing Circumscriptive Deductive Databases, CS-TR 91-66, Computer Science Dept., Univ. Maryland, University of Maryland, College Park, Maryland, 20742, USA.

PEREIRA, L. M., J. ALFERES and J. APARICIO (1991), Contradiction Removal within Well-founded Semantics, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 105–119.

——— (1992), Adding Closed World Assumptions to Well Founded Semantics, in ICOT (ed.), *Fifth Generation Computer Systems, Proceedings of the Conference*, OMSHA, 562–569.

PEREIRA, L. M., J. APARICIO and J. ALFERES (1993), Non-Monotonic Reasoning with Logic Programming, *Journal of Logic Programming* **17**, 227–264.

PIMENTEL, S. G. and J. L. CUADRADO (1989), A Truth-Maintenance System based on stable models, in E. L. Lusk and R. A. Overbeek (eds.), *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, Mass., 274–290.

PLAZA, J. A. (1991), Completeness For Propositional Logic Programs With Negation, in Z. Ras and M. Zemankova (eds.), *Proceedings of the 6th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1991*, LNAI 542, Springer, Berlin, 600–609.

PRZYMUSINSKA, H. and T. PRZYMUSINSKI (1993), Translating non-monotonic theories into logic programs, Research report, University of California at Riverside, (In preparation).

——— (1994), Stationary extensions of default theories, *Fundamenta Informaticae* (In print.), Special issue devoted to the Fourth Workshop on Non-monotonic Reasoning, Plymouth, Vermont, 1992.

PRZYMUSINSKA, H., T. PRZYMUSINSKI and H. SEKI (1992), Soundness and completeness of partial deductions for well-founded semantics, in A. Voronkov (ed.), *Logic Programming and Automated Reasoning, St. Petersburg, Russia, July 1992.*, LNAI 624, Springer, Berlin, 1–12.

PRZYMUSINSKI, T. (1988), On the declarative semantics of deductive databases and logic programs, in J. Minker (ed.), *Foundations of Deductive Databases*, Morgan Kaufmann, 193–216.

Przymusinski, T. (1989a), Every Logic Program has a natural Stratification and an iterated fixed point Model, in *Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, ACM Press, 11–21.

——— (1989b), On the declarative and procedural Semantics of logic Programs, *Journal of Automated Reasoning* **5**, 167–205.

——— (1990a), Well-founded Semantics Coincides With Three-Valued Stable Semantics, *Fundamenta Informaticae* **XIII**, 445–463.

——— (1991a), Autoepistemic Logics of Closed Beliefs and Logic Programming, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 3–20.

——— (1991b), Semantics of Disjunctive Logic Programs and Deductive Databases, Tech. rep., Department of Computer Science, University of California at Riverside.

——— (1991c), Stable Semantics for Disjunctive Programs, *New Generation Computing Journal* **9**, 401–424, (Extended abstract appeared in: Extended stable semantics for normal and disjunctive logic programs. *Proceedings of the 7-th International Logic Programming Conference, Jerusalem*, pages 459–477, 1990. MIT Press, Cambridge, Mass.).

——— (1991d), Stationary Semantics for Normal and Disjunctive Logic Programs, in C. Delobel, M. Kifer and Y. Masunaga (eds.), *DOOD '91, Proceedings of the 2nd International Conference*, Springer, Berlin, LNCS 566.

——— (1995), Static Semantics For Normal and Disjunctive Logic Programs, *Annals of Mathematics and Artificial Intelligence* **Special Issue on Disjunctive Programs**, to appear.

Przymusinski, T. and D. S. Warren (1992), Wellfounded Semantics: Theory and Implementation, Tech. rep., Department of Computer Science, University of California at Riverside and SUNY at Stony Brook.

Przymusinski, T. C. (1990b), Non-monotonic reasoning vs. logic programming: A new perspective, in D. Partridge and Y. Wilks (eds.), *The Foundations of Artificial Intelligence. A Sourcebook*, Cambridge University Press, London, 49–79.

——— (1991e), Three-valued non-monotonic formalisms and semantics of logic programs, *Artificial Intelligence* **49**, 309–343, (Extended abstract appeared in: Three-valued non-monotonic formalisms and logic programming. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89), Toronto, Canada*, pages 341–348, Morgan Kaufmann, 1989.).

RAJASEKAR, A. (1991), Disjunctive Logic Programs and Constraint Processing, in Z. Ras and M. Zemankova (eds.), *Proceedings of the 6th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1991*, LNAI 542, Springer, Berlin, 610–619.

RAJASEKAR, A., J. LOBO and J. MINKER (1989a), Skeptical Reasoning and Disjunctive Programs, in *Proceedings of the 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*.

