SEMANTICS OF (DISJUNCTIVE) LOGIC PROGRAMS BASED ON PARTIAL EVALUATION

STEFAN BRASS, AND JÜRGEN DIX[1]

▷ We present a new and general approach for defining, understanding, and computing logic programming semantics. We consider disjunctive programs for generality, but our results are still interesting if specialized to normal programs. Our framework consists of two parts: *(1)* a *semantical*, where semantics are defined in an abstract way as the *weakest* semantics satisfying certain properties, and *(2)* a *procedural*, namely a bottom-up query-evaluation method based on operators working on *conditional facts*. As to *(1)*, we concentrate in this paper on a particular set of abstract properties (the most important being the *unfolding* or *partial evaluation property* GPPE) and define a new semantics D-WFS, which extends WFS and GCWA. We also mention that various other semantics, like FITTING'S comp$_3$, SCHLIPF'S WFS$_C$, GELFOND/LIFSCHITZ' STABLE and ROSS/TOPOR'S WGCWA (also introduced independantly by RAJASEKAR/LOBO/MINKER), can be captured in our framework. In *(2)* we compute for any program P a residual program $res(P)$, and show that $res(P)$ is equivalent to the original program under very general conditions on the semantics (which are satisfied, e.g., by the well-founded, stable, stationary, and static semantics). Many queries with respect to these semantics can already be answered on the basis of the residual program. In fact, $res(P)$ is *complete* for D-WFS, WFS and GCWA. ◁

[1]Currently on leave from University of Koblenz, Dept. of CS, Rheinau 1, 56075 Koblenz, Germany.

Address correspondence to Jürgen Dix, Dept. of University of Maryland, Computer Science, College Park, Maryland 20742, USA, e-mail: dix@cs.umd.edu and Stefan Brass, School of Information Sciences, University of Pittsburgh, 135 North Bellefield Avenue, Pittsburgh, PA 15260, USA, e-mail: sbrass@sis.pitt.edu

This is a completely revised and much extended version of the papers [BD95b] presented at the 12th International Conference on Logic Programming, Tokyo, June 1995 and of [BD95a].

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/94/$7.00

1. INTRODUCTION

Already for normal programs, there are quite a lot of proposed semantics, but for disjunctive programs, the number of possibilities explodes (for an overview and comparison, see [Dix95a, Dix95b] for normal, and [Dix95c, BD98b, DM94a] for disjunctive programs).

In this paper, we introduce a simple, but powerful framework for *analyzing*, *defining*, and *computing* semantics based on elementary program transformations. For instance, we require that if A appears in no rule head, then it should be possible to evaluate ocurrences of $not\,A$ by true, i.e. to delete $not\,A$ from all rule bodies. For many semantics it is also possible to delete a tautological rule like $p \leftarrow p$ without changing the meaning of the program: this is essentially the step away from classical logic programming semantics (such as CLARK'S completion [Cla78] and its variants) to much stronger nonmonotonic (or deductive database) semantics (such as WFS and STABLE). Other important transformations are unfolding (partial evaluation) and the evaluation of negative body literals in trivial cases.

Our approach has *semantical* as well as *computational* consequences. With respect to *semantics*, our main results are:

1. New characterizations of the standard well-founded semantics ([VGRS91]) as well as of the generalized closed world assumption GCWA ([Min82]) as the weakest semantics allowing our elementary transformations. "Weakest" is meant in the sense of the information ordering: any ground literal whose truth value does not become obvious by our very simple transformations is undefined in the well-founded model. This shows again how natural WFS and GCWA are.

2. When we look at disjunctive programs, the same transformations allow us to define a disjunctive extension of the well-founded semantics, which we call D-WFS. This disjunctive counterpart of WFS has good properties and a natural behaviour, and allows some important transformations already by definition.

3. While we exemplify this method for defining/characterizing semantics in depth only in the case of WFS/D-WFS, it works also with other sets of program transformations or other definitions of the universe of "abstract semantics", which we study. For example FITTING'S 3-valued version comp_3 of CLARK'S completion ([Fit85]), SCHLIPF'S wellfounded by case semantics WFS_C ([Sch92]), GELFOND/LIFSCHITZ' STABLE ([GL88, GL91]) and ROSS/TOPOR'S WGCWA ([RT88, RLM89]) can be captured with appropriate transformation rules.

   We believe that many interesting results will be possible by applying our ideas in slightly modified frameworks. It is important to look at the space of all possible "abstract semantics", because otherwise it can be only by chance that we know certain semantics. There could be semantics with much better properties, which are, however, yet unknown.

4. We also show that a subset of our transformations exactly characterizes those semantics which look only at the minimal models of the given program, and not at its syntax.

Our approach also contributes to the *computation* of semantics. In particular we develop a *bottom-up computation* of semantics allowing our elementary transformations:

1. We show that a normal form, called the *residual program*, can be constructed by our elementary transformations from every (disjunctive) logic program. The residual program consists of conditional facts, i.e. *ground rules without positive body literals*. Conditional facts result from delaying the negative body literals during a bottom-up evaluation of an allowed logic program (the delayed literals are attached as "conditions" to the derived facts). Conditional facts have already been studied in [Bry89, Bry90, DK89b, DK89a, HY91] (for non-disjunctive programs). We especially generalize the T_P-operator and the reductions introduced by Bry to the disjunctive case. However, our main result is the relation of this bottom-up computation to our elementary program transformations.

2. A consequence of this is that the residual program is equivalent to the original program for *any* semantics allowing our elementary program transformations. Besides the well-founded semantics and our D-WFS, for instance also the stable model semantics (for normal and disjunctive programs), the wellfounded-by-case semantics ([Sch92]) as well as the static [Prz95] and stationary [Prz91a] semantics satisfy this condition. For every such semantics, the computation of the residual program can be a useful preprocessing step, because it is equivalent to the original program, but it is ground and contains no positive body literals (and usually very few negative body literals).

   In the case of WFS and D-WFS, the truth value of all ground atoms can be trivially decided based on the residual program: if there is a fact $A \leftarrow \mathit{true}$, A is true, and if A does not occur in any rule head, A is false. All other ground atoms are undefined. For other semantics, like the stable model semantics, only the few "islands of complexity" remain. They must be evaluated with other techniques.

3. It also directly follows from our results that any semantics allowing our transformations is not changed by adding atoms which are true in the well-founded model (see Subsection 4.5). This is an important property extensively used in the recent method of Niemelä and Simons ([NS96]) for computing stable models. Because WFS is of quadratic complexity, and therefore located one level below the stable semantics, this property tells us that first computing WFS does no harm to the set of stable models. It is a restricted form of cautious monotony which pays off from the computational viewpoint. See also [DFN98].

4. Although originally we developed our program transformations just for declarative purposes (we wanted a semantics to satisfy some natural conditions) and used them to prove a completeness result with respect to our bottom-up procedure, it is also possible *to apply them to a given program and compute the normal form directly.* This works because our calculus of transformations is *confluent* and *terminating*: the normal form is therefore uniquely terminated. This use of our transformations is worked out in detail in [BD98a], and we refer the reader to this article.

Let us finally mention related work. Transformational approaches have been considered in [DM93, CL95]. While in [DM93], abstract properties have been used only for speeding up query evaluation in [CL95], the main focus of the program transformation is to make explicit possible uses of disjunctive information. In [CW95], also a "residual program" is computed, but this is done top-down, and their processing of the residual program is quite different. Partial evaluation (unfolding) has also been studied in [SS94, SS97].

Our paper is structured as follows: in Section 2, we first introduce an abstract notion of logic programming semantics. Then we define the elementary program transformations which we use in this paper. We define our semantics D-WFS as the weakest abstract semantics allowing these transformation. At the end of Section 2, we explain that this method for defining and characterizing semantics is in fact much more general, and can be applied to other semantics as well.

In Section 3, we develop a bottom-up query evaluation algorithm based on conditional facts and the residual program. We also give an alternative characterization of the residual program which generalizes to infinite programs or programs with function symbols.

Section 4 contains our main results. We first show that the computed semantics is indeed our abstractly characterized semantics D-WFS. However, we prove in fact much more, namely that the residual program can be reached from the original program by means of our elementary transformations (it is the uniquely determined normal form). This allows to use the computation of the residual program as a preprocessing step for other semantics as well. We also prove a nice characterization of the semantics allowing a subset of our transformations in terms of minimal models. We conclude Section 4 with a proof that it is possible to evaluate literals whose truth value is known in the well-founded model also in any other semantics allowing our transformations. We claim that this is an important property for computing stable models more efficiently (as is well-known, computing stable models is a NP-complete problem, therefore a really efficient, i.e. polynomial algorithm is unlikely to exist). However, substantiating this claim will be subject to future research.