——— (1989b), Weak Generalized Closed World Assumption, *Journal of Automated Reasoning* **5**, 293–307.

RAJASEKAR, A. and J. MINKER (1989), A Stratification Semantics for General Disjunctive Programs, in E. L. Lusk and R. A. Overbeek (eds.), *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, Mass., 573–586.

RAMAKRISHNAN, R. (1991), Magic Templates: A Spellbinding Approach to Logic Programs, *Journal of Logic Programming* **11**, 189–216.

RAMAKRISHNAN, R. and S. SUDARSHAN (1991), Top-Down vs. Bottom-Up Revisited, in V. Saraswat and K. Ueda (eds.), *Proceedings of the 1991 Int. Symposium on Logic Programming*, MIT Press, Cambridge, Mass., 321–335.

ROSS, K. A. (1989), The well-founded semantics for disjunctive logic programs, in *Proceedings of the first International Conference on Deductive and Object Oriented Databases, Kyoto, Japan*, 1–22.

——— (1990), The well founded semantics for disjunctive logic programs, in W. Kim, J.-M. Nicolas and S. Nishio (eds.), *Deductive and Object-Oriented Databases, Proc. of the First International Conference (DOOD'89)*, North-Holland, 385–402.

——— (1992), A procedural semantics for well-founded negation in logic programs, *Journal of Logic Programming* **13**, 1–22.

ROSS, K. A. and R. A. TOPOR (1988), Inferring negative Information from disjunctive Databases, *Journal of Automated Reasoning* **4**, 397–424.

SACCA, D. (1993), The Expressive Power of Stable Models For DATALOG Queries with Negation, in *Proceedings of Workshop on Logic Programming with Incomplete Information, Vancouver Oct. 1993, following ILPS' 93*, 150–162.

SACCA, D. and C. ZANIOLO (1991), Partial models and Three-Valued Models in Logic Programs with Negation, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 87–104.

Sagiv, Y. (1990), Is There Anything Better Than Magic, in S. Debray and M. Hermenegildo (eds.), *Proceedings of the 1990 North American Conference on Logic Programming*, MIT Press, Cambridge, Mass., 235–253.

Sakai, K. and T. Miyachi (1983), Incorporating naive negation into Prolog, Tr-028, ICOT.

Sakama, C. and K. Inoue (1993), Negation in Disjunctive Logic Programs, in D. Warren and P. Szeredi (eds.), *Proceedings of the 10th Int. Conf. on Logic Programming, Budapest*, MIT Press, Cambridge, Mass.

Sakama, C. and H. Seki (1995), Partial Deduction of Disjunctive Logic Programs: A Declarative Approach, in *Fourth International Workshop on Logic Program Synthesis and Transformation (LOPSTR'94)*, LNCS, Springer, Berlin, forthcoming.

Sato, T. (1990), Completed Logic Programs and their Consistency, *Journal of Logic Programming* **9**, 33–44.

Schaerf, M. (1993), Negation and minimality in non-Horn databases, in C. Beeri (ed.), *Proceedings of the Twelfth Conference on Principle Of Database Systems (PODS-93)*, ACM Press, 147–157.

Schlipf, J. S. (1990a), Formalizing a Logic for Logic Programming, in *Proceedings of the International Symposium on Artificial Intelligence '90*.

——— (1990b), The Expressive Powers of the Logic Programming Semantics, in *Proceedings of the Ninth ACM Symposium on Principles of Databases*, 196–204.

——— (1992a), A Survey of Complexity and Undecidability Results in Logic Programming, in H. Blair, W. Marek, A. Nerode and J. Remmel (eds.), *Proceedings of the Workshop on Complexity and Recursion-theoretic Methods in Logic Programming, following the JICSLP'92*, informal.

——— (1992b), Formalizing a Logic for Logic Programming, *Annals of Mathematics and Artificial Intelligence* **5**, 279–302.

Seipel, D. (1994), An efficient computation of the extended generalized closed world assumption by support-for-negation sets, in *Proc. Int. Conf. on Logic Programming and Automated Reasoning (LPAR'94)*, LNAI 822, Springer, Berlin, 245–259.

Seipel, D. and H. Argenton (1993), A data structure for efficient deduction in disjunctive logic programs, in C. Beierle (ed.), *9. Workshop Logische Programmierung*, Informatik Berichte 146-10/1993, FernUniversiät Hagen, 90–94.

SHEPHERDSON, J. C. (1984), Negation as Failure: A Comparison of Clark's completed Database and Reiter's CWA, *Journal of Logic Programming* **1**, 51–79.