In Section 5 we explain how D-WFS relates to various other semantics. In particular, we establish there the equivalence to the classical well-founded semantics for non-disjunctive programs and to the generalized closed world assumption for positive disjunctive programs. This shows that our results indeed give a nice characterization of the classical WFS.

Computational properties are shortly discussed in Section 6. Finally, we give a short summary and an outlook on future work in Section 7. Most of the proofs have been delayed to an appendix.

2. ABSTRACT SEMANTICS AND TRANSFORMATIONS

In this paper, we consider allowed disjunctive DATALOG$^{\neg}$ programs over some fixed function-free finite signature Σ. In fact, in the semantical part of this paper, we consider only the ground instantiation of the programs, because we claim that any sensible semantics should assign the same meaning to a program P and its instantiation $\mathit{ground}(P)$. So the variables are seen only as a shorthand for denoting ground programs in a more compact way. This means that in the semantical part, we could as well have worked with propositional programs.

However, in the computational part, it would be very inefficient to compute first the ground instantiation of the given program. Here we make use of the allowedness condition: *every variable of the rule must occur also in a positive body literal.* This guarantees that in every rule application, all variables are bound to a constant. It is true that in this way we again manage to consider only ground programs. But we do not consider the complete instantiation of the program — only rule instances with possibly true body literals. We never consider a rule instance containing a positive body literal which does not match any previously derived "conditional fact".

Let us finally clarify the requirement that we work with a fixed finite function-free signature.

1. The signature must be fixed, because our program transformations may change the set of actually occurring symbols, but we wish to keep the syntactic base.

2. In some of our theorems we need that the instantiated program is finite. Of course, this is an important restriction, but it was very fruitful and has lead to a very nice theory.

3. Furthermore, from a more practical viewpoint, we also have to avoid infinite loops. Note that for arbitrary first-order programs, even the wellfounded semantics is highly undecidable (Π^1_1-complete over the integers, see [Sch95]). Therefore, we can not capture such a semantics with elemantary and constructive transformation rules.

For the semantical part, our assumptions are not real restrictions: we have shown in [DS98] that our semantical framework can be generalized to arbitrary disjunctive first-order programs by introducing a non-constructive *Loop-Detection Rule*. This rule is also a transformation rule[1]but in contrast to the rules we consider here, testing whether it can be applied to a particular program is not decidable (this is similar to *local stratifiability*).

Even our bottom-up procedure (to be introduced in the next section) is easily generalized to handle infinite propositional programs (see Subsection 3.3). Thus our assumptions can be seen as suitable properties to ensure feasible computations, but not as essentially *built-in* restrictions of our overall framework.

Definition 2.1. (Logic Program). A logic program P is a finite set of rules of the form

$$A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \mathit{not}\, C_1 \wedge \cdots \wedge \mathit{not}\, C_n,$$

where the $A_i/B_i/C_i$ are Σ-atoms, $k \geq 1$, $m \geq 0$, $n \geq 0$, and every variable of the rule appears in one of the B_i (allowedness). We identify such a rule with the triple consisting of the sets of atoms

$$\mathcal{A} := \{A_1, \ldots, A_k\}, \qquad \mathcal{B} := \{B_1, \ldots, B_m\}, \qquad \mathcal{C} := \{C_1, \ldots, C_n\},$$

and write it as $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}$. The set of all logic programs over Σ will be denoted by LP_Σ.

Definition 2.2. (Instantiation, Possibly True Facts). We write $\mathit{ground}(P)$ for the

[1] To be precise, the rule operates on arbitrary infinite propositional programs.

full instantiation of P (with respect to Σ) and $\mathit{heads}(P)$ for the set of ground atoms occurring in rule heads in $\mathit{ground}(P)$.

2.1. Logic Programming Semantics

Our definition of a logic programming semantics is a very general one. We simply assume that a semantics maps every logic program into a set of pure ground disjunctions.We sometimes call these disjunctions *deduced* formulae, simply meaning that they follow under the semantics.

This includes for instance "modeltheoretic semantics" such as the stable model semantics, which define a set of models for every logic program. We simply deduce those ground disjunctions Q which are true in all of these models (sceptical view). Also, if a semantics assigns to every program a completion (i.e. a first order theory), then we take the *pure disjunctions* which follow from this completion. Pure disjunctions consist either of only positive or of only negative literals: there are no atoms and negative literals *at the same time* (no mixed disjunctions).

Of course, the question arises why we are only interested in deducing pure ground disjunctions:

- First, nonground (universally quantified) formulas cannot be deduced from allowed logic programs.
- Second, it is natural to assume that $Q_1 \wedge Q_2$ is implied if and only if both, Q_1 as well as Q_2, are implied. So a semantics has no freedom to decide which (ground) conjunctions are implied and which are not.

  However, for disjunctions it is possible that $Q_1 \vee Q_2$ is implied, but neither Q_1 nor Q_2 are implied. Therefore it does not suffice to look only at implied ground literals.
- Third, we look only at pure disjunctions because we believe that there is a difference between a logic programming system and an automatic theorem prover. We do not want to conclude implied rules, for instance, but we are interested to answer queries. For instance, the STATIC semantics allows unfolding as long as we look only at the implied pure disjunctions, but unfolding is no equivalence transformation if we look at all possible consequences. As an example, consider the following logic program:

  $p \vee q.$
  $r \leftarrow p.$

  Unfolding the body literal p yields the logic program

  $p \vee q.$
  $r \vee q.$

  While the first program obviously implies the rule $r \leftarrow p$, this is not true for the second program under the STATIC semantics. However, the two programs are equivalent under the STATIC semantics as long as we look only at pure disjunctions (because then only the minimal models of the program are needed).

- Finally, we will define in this paper a disjunctive extension of the well-founded semantics. The standard well-founded semantics for normal programs can be seen as defining sets of positive and negative ground literals which can be concluded from the program. Therefore, it seems natural to extend this in the disjunctive case to positive and negative ground disjunctions.

But in fact, a large part of our theory does not depend on the exact definition of a semantics, and therefore it can be considered as a real framework (see Subsection 2.4).

Definition 2.3. (Semantics). A semantics $\mathcal{S}$ is a mapping which assigns to every logic program $P \in LP_\Sigma$ a set $\mathcal{S}(P)$ of pure disjunctions of ground literals over Σ. It must satisfy the following requirements:

1. $\mathcal{S}(P) = \mathcal{S}(\mathit{ground}(P))$ (instantiation invariance).
2. If $Q \in \mathcal{S}(P)$ and $Q \subseteq Q'$ (i. e. Q is a subdisjunction of Q'), then $Q' \in \mathcal{S}(P)$ (right weakening).
3. If $\mathcal{A} \leftarrow \mathit{true} \in P$ for a disjunction $\mathcal{A}$, then $\mathcal{A} \in \mathcal{S}(P)$ (necessarily true).
4. If $A \notin \mathit{heads}(P)$ for some Σ-ground atom A, then $\mathit{not}\, A \in \mathcal{S}(P)$ (nec. false).

If $Q \in \mathcal{S}(P)$, we also write $P \mid\!\sim_{\mathcal{S}} Q$.

This is a very general definition, and practically all proposed semantics fit into this framework.

It might be argued at this point that our definition is in fact too general, because it does not even guarantee closure under logical consequences. For instance, consider the following program P:

$p.$
$q \leftarrow p.$

Every semantics must allow to conclude p, but q is not nessarily contained in $\mathcal{S}(P)$. Of course, our semantics D-WFS is closed under logic consequences (Theorem 4.4), but this is something we have to prove. There is nothing bad in allowing also strange semantics in the beginning. Our characterizations get only stronger by starting with such weak requirements in the basic definition.

Note also that our notion of a semantics does not neccessarily restrict us to consider only Herbrand-models (although this is one of the main applications). This is because the underlying signature can contain additional constants, not occurring in the given program. Such additional constants often have the effect to avoid problems related to Herbrand-domains, such as the *universal-query-problem* (see [GPP90, Dix95c]).

2.2. Program Transformations

We base our discussion on abstract properties of logic programming semantics. All of them require that certain elementary transformations do not change the semantics of a given logic program:

Definition 2.4. (Program Transformation). A program transformation is any binary relation $\mapsto$ between instantiated logic programs.

For the sake of simplicity, we consider only ground logic programs in the semantical part. This is possible since we have required "instantiation invariance" in our definition of a semantics: a semantics is already completely defined by its values for ground programs.

Note that a program transformation is a relation, and not a function, because we consider only elementary changes like deleting a single tautological rule. This makes it simpler to prove that a semantics allows such a transformation. We have shown in [BD98a] that the rewriting system consisting of the transformations which we consider here is terminating and confluent.

Definition 2.5. (Equivalence Transformation). We call a transformation $\mapsto$ an $\mathcal{S}$-equivalence transformation iff $\mathcal{S}(P_1) = \mathcal{S}(P_2)$ for all ground programs P_1 and P_2 with $P_1 \mapsto P_2$. In this case, we also say that the semantics $\mathcal{S}$ allows the transformation $\mapsto$.

An very important such transformation is partial evaluation in the sense of the "unfolding" operation. It is the "Generalized Principle of Partial Evaluation (GPPE)" ([DM94b, BD97]) (it has also been considered by Sakama and Seki in [SS94, SS97] under the name *partial deduction*). For ground normal logic programs, unfolding simply means to replace a positive body literal by all bodies of rules having this literal in the head. For instance, suppose there are only the following two rules about q:

$q \leftarrow s_1 \wedge \mathit{not}\, t.$
$q \leftarrow s_2.$

Then, if we unfold q in the rule $p \leftarrow q \wedge \mathit{not}\, r$, we get the rules

$p \leftarrow s_1 \wedge \mathit{not}\, t \wedge \mathit{not}\, r.$
$p \leftarrow s_2 \wedge \mathit{not}\, r.$

The original rule $p \leftarrow q \wedge \mathit{not}\, r$ is replaced by these two new rules, i.e. it is not contained in the resulting program. In the case of disjunctive programs, we must also treat a possible disjunctive context of the unfolded literal in a rule head. The formal definition is as follows:

Definition 2.6. (GPPE). A ground program P_2 results from a ground program P_1 by unfolding ($P_1 \mapsto_U P_2$) iff there is a rule $\mathcal{A} \leftarrow \big(\mathcal{B} \cup \{B\}\big) \cup \mathit{not}\, \mathcal{C}$ in P_1 such that

$$\begin{aligned} P_2 = P_1 &- \big\{\mathcal{A} \leftarrow (\mathcal{B} \cup \{B\}) \cup \mathit{not}\, \mathcal{C}\big\} \\ &\cup \big\{\mathcal{A} \cup (\mathcal{A}' - \{B\}) \leftarrow (\mathcal{B} \cup \mathcal{B}') \wedge \mathit{not}\, (\mathcal{C} \cup \mathcal{C}') \;\big|\; \\ &\qquad \mathcal{A}' \leftarrow \mathcal{B}' \wedge \mathit{not}\, \mathcal{C}' \text{ is a rule in } P_1 \text{ with } B \in \mathcal{A}'\big\}. \end{aligned}$$

We will need also a weak version of GPPE, which does not eliminate the original rule, but only adds the result of unfolding:

Definition 2.7. (Weak GPPE). A ground program P_2 results from a ground program P_1 by weak unfolding ($P_1 \mapsto_W P_2$) iff there is a rule $\mathcal{A} \leftarrow (\mathcal{B} \cup \{B\}) \cup not\,\mathcal{C}$ in P_1 such that

$$P_2 = P_1 \cup \big\{\mathcal{A} \cup (\mathcal{A}' - \{B\}) \leftarrow (\mathcal{B} \cup \mathcal{B}') \wedge not\,(\mathcal{C} \cup \mathcal{C}')\ \big|\ \mathcal{A}' \leftarrow \mathcal{B}' \wedge not\,\mathcal{C}' \text{ is a rule in } P_1 \text{ with } B \in \mathcal{A}'\big\}.$$

The next property we need is that tautological clauses like $p \leftarrow p$ do not influence the semantics of a logic program. Note that tautologies never imply new facts in non-disjunctive programs (we must know p already in order to derive p by means of such a rule). In disjunctive programs, rules like $p \vee q \leftarrow p$ can imply subsumed (nonminimal) disjunctive facts, which we will consider also as irrelevant here (see below). This and the following property together correspond to the "equivalence" principle of [Dix95b]. They require that it should be possible to delete obviously redundant rules.

Definition 2.8. (Elimination of Tautologies). A ground program P_2 results from a ground program P_1 by elimination of tautologies ($P_1 \mapsto_T P_2$) iff there is a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge not\,\mathcal{C}$ in P_1 such that $\mathcal{A} \cap \mathcal{B} \neq \emptyset$ and $P_2 = P_1 - \{\mathcal{A} \leftarrow \mathcal{B} \wedge not\,\mathcal{C}\}$.

Lemma 2.1. If a semantics $\mathcal{S}$ allows unfolding and the elimination of tautologies, it also allows weak unfolding.

PROOF. Let $P_1 \mapsto_W P_2$ and let B be the unfolded literal.

1. If P_1 contains the tautology $B \leftarrow B$, unfolding and weak unfolding give the same result, because the rule $head \leftarrow \big(\mathcal{B} \wedge \{B\}\big) \wedge not\,\mathcal{C}$ is reproduced when inserting $B \leftarrow B$. Therefore, also $P_1 \mapsto_U P_2$ holds and thus $\mathcal{S}(P_1) = \mathcal{S}(P_2)$.

2. If P_1 does not contain this tautology, let $P_1' := P_1 \cup \{B \leftarrow B\}$ and $P_2' := P_2 \cup \{B \leftarrow B\}$. So we have $P_1' \mapsto_T P_1$ and $P_2' \mapsto_T P_2$. As explained above, $P_1' \mapsto_U P_2'$ holds. But since $\mathcal{S}$ allows $\mapsto_T$ and $\mapsto_U$, we have $\mathcal{S}(P_1) = \mathcal{S}(P_1') = \mathcal{S}(P_2') = \mathcal{S}(P_2)$. □

The next transformation allows to delete a rule if we already have a stronger rule. For instance, $p \leftarrow q \wedge not\, r$ is not necessary if we also have the rule $p \leftarrow q$. This transformation is possible for all proposed semantics for normal programs. In the case of disjunctive programs, it also allows to delete a rule like $p \vee q \leftarrow r$ if there is already the rule $p \leftarrow r$. Therefore it is satisfied only for semantics with the non-inclusive interpretation of $\vee$.

Definition 2.9. (Elimination of Nonminimal Rules). A ground program P_2 results from a ground program P_1 by elimination of nonminimal rules ($P_1 \mapsto_M P_2$) iff there are two distinct rules $\mathcal{A} \leftarrow \mathcal{B} \wedge not\,\mathcal{C}$ and $\mathcal{A}' \leftarrow \mathcal{B}' \wedge not\,\mathcal{C}'$ in P_1 such that $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{B}' \subseteq \mathcal{B}$, $\mathcal{C}' \subseteq \mathcal{C}$, and $P_2 = P_1 - \{\mathcal{A} \leftarrow \mathcal{B} \wedge not\,\mathcal{C}\}$.

We already required that $not\,A$ should be derivable if A appears in no rule head. But then it should be possible to evaluate the body literal $not\ A$ to true, i.e. to delete $not\ A$ from all rule bodies: this is guaranteed by *Positive Reduction*. As an example, we can replace $p \leftarrow q \wedge not\, r$ by $p \leftarrow q$ if r does not appear in any rule head.

Definition 2.10. (Positive Reduction). A ground program P_2 results from a ground program P_1 by positive reduction ($P_1 \mapsto_P P_2$) iff there is a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\,\mathcal{C}$ in P_1 and $C \in \mathcal{C}$ such that $C \notin \mathit{heads}(P_1)$ and

$$\begin{aligned} P_2 = P_1 &- \{\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\,\mathcal{C}\} \\ &\cup \big\{\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\,(\mathcal{C} - \{C\})\big\}. \end{aligned}$$

Conversely, if the logic program contains $A_1 \vee \cdots \vee A_k \leftarrow \mathit{true}$, at least one of these atoms must be true, so a rule body containing $\mathit{not}\,A_1 \wedge \cdots \wedge \mathit{not}\,A_k$ is surely false, therefore the entire rule is useless, and it should be possible to delete it: this gives us *Negative Reduction*.