SHEPHERDSON, J. C. (1985), Negation as failure. II, *Journal of Logic Programming* **I**, 185–202.

——— (1988), Language and Equality Theory in Logic Programming, Pm-88-08, School of Mathematics, University of Bristol, School of Mathematics, University Walk.

SHEPHERDSON, J. C. (1988), Negation in Logic Programming, in J. Minker (ed.), *Foundations of Deductive Databases*, Morgan Kaufmann, 19–88.

SHEPHERDSON, J. C. (1989), A sound and complete Semantics for a Version of Negation as Failure, *Theoretical Computer Science* **65**, 343–371.

——— (1991a), Correct Answers to allowed Programs and Queries are ground, *Journal of Logic Programming* **3 + 4**, 359–362.

——— (1991b), Unsolvable Problems for SLDNF-Resolution, *Journal of Logic Programming* **10**, 19–22.

STÄRK, R. F. (1994), Input/output dependencies of normal logic programs, *Journal of Logic and Computation* **4**, 249–262.

SUBRAHMANIAN, V., NAU and VAGO (1992), WFS + Branch and Bound = Stable Models, Technical Report CS-TR-2935, UMIACS-TR-92-82, University of Maryland, Submitted for publication.

TAMAKI, H. and T. SATO (1986), OLD Resolution with Tabulation, in *Proceedings of the Third International Conference on Logic Programming, London*, LNAI, Springer, Berlin, 84–98.

TÄRNLUND, S.-A. (1977), Horn Clause Computability, *BIT* **17**, 215–226.

TRUSZCZYŃSKI, M. (1991), Embedding Default Logic into Modal Nonmonotonic Logics, in A. Nerode, W. Marek and V. S. Subrahmanian (eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, MIT Press, Cambridge, Mass., 151–165.

ULLMAN, J. D. (1989), Bottom-up Beats Top-down for Datalog, in *Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, ACM Press, 140–149.

ÜMIT YALÇINALP, L. and L. STERLING (1988), An Integrated Interpreter for Explaining Prolog's Successes and Failures, in H. Abramson and M. H. Rogers (eds.), *Meta-Programming in Logic Programming*, MIT-Press, 192–203.

VAN EMDEN, M. and R. KOWALSKI (1976), The semantics of predicate logic as a programming language, *JACM* **23**, 733–742.

VAN GELDER, A. (1989), The Alternating Fixpoint of Logic Programs with Negation, in *Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, ACM Press, 1–10.

VAN GELDER, A., K. A. ROSS and J. S. SCHLIPF (1988), Unfounded Sets and well-founded Semantics for general logic Programs, in *Proceedings 7th Symposion on Principles of Database Systems*, 221–230.

——— (1991), The well-founded semantics for general logic programs, *Journal of the ACM* **38**, 620–650.

VIEILLE, L. (1989), Recursive Query Processing: The Power of Logic, *Theoretical Computer Science* **69**.

VORBECK, M. (1991), CNF-Prolog: A Meta-Interpreter for Chan's Constructive Negation, Theory, Tech. rep., Master Thesis, Karlsruhe University (in german).

WALLACE, M. (1989a), A computable Semantics for general logic Programs, *Journal of Logic Programming* **6**, 269–297.

——— (1989b), Unrestricted logic programs Or If stratification is the cure, what is the malady, Ir-kb-67, ECRC Munich, ECRC, Arabellastr. 17, D-8000 München 1.

——— (1993), Tight, consistent and computable completions for unrestricted logic programs, *Journal of Logic Programming* **15**, 243–273.

WARREN, D. S. (1991), Computing the Well-Founded Semantics of Logic Programs, Technical Report 91/12, Computer Science Department SUNY at Stony Brook.

WITTEVEEN, C. (1991), Skeptical Reason Maintenance is Tractable, in J. Allen, R. Fikes and B. Sandewall (eds.), *Proceedings of the second Conference on Principles of Knowledge Representation and Reasoning, Cambridge, Massachusetts*, Morgan Kaufmann, 570–581.

YOU, J.-H. and L. LI (1989), Supported Circumscription and its Relation to Logic Programming with Negation, in E. L. Lusk and R. A. Overbeek (eds.), *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, Mass., 291–309.

YOU, J.-H. and L.-Y. YUAN (1990), Three-valued Formalization of Logic Programming: is it needed, in *Proc. of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,*, ACM Press, 172–182.

——— (1993), On the Equivalence of Semantics for Normal Logic Programs, in *Proceedings of Workshop on Logic Programming with Incomplete Information, Vancouver Oct. 1993, following ILPS' 93*, 161–176, available as Research Report of ISIS-RR-93-E12, Fujitsu Labs.