Definition 2.11. (Negative Reduction). Let P_1 and P_2 be ground programs. $P_1 \mapsto_N P_2$ iff there are rules $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\,\mathcal{C}$ and $\mathcal{A}' \leftarrow \mathit{true}$ in P_1 such that $\mathcal{A}' \subseteq \mathcal{C}$ and $P_2 = P_1 - \{\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\,\mathcal{C}\}$.

These notions of reduction have been introduced in [Dix92] for normal programs and in [DM94a] for disjunctive programs. It turned out that an application of these principles may reduce the size of a program drastically, because many literals are decided to be true or false.

Let us now introduce names for various interesting combinations of the above elementary transformations:

Definition 2.12. (Combined Transformations)

- Let $\mapsto_{UWT} := \mapsto_U \cup \mapsto_W \cup \mapsto_T$, i.e. the combination of unfolding, weak unfolding, and the elimination of tautologies.
- Let $\mapsto_{MPN} := \mapsto_M \cup \mapsto_P \cup \mapsto_N$, i.e. the combination of the above "reduction" transformations.
- Let $\mapsto_{\mathrm{all}} := \mapsto_{UWT} \cup \mapsto_{MPN}$, i.e. the combination of all the above transformations.
- Let $\mapsto_{\mathrm{dwfs}} := \mapsto_U \cup \mapsto_T \cup \mapsto_M \cup \mapsto_P \cup \mapsto_N$ be the combination of the five transformations which characterize our semantics D-WFS (all the above transformations except weak unfolding which is redundant by Lemma 2.1).
- Finally, let $\mapsto_{\mathrm{mmod}} := \mapsto_U \cup \mapsto_W \cup \mapsto_T \cup \mapsto_M$, i.e. the transformations which do not change minimal models.

Note that our properties make sense for semantics like GCWA and WFS, which are defined only on positive resp. normal programs, because these classes of programs are closed under our transformations.

2.3. Definition of a Disjunctive Well-Founded Semantics

If we now look at the space of all possible abstract semantics, there are of course many semantics allowing the above transformations. However, we will show that there is one uniquely determined *weakest* semantics.

Definition 2.13. (Weaker Semantics). We call a semantics $\mathcal{S}_1$ weaker than a semantics $\mathcal{S}_2$ ($\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$) iff $\mathcal{S}_1(P) \subseteq \mathcal{S}_2(P)$ for all programs P.

Lemma 2.2. The set of semantics ordered by $\sqsubseteq$ is a complete lattice.

PROOF. Obviously, $\sqsubseteq$ is a partial order (we denote in the following with P a program and with Q a pure disjunction):

$$\mathcal{S}_1 \sqsubseteq \mathcal{S}_2 \iff \big\{(P,Q) \mid Q \in \mathcal{S}_1(P)\big\} \subseteq \big\{(P,Q) \mid Q \in \mathcal{S}_2(P)\big\}.$$

The least upper bound and greatest lower bound are also computed as in the standard powerset lattice, namely we take the union resp. intersection of the sets $\mathcal{S}(P)$. It is easy to check that the resulting mapping is indeed a semantics according to our definition. □

We introduce the following notation for the weakest possible semantics (the bottom element of the lattice):

Definition 2.14. (Known Disjunctive Facts). Let *known* be the semantics defined by:

1. For a positive ground disjunction Q:

$$Q \in \mathit{known}(P) \;:\Longleftrightarrow\; \text{there is } \mathcal{A} \subseteq Q \text{ with } \mathcal{A} \leftarrow \mathit{true} \;\in\; \mathit{ground}(P).$$

2. For a negative ground disjunction Q:

$$Q \in \mathit{known}(P) \;:\Longleftrightarrow\; \text{there is } \mathit{not}\ A \;\in\; Q \text{ with } A \notin \mathit{heads}(P).$$

Lemma 2.3. Let SEM be a set of semantics allowing a transformation $\mapsto$. Then $\texttt{lub}(\mathit{SEM})$ *and* $\texttt{glb}(\mathit{SEM})$ *also allow* $\mapsto$.

PROOF. Let P_1 and P_2 be any ground programs with $P_1 \mapsto P_2$. Then we have for all $\mathcal{S} \in \mathit{SEM}$: $\mathcal{S}(P_1) = \mathcal{S}(P_2)$. For the greatest lower bound, we have:

$$\big(\texttt{glb}(\mathit{SEM})\big)(P_1) = \bigcap_{\mathcal{S}\in\mathit{SEM}} \big(\mathcal{S}(P_1)\big) = \bigcap_{\mathcal{S}\in\mathit{SEM}} \big(\mathcal{S}(P_2)\big) = \big(\texttt{glb}(\mathit{SEM})\big)(P_2).$$

The same works for $\texttt{lub}$ (with $\bigcup$ instead of $\bigcap$). □

Theorem 2.1 (Weakest Semantics). For any transformation $\mapsto$ there is a unique weakest semantics $\mathcal{S}_{\mapsto}$ allowing the transformation $\mapsto$.

PROOF. This is trivial: we simply take the greatest lower bound of all semantics allowing the transformation $\mapsto$. By Lemma 2.3 it also allows $\mapsto$. □ However, there is also a more constructive proof. Let us denote by $\equiv$ the reflexive, symmetric, and transitive closure of the transformation $\mapsto$. Then we define:

$$\mathcal{S}_{\mapsto}(P) := \bigcup_{P' \equiv \mathit{ground}(P)} \mathit{known}(P').$$

From this construction it is clear that $\mathcal{S}_{\mapsto}$ is invariant under the transformations and that any other semantics $\mathcal{S}$ with these properties must at least derive the same disjunctions, so $\mathcal{S}_{\mapsto} \sqsubseteq \mathcal{S}$.

Definition 2.15. (D-WFS). The semantics D-WFS is defined as the weakest semantics allowing unfolding, elimination of tautologies and nonminimal rules, and positive and negative reduction, i.e. the weakest semantics allowing the transformation $\mapsto_{\mathrm{dwfs}}$.

Note that at this point it is still possible that the semantics is inconsistent, i.e. that for some P and $A_1, \dots, A_n$, it is possible that $A_1 \vee \cdots \vee A_n \in$ D-WFS(P), and *not* $A_i \in$ D-WFS(P) for $i = 1, \dots, n$. This is no particular failure of this way to define a semantics, because other prominent semantics such as STABLE can be inconsistent, too. It would simply mean that the required properties are too strong. However, as will become clear from the characterization in the next section, our semantics D-WFS is always consistent. In fact, it is a rather weak semantics.

We note that the restriction to a *function-free* and *finite* signature is essential in this section. Our reductions do not allow us to "unfold infinite loops". Let us consider the program consisting of the single rule

$p(X) \leftarrow p\big(f(X)\big)$

(or the equivalent infinite ground program). If D-WFS were defined for such programs as above, it would not be able to derive *not* $p(t)$ from this program. So we need that the ground instantiation of all considered programs is finite.

However, as already mentioned before Definition 1, the non-constructive loop-detection rule of [DS98] allows to derive *not* $p(t)$.

2.4. A Framework for Defining Semantics

Let us conclude this section by noting that the proposed method for defining and characterizing semantics is in fact a very general one. The idea is to

1. Define a space of candidate semantics as mappings from logic programs to some "semantic domain", satisfying certain minimal requirements. These semantics should be ordered by a "weaker" relation $\sqsubseteq$.

2. Select a number of good properties the semantics should have. It is important that these properties are inherited to the greatest lower bound of semantics satisfying them. This is automatically satisfied for properties of the form considered here (that certain program transformations do not change the semantics) if the relation $\sqsubseteq$ is derived from some order on the "semantic domain" by pointwise comparison for all programs.

3. Look at the $\sqsubseteq$-smallest semantics having the properties.

For instance, in [BD97], we have defined a semantics modeltheoretically as a mapping which assigns to every logic program P a set of three-valued Herbrand models of P (subject to the condition that atoms not occurring in P are interpreted as false). So the "semantic domain" consists of sets of three-valued Herbrand interpretations. A semantics is stronger iff it allows less models, so we define

$\mathcal{S}_1 \sqsubseteq \mathcal{S}_2 \iff$ for all P: $\mathcal{S}_1(P) \supseteq \mathcal{S}_2(P)$.

In order to show that not only D-WFS can be captured in our framework, we need to introduce two further transformation rules.

Definition 2.16. (Elimination of Contradictions, Supraclassicality). A ground program P_2 results from a ground program P_1 by elimination of contradictions ($P_1 \mapsto_T P_2$) iff there is a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}$ in P_1 such that $\mathcal{B} \cap \mathcal{C} \neq \emptyset$ and $P_2 = P_1 - \{\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}\}$.

A ground program P_2 results from a ground program P_1 by applying supraclassicality ($P_1 \mapsto_T P_2$) iff there is an atom A such that $P_1 \models A$ and $P_2 = P_1 \cup \{A \leftarrow\}$. (Here $P_1 \models A$ just means *classical entailment*: P_1 is viewed as a propositional theory and A as an atom therein.

Elimination of contradictions just means to eliminate rules with contradicting bodies, like $p \vee q \leftarrow r, s, \mathit{not}\ r, \mathit{not}\ q$. Supraclassicality means that whenever an atom A follows classically from the program (viewed as a classical theory), then this atom can be safely added to the program. This formalizes the property that the semantics should be at least as strong as classical logic.

We are now able to state the following theorems, the proofs of which can be found in [BD97, Theorem 4.1], [BD97, Theorem 4.4] and [Dix95a, Theorem 4.9].

Theorem 2.2. (GCWA, STABLE and WFS_C as weakest semantics)

1. *GCWA is the weakest semantics for positive disjunctive programs satisfying positive and negative Reduction, GPPE and Elimination of Tautologies.*
2. *STABLE is the weakest semantics for arbitrary disjunctive programs satisfying positive and negative Reduction, GPPE, Elimination of Tautologies and Elimination of Contradictions.*
3. *WFS_C is the weakest semantics for non-disjunctive programs satisfying positive and negative Reduction, GPPE, Elimination of Tautologies and Supraclassicality.*

In all the above semantics, Elimination of Tautologies was an important ingredient. But we can also capture semantics where this transformation does not hold. In fact, we need only two very special cases of the GPPE rule: *Success* and *Failure*.

Definition 2.17. (Success and Failure) The success-transformation can be applied to a program P whenever there is a rule $A \leftarrow$ in P. Success then removes a positive occurrence of A in the body of another rule.

Dually, the failure-transformation can be applied to a program P whenever there is a an atom A which does not occur in any head of a rule. Failure then removes a rule which contains A positively in its body.

The following theorem is contained in [DFN98, Theorem 5.3].

Theorem 2.3. ($comp_3$ (resp. $\mathrm{lfp}(\Phi_P)$*) as weakest semantics)* FITTING'S *semantics $comp_3$ (which is defined as* $\mathrm{lfp}(\Phi_P)$*) is the weakest semantics for non-disjunctive programs satisfying Success, Failure, positive and negative Reduction.*

The following table (from [BD95b]) gives an overview which semantics allow which transformations. The entry "•" means that the corresponding property holds, "—" means that it does not hold. We have not included an entry for positive or negative Reduction, because these conditions hold for all semantics listed below.

Properties of Logic-Programming Semantics						
Semantics	Domain	Taut.	GPPE	Nonmin.	Contra	Supra
Clark's comp [Cla78]	Nondis.	—	•	•	•	•
Fitting's comp_3 [Fit85]	Nondis.	—	•	•	—	—
GCWA [Min82]	Pos.	•	•	•	•	•
WGCWA [RLM89][2]	Pos.	—	•	—	—	—
Stable [GL91, Prz91b]	Dis.	•	•	•	•	•
WFS [VGRS91]	Nondis.	•	•	•	—	—
WFS_C [Sch92]	Nondis.	•	•	•	—	•
Strong WFS [Ros89]	Dis.	—	—	—	—	—
Static [Prz95]	Dis.	•	•	•	—	—

We strongly conjecture that other interesting results can be achieved by modifying the basic requirements on the semantics and possibly by looking at other properties. Many of the results and techniques of this paper will remain applicable, because they only refer to program transformations.

3. BOTTOM-UP QUERY EVALUATION

In this section we explain how to compute a normal form of the given program, called the residual program. In contrast to the last section, we no longer work on instantiated programs. However, we use the allowedness-condition to bind the variables and avoid floundering. Our approach is based on the notion of "conditional facts", as developed by Bry in [Bry89, Bry90] and Dung/Kanchansut in [DK89b, DK89a] (both for the non-disjunctive case). The idea is to delay the evaluation of negative body literals, and to attach them as conditions to the derived (disjunctive) facts.

Definition 3.1. (Conditional Fact). A conditional fact is a rule without positive body literals, i.e. it is of the form $A_1 \vee \cdots \vee A_k \leftarrow \mathit{not}\, C_1 \wedge \cdots \wedge \mathit{not}\, C_m$, where the A_i and the C_j are ground atoms ($k \geq 1$, $m \geq 0$).

The necessity that the A_i and C_j are ground automatically follows from the allowedness restriction.

3.1. Computation of Derived Conditional Facts

The usual bottom-up fixpoint computation is also possible with conditional facts: in the non-disjunctive case, one can simply store the conditions of a fact in an additional set-valued argument. Derived facts get the union of the conditions of the facts matched with the body literals (plus the corresponding instances of the negative body literals of the rule itself). This is demonstrated in the following example:

[2] WGCWA is equivalent to the semantics introduced by Ross/Topor in [RT88].

$$
\begin{array}{ccccccccc}
p(a) & \leftarrow & & & \mathit{not}\, s(b) & \wedge & \mathit{not}\, r(b). \\
\uparrow & & & & \uparrow & & \uparrow \\
p(X) & \leftarrow & q_1(X) & \wedge & q_2(X,Y) & \wedge & \mathit{not}\, r(Y). \\
& & \uparrow & & \uparrow & & \\
& & q_1(a) & & q_2(a,b) \leftarrow \mathit{not}\, s(b) & &
\end{array}
$$

In the disjunctive case, one applies the "hyperresolution" operator by adding to every fact also a disjunctive context [CL73, MR90, Bra94]. An implementation with database techniques has been suggested in [BL92]. Formally, the immediate consequence operator is generalized to conditional facts as follows:

Definition 3.2. (Immediate Consequences with Conditions). For a set Γ of conditional facts we define:

$$T_P(\Gamma) := \Big\{ \Big(\mathcal{A}_0 \cup \bigcup_{i=1}^{m} (\mathcal{A}_i - \{B_i\})\Big) \leftarrow \mathit{not}\, \Big(\mathcal{C}_0 \cup \bigcup_{i=1}^{m} \mathcal{C}_i\Big) \;\Big|\; \text{there are a ground instance } \mathcal{A}_0 \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \mathit{not}\, \mathcal{C}_0 \text{ of a rule in } P \text{ and cond. facts } \mathcal{A}_i \leftarrow \mathit{not}\, \mathcal{C}_i \in \Gamma \text{ with } B_i \in \mathcal{A}_i \ (i = 1, \ldots, m) \Big\}.$$

Example 3.1. The following example illustrates the idea. From

$\mathit{winning}(X) \leftarrow \mathit{move}(X,Y) \wedge \mathit{not}\ \mathit{winning}(Y)$

and the fact $\mathit{move}(a,b)$, we derive the conditional fact

$\mathit{winning}(a) \leftarrow \mathit{not}\ \mathit{winning}(b)$.

If we apply the rule

$\mathit{goodstate}(X) \leftarrow \mathit{winning}(X) \wedge \mathit{not}\ \mathit{excluded}(X)$

to the above conditional fact, we get the rule

$\mathit{goodstate}(a) \leftarrow \mathit{not}\ \mathit{excluded}(a) \wedge \mathit{not}\ \mathit{winning}(b)$.

Lemma 3.1. The operator T_P is monotonic and even continuous in the lattice of sets of conditional facts ordered by $\subseteq$.

Proof. The proof is the same as in the standard case. □

So we can compute the smallest fixpoint of T_P as usual: we start with $\Gamma_0 := \emptyset$ and then iterate $\Gamma_i := T_P(\Gamma_{i-1})$ until nothing changes. This must happen because there are only finitely many predicates and constants to build ground atoms occurring in conditional facts, and there are only finitely many subsets of all these atoms (corresponding to head and body).

The operator T_P and the idea of using conditional facts already appeared in work of Dung/Kanchansut ([DK89a]) and Bry ([Bry89, Bry90]) for *nondisjunctive* programs. In [HY91] a somewhat related approach (again for normal programs) was defined: a semantics was reduced to programs containing only negative literals in their rule-bodies.

3.2. Application of Reductions

So we now have a logic program with rules of a very particular kind, namely containing no positive body literals. The next step of the proposed query evaluation algorithm is to simplify it by means of positive and negative reduction, and the elimination of nonminimal rules. This leads to the following reduction operator on sets of conditional facts (a generalization of reductions studied in [Bry90]):

Definition 3.3. (Reductions of Conditional Facts). For a set Γ of conditional facts we define:

$$\begin{aligned} R(\Gamma) := \{\mathcal{A} \leftarrow \mathit{not}\ \big(\mathcal{C} \cap \mathit{heads}(\Gamma)\big) \mid\ & \mathcal{A} \leftarrow \mathit{not}\ \mathcal{C} \in \Gamma, \text{ and} \\ & (1) \text{ there is no } \mathcal{A}' \leftarrow \mathit{true} \in \Gamma \text{ with } \mathcal{A}' \subseteq \mathcal{C}, \\ & (2) \text{ there is no } \mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}' \in \Gamma \text{ with } \mathcal{A}' \subseteq \mathcal{A}, \\ & \quad \mathcal{C}' \subseteq \mathcal{C} \text{ where at least one } \subseteq \text{ is proper}\}. \end{aligned}$$

We again iterate this operator until nothing changes. Since the total number of atoms occurring in Γ is reduced in each step, this process must come to an end.

For instance, consider the program

$p \leftarrow \mathit{not}\ q.$
$q \leftarrow \mathit{not}\ r.$

Then the first application of R evaluates r to false, so we get

$p\ \leftarrow \mathit{not}\ q.$
$q.$

But now R is applicable again, and deletes the first rule (because q is obviously true), so we end up with

$q.$

Definition 3.4. (Residual Program). Let P be a logic program, $\Gamma_0 := \mathrm{lfp}(T_P)$, $\Gamma_i := R(\Gamma_{i-1})$, and $n \in \mathbb{N}$ with $\Gamma_n = \Gamma_{n-1}$. Then we call $\mathit{res}(P) := \Gamma_n$ the residual program of P.

Although our operators T_P, R resemble to Fitting's operator Φ_P for nondisjunctive programs they are used differently. Not only do they operate on conditional facts, they are also applied in such a way that for programs like $p \leftarrow p$ we get the empty program. Therefore $\mathit{not}\, p$ is derivable whereas Fitting's operator would leave p undefined.

The notion of *residual program* now gives us a straightforward way to define a semantics from an implementation point of view (we just described a constructive bottom-up evaluation procedure):

Definition 3.5. (Computed Semantics). We define a mapping $\mathcal{S}_{res}$ from logic programs to sets of pure ground disjunctions by

$\mathcal{S}_{res}(P) := \mathit{known}\big(\mathit{res}(P)\big).$

Of course, the so defined semantics will turn out to be our semantics D-WFS. This is subject of Section 4. But first let us verify that the mapping $\mathcal{S}_{res}$ is indeed a semantics:

Lemma 3.2. The mapping $\mathcal{S}_{res}$ satisfies the conditions of a semantics.

Lemma 3.3. For all sets Γ and Γ' of conditional facts:

$\Gamma \mapsto_{MPN} \Gamma' \implies res(\Gamma) = res(\Gamma')$.

I.e. the result of the reduction phase is invariant under elimination of nonminimal rules as well as positive and negative reduction.

By the way, since $\Gamma \mapsto^*_{MPN} res(\Gamma)$, this also implies the confluence of $\mapsto_{MPN}$ among sets of conditional facts. For implementations it is very important to know that the reductions can in fact be performed in any sequence. We have discussed data structures for this in [BD95a].

3.3. Alternative Characterization of Reduction Phase

It has been criticized that the reduction phase is not done by a standard fixpoint construction, and therefore does not generalize to the infinite case. In contrast, our T_P operator has all the nice properties of the standard operator, so $\mathrm{lfp}(T_P)$ is also defined for programs with an infinite ground instantiation. We solve this problem by giving an alternative construction for $\mathcal{S}_{res}$ and the residual program. It is based on the insight that for the reductions of our operator R we need only the positive disjunctions we know already for sure, and the atoms which are no longer possibly true.

Definition 3.6. (Reduct). Let Γ be a set of conditional facts. Given a set Δ of pure ground disjunctions, we write Δ^+ for the positive disjunctions in Δ, and Δ^- for the negative ground literals contained (as one-element disjunctions) in Δ. The reduct of Γ with respect to Δ is defined as:

$$\Gamma/\Delta := \Big\{\mathcal{A} \leftarrow \mathit{not}\,(\mathcal{C} - \Delta^-) \;\Big|\; \begin{array}{l} \mathcal{A} \leftarrow \mathit{not}\,\mathcal{C} \;\in\; \Gamma \text{ and} \\ (1) \text{ there is no } \mathcal{A}' \in \Delta^+ \text{ with } \mathcal{A}' \subseteq \mathcal{C} \\ (2) \text{ there is no } \mathcal{A}' \leftarrow \mathit{not}\,\mathcal{C}' \text{ in } \Gamma \text{ with} \\ \quad \mathcal{A}' \subseteq \mathcal{A} \text{ and } (\mathcal{C}' - \Delta^-) \subseteq (\mathcal{C} - \Delta^-) \\ \quad \text{(where at least one} \subsetneq \text{is proper)}\end{array}\Big\}.$$

Definition 3.7. (Derivation of Pure Disjunctions). For every set Γ of conditional facts, we define an operator D_Γ which derives pure ground disjunctions from pure ground disjunctions:

$D_\Gamma(\Delta) := \mathit{known}(\Gamma/\Delta)$.

Lemma 3.4. The operator D_Γ is monotonic (in the standard powerset lattice).

The following theorem shows that this construction is equivalent to the above definition of the residual program and $\mathcal{S}_{res}$:

Theorem 3.1. Let P be any (finite) logic program, and $\Gamma := \mathrm{lfp}(T_P)$. Then:

$\mathcal{S}_{res}(P) = \mathrm{lfp}(D_\Gamma)$ *and* $res(P) = \Gamma/\,\mathrm{lfp}(D_\Gamma)$.

4. MAIN RESULTS

In Section 2, we introduced a framework to define semantics in an abstract way as mappings from programs into semantic domains. In particular we studied elementary program transformations in this setting. In Section 3 we described a constructive bottom up procedure to compute a normal form of a program. We show in this section that both approaches are closely related. This proves that the computed semantics $\mathcal{S}_{res}$ is identical to our semantics D-WFS, so we have soundness and completeness of our algorithm.

However, our results go much further, because they apply to *any* semantics allowing certain transformations. We also show a nice relation of a subset of our transformations to minimal models and consider the possibility of "lifting" definitions of semantics from residual programs to all programs.

4.1. Soundness of Residual Program Computation

If we only know that a semantics $\mathcal{S}$ allows unfolding (GPPE) and elimination of tautologies, we already can apply the first part of our algorithm:

Theorem 4.1 (Conditional Facts can be Constructed by Transformations). The set of derived conditional facts can be constructed from the ground instantiation of a program by applying only unfolding, weak unfolding, and the elimination of tautologies. I.e. $\mathit{ground}(P) \mapsto^{*}_{UWT} \mathrm{lfp}(T_P)$ *holds for all programs* P.

Since by Lemma 2.1 a semantics which allows unfolding and the elimination of tautologies automatically also allows weak unfolding, we get the following corollary:

Corollary 4.1 (Equivalence of Implied Conditional Facts). If a semantics $\mathcal{S}$ *allows unfolding (GPPE) and the elimination of tautologies, then* $\mathcal{S}(P) = \mathcal{S}\big(\mathrm{lfp}(T_P)\big)$ *holds for all programs* P.

In other words, if a semantics allows unfolding and the elimination of tautologies, then it also allows the transformation $\mathit{ground}(P) \mapsto \mathrm{lfp}(T_P)$.

So under these weak conditions on the semantics, we can already apply the computation of the implied conditional facts as a preprocessing step. Afterwards, we still need some algorithm to compute the semantics, but instead of arbitrary programs, it has to handle only ground programs without positive body literals.

Of course, if the semantics $\mathcal{S}$ allows in addition the elimination of nonminimal rules and positive and negative reduction, then also the application of the reduction operator R does not change the semantics of the program. So we get:

Corollary 4.2 (Transformations Yield Residual Program). $\mathit{ground}(P) \mapsto^{*}_{\mathrm{all}} \mathit{res}(P)$ *holds for all programs* P*. I.e. the residual program* $\mathit{res}(P)$ *can be reached from the ground instantiation of* P *by applying the elementary transformations introduced in Section 2.2.*

Corollary 4.3 (Equivalence of Residual Program). If a semantics $\mathcal{S}$ *allows unfolding (GPPE), elimination of tautologies and of nonminimal rules, as well as positive and negative reduction, then for all programs* P*:*

$\mathcal{S}(P) = \mathcal{S}\big(\mathit{res}(P)\big)$.

Since our semantics D-WFS allows the above five transformations, we have

$$\text{D-WFS}(P) = \text{D-WFS}\big(res(P)\big).$$

Then we get $\text{D-WFS}\big(res(P)\big) \supseteq known\big(res(P)\big)$ by definition of semantics. Together, this implies $\text{D-WFS}(P) \supseteq \mathcal{S}_{res}(P)$, i.e. the soundness of our algorithm. We will show the other direction "$\subseteq$" with Theorem 4.3.

However, the above result is not only useful for showing the soundness of our computation of D-WFS. For instance, consider the stable model semantics, which also allows the above five transformations. With the same argument, we get that the original program P and the residual program $res(P)$ have the same stable models. To be precise, taking stable models is not quite a semantics in the above sense. However, we can still use Theorem 4.1 and Corollary 4.2. So the above results are independent of the exact definition of a semantics, it can be any mapping from logic programs into some semantic domain.

Of course, we still have to compute the stable models of the residual program. But in the residual program, most ground literals are already decided to be true or false — any element of $known\big(res(P)\big)$ is also a valid consequence of the stable semantics. Only the few hard cases, the "islands of complexity" remain. For instance, if P is a non-disjunctive stratified program, then the residual program is simply a set of facts, so $known\big(res(P)\big)$ is already the unique stable model (the perfect model in this case). So many queries can already be answered on the basis of the residual program. But even if we want the complete stable models, there are algorithms (like [BNNS94]) which are good for treating the most general case, but seem to be inefficient for simpler cases. Such algorithms can profit from our computation of the residual program as a preprocessing phase. In [BD95a], we also proposed an algorithm for doing the remaining work in the computation of stable models (based on a disjunctive extenstion of CLARK's completion).

Another application is in our prototype implementation of the STATIC semantics [BDP96]: it also consists of the computation of the residual program plus an algorithm for treating the few remaining hard cases. Again, the general algorithm would be too inefficient if applied directly to the original program, but it is good enough for evaluating the few negative body literals remaining in the residual program. Also, the general algorithm is simplified, because it does not have to treat positive body literals or variables.

4.2. Relation to Minimal Models

For modeltheoretically defined semantics like STABLE or STATIC, it is useful to consider the relation of our transformations to minimal models. The following interesting characterization also stresses again the importance of our transformations. We will show that a semantics allows unfolding (GPPE), the elimination of tautologies, and the elimination of nonminimal rules if and only if it looks exactly at the minimal models of the programs, and not at the syntax of the programs. I.e. we require that $\mathcal{S}(P_1) = \mathcal{S}(P_2)$ for all programs P_1 and P_2 having the same minimal models. To be precise, let us first define what we mean by a minimal model:

Definition 4.1. (Model). A model of a logic program P is any set I of ground literals such that for every ground instance

$A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_m \wedge not\, C_1 \wedge \cdots \wedge not\, C_n,$

of a rule in P:

$\{B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n\} \subseteq I \implies \{A_1, \ldots, A_k\} \cap I \neq \emptyset.$

So models treat A and $not\, A$ as two distinct and unrelated atoms. It is possible that they are both true or both false. Models correspond to classical models if we introduce for every predicate p a new predicate not_p and replace everywhere $not\, p(\ldots)$ by $not\_p(\ldots)$. This trick was already used in [Prz91a] and we refer to [BDNP98] for a more detailed investigation.

Definition 4.2. (Minimal Model). A model I of a program P is minimal if there is no model I' of P such that I and I' contain the same set of negative literals, but the positive literals contained in I' are a strict subset of those in I.

This definition of minimal model is inspired by the construction of static expansions [Prz95], where the "objective literals" are minimized while the interpretation of the "belief literals" (used for negations) is kept fixed. It is a very useful notion. For instance, a stable model can be defined as a minimal model which is total and consistent, i.e. contains either A or $not\, A$ for every ground atom A (see [BD97, Theorem 4.3]).

With the above notion of a model, we get the following useful semantical description of the set of derived conditional facts:

Lemma 4.1. For every program P, the operator T_P is correct and complete:

1. *Every conditional fact contained in* $\mathrm{lfp}(T_P)$ *is a logical consequence of P.*
2. $\mathrm{lfp}(T_P)$ *contains all conditional facts $\mathcal{A} \leftarrow not\, \mathcal{C}$, which are logical consequences of P and which are minimal (where "minimal" means that there is no $\mathcal{A}' \leftarrow not\, \mathcal{C}'$, which is also a logical consequence and satisfies $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$, where at least one "$\subseteq$" is proper).*

Now the transformations unfolding (GPPE), elimination of tautologies and elimination of nonminimal rules do not change the set of minimal models. This is trivial for the elimination of tautologies and nonminimal rules, because they are logical equivalence transformations, i.e. they do not change the set of models.

Lemma 4.2. Let $P_1 \mapsto_U P_2$. Then P_1 and P_2 have the same minimal models.

This gives us the direction: if $\mathcal{S}(P_1) = \mathcal{S}(P_2)$ for all programs P_1 and P_2 having the same minimal models, then $\mathcal{S}$ allows these three transformations.

The other direction is proven by showing that for every pair of distinct ground programs P_1 and P_2, which are irreducible with respect to these transformations, there is an interpretation I which is a minimal model of one of the two programs, but not of the other.

Theorem 4.2 (Semantics Looking only at Minimal Models). A semantics $\mathcal{S}$ allows unfolding (GPPE), elimination of tautologies, and elimination of nonminimal *rules if and only if it satisfies the following condition:*

$\mathcal{S}(P_1) = \mathcal{S}(P_2)$ *for all programs* P_1 *and* P_2 *which have the same set of minimal models.*

In addition, also the confluence of the rewriting system consisting of unfolding, weak unfolding, elimination of tautologies and elimination of nonminimal rules follows immediately from the following facts:

1. These transformations do not change the set of minimal models.

2. The rewriting system is terminating: from every ground program P, we get an irreducible program P' by first computing the set of implied conditional facts (see Theorem 4.1) and then applying the elimination of nonminimal rules as long as possible.

3. Different irreducible programs have different sets of minimal models.

So from every ground program P, it is always possible to reach an irreducible program by applying these transformations, but it is not possible to get to two different irreducible programs. Questions of confluence and termination have been further investigated in [BD98a].

4.3. Semantics Defined on the Residual Program

Now let us return to the computation of D-WFS, which is much simpler than STABLE or STATIC, because it can be directly answered from the residual program. The completeness of our algorithm is still missing, i.e. D-WFS$(P) \subseteq \mathcal{S}_{res}(P)$. In order to prove it, we need the following theorem which is interesting in itself:

Theorem 4.3 (Invariance of Residual Program). For all ground programs P_1, P_2*:*

$P_1 \mapsto_{\text{all}} P_2 \implies \mathit{res}(P_1) = \mathit{res}(P_2)$.

I.e. the residual program is invariant under the elementary program transformations considered in this paper.

This has the following astonishing corollary:

Corollary 4.4 (Guaranteed Properties). Let $\mathcal{S}$ *satisfy* $\mathcal{S}(P) = \mathcal{S}\big(\mathit{res}(P)\big)$ *for all* P*. Then* $\mathcal{S}$ *allows unfolding (GPPE), the elimination of tautologies and of nonminimal rules, as well as positive and negative reduction.*

PROOF. Let $P_1 \mapsto_{\text{all}} P_2$. Then $\mathcal{S}(P_1) = \mathcal{S}\big(\mathit{res}(P_1)\big) = \mathcal{S}\big(\mathit{res}(P_2)\big) = \mathcal{S}(P_2)$. □

This means that when we define a semantics only for residual programs, and extend it to arbitrary programs via $\mathcal{S}(P) := \mathcal{S}\big(\mathit{res}(P)\big)$, then $\mathcal{S}$ will automatically satisfy all our properties. Of course, we have to ensure that the so defined mapping $\mathcal{S}$ is really a semantics. For our notion of semantics, this is simple. But for a modeltheoretic notion of a semantics, it is important to note that not all models of $\mathit{res}(P)$ are also models of P. Supported models of $\mathit{res}(P)$, however, are also models of P. The same holds for the normal models defined below (see Lemma 4.3). We do not know yet a necessary and sufficient condition for the models of $\mathit{res}(P)$ to be also models of P (resp. every program with the given residual program).

Example 4.1. Consider the following P:

$p \leftarrow q.$
$q \vee r.$

Here, $res(P) = \{p \vee r,\ q \vee r\}$. Now $I := \{q, r\}$ is a Herbrand model of $res(P)$ with $I \not\models P$. $\square$

From Corollary 4.4 it follows immediately that our semantics $\mathcal{S}_{res}$ has the required properties. Since D-WFS is the weakest semantics with these properties, we get D-WFS$(P) \subseteq \mathcal{S}_{res}(P)$. This is the missing completeness, so together we have D-WFS$(P) = \mathcal{S}_{res}(P)$, in other words:

Corollary 4.5 (Computation of D-WFS).

$$Q \in \text{D-WFS}(P) \iff \begin{array}{l} \textit{there is } \mathcal{A} \subseteq Q \textit{ with } \mathcal{A} \leftarrow \textit{true} \in \textit{res}(P) \quad \textit{or} \\ \textit{there is not } A \in Q \textit{ and } A \notin \textit{heads}\big(\textit{res}(P)\big). \end{array}$$

Note that Corollary 4.4 is the exact converse of Corollary 4.3. They can be formulated together as follows:

$\mathcal{S}$ allows unfolding (GPPE), positive and negative Reduction,
Elimination of Tautologies and of Nonminimal Rules

if and only if

$\textit{ground}(P) \mapsto \textit{res}(P)$ is a $\mathcal{S}$-equivalence transformation (for all P).

Finally note that Theorem 4.3 together with Corollary 4.2 immediately gives us the confluence of $\mapsto_{\text{all}}$ [BD98a].

4.4. Closure under Logical Consequences

We still have to prove that D-WFS is closed under logical consequences. In order to do this, it will be useful to have some restrictions on models of the residual program which ensure that they will be also models of the original program. We already mentioned that this holds for supported models. However, residual programs such as $p \leftarrow \textit{not}\, p$ do not have supported models. The following kind of model always exists:

Definition 4.3. (Normal Model). An interpretation I is a normal model of a program P iff

1. I is a minimal model of P,
2. $I \models \textit{not}\, A$ for every atom A with $A \notin \textit{heads}(P)$,
3. $I \not\models \textit{not}\, A_1 \wedge \cdots \wedge \textit{not}\, A_n$ for every disjunctive fact $A_1 \vee \cdots \vee A_n \leftarrow \textit{true}$ contained in P.

Lemma 4.3. If I is a normal model of $res(P)$, it is also a normal model of P.

Theorem 4.4 (Closure under Logical Consequences). For every program P and pure ground disjunction Q:

$P \cup \textit{D-WFS}(P) \mathrel{|\!\sim} Q \implies Q \in \textit{D-WFS}(P)$.

Of course, implication $\mathrel{|\!\sim}$ is meant here with respect to the above notion of a model (which treats A and *not* A as unrelated). However, the interpretations constructed in the proof are at least consistent.

4.5. Cumulativity for $res(P)$

In this section we are showing a very interesting property of our residual program, which has nice computational implications for computing stable models. Namely if a disjunction $A_1 \vee \ldots \vee A_n$ follows from D-WFS(P), then

the stable models of P and of $P \cup \{A_1 \vee \ldots \vee A_n\}$ are identical.

This makes it possible

- to use D-WFS as a first step in the computation of stable models,

namely by adding to P the disjunctions true in D-WFS(P). The former property has been first noticed for the non-disjunctive case by Schlipf (see[Dix95b, Theorem 3.7]). In particular, for the non-disjunctive case this means that it always pays off to first compute the wellfounded semantics (which can be done in polynomial time, as opposed to computing stable models, which is one level higher in the polynomial hierarchy) and to use the results to simplify the program. In fact, this is one of the main techniques employed in [NS96] which is currently the fastest method to compute stable models (see also [DFN98]).

Lemma 4.4. (Adding Conditional Facts)

1. *$\mathcal{A} \leftarrow not\, \mathcal{C} \in \mathrm{lfp}(T_P)$ implies $\mathrm{lfp}(T_P) = \mathrm{lfp}(T_{P \cup \{\mathcal{A} \leftarrow not\, \mathcal{C}\}})$.*
2. *If in addition $\mathcal{C} \cap heads(res(P)) = \emptyset$, then*

$$R(\mathrm{lfp}(T_{P \cup \{\mathcal{A} \leftarrow not\, \mathcal{C}\}})) = R(\mathrm{lfp}(T_{P \cup \{\mathcal{A} \leftarrow\}})).$$

The proof is obvious from the definition of $\mathrm{lfp}(T_P)$.

Corollary 4.6. (Invariance of $res(P)$ for D-WFS-Cumulativity) Let $\mathcal{A}$ be contained in the semantics of D-WFS(P). Then

$res(P) = res(P \cup \{\mathcal{A}\})$.

Proof. If $\mathcal{A} \in$ D-WFS(P) then there is a $\mathcal{C}$ such that

1. $\mathcal{A} \leftarrow not\, \mathcal{C} \in \mathrm{lfp}(T_P)$,
2. $\mathcal{A} \cap \mathcal{C} = \emptyset$,
3. $\mathcal{C} \cap heads(res(P)) = \emptyset$.

Using the lemma above, we have

$\mathrm{lfp}(T_P) = \mathrm{lfp}(T_{P \cup \{\mathcal{A} \leftarrow not\, \mathcal{C}\}})$.

Therefore

$$res(P) = res(P \cup \{\mathcal{A} \leftarrow not\, \mathcal{C}\}) = R^{\omega}(\mathrm{lfp}(T_{P\cup\{\mathcal{A}\leftarrow not\, \mathcal{C}\}})).$$

But since

$$R(\mathrm{lfp}(T_{P\cup\{\mathcal{A}\leftarrow not\, \mathcal{C}\}})) = R(\mathrm{lfp}(T_{P\cup\{\mathcal{A}\leftarrow\}}))$$

we have

$$R^{\omega}(\mathrm{lfp}(T_{P\cup\{\mathcal{A}\leftarrow not\, \mathcal{C}\}})) = R^{\omega}(\mathrm{lfp}(T_{P\cup\{\mathcal{A}\leftarrow\}})) = res(P \cup \mathcal{A}). \qquad \square$$

Since the residual program is a sound transformation for the disjunctive stable models semantics, we get a generalization of a theorem of SCHLIPF which was formulated for nondisjunctive programs ([Dix95b, Theorem 3.7]).

Theorem 4.5. (Restricted Cumulativity for STABLE) Let $\mathcal{A}$ be contained in the semantics of D-WFS(P). Then the programs P and $P \cup \{\mathcal{A}\}$ have identical stable models.

Note that strengthening the assumption *"$\mathcal{A}$ contained in D-WFS(P)"* to

"$\mathcal{A}$ contained in STABLE(P)"

is not possible: this would be exactly the cumulativity condition which does not hold for STABLE([Dix95a]).

5. COMPARISON WITH OTHER APPROACHES

In this section, we compare our semantics D-WFS to the standard well-founded semantics for non-disjunctive programs (Subsection 5.1), to the disjunctive stable model semantics (Subsection 5.2), to the static semantics (Subsection 5.3), and to the GCWA (defined for positive disjunctive programs) (Subsection 5.4). Note that the D-WFS semantics is defined (via grounding) for arbitrary programs. The restriction to allowed Datalog programs applies only to the procedural part. It turns out that

1. D-WFS agrees with WFS and GCWA on the restricted classes of programs for which these semantics are defined.

2. D-WFS is strictly weaker than both the disjunctive stable semantics and the static semantics.

This means that D-WFS at least does not allow to derive any unsensible conclusions.

From the fact that D-WFS generalizes the well-founded semantics, we can also conclude a nice characterization of the standard well-founded semantics. We believe that this characterization is an important contribution of this paper. Furthermore, it gives us a very simple algorithm to compute the well-founded model via the residual program. The computational implications are considered further in Section 6.

5.1. Relation to Well-Founded Semantics

First, it is important to note again that the well-founded semantics really allows all of our transformations:

Theorem 5.1 (WFS allows our Transformations). For non-disjunctive ground programs P_1 and P_2, if $P_1 \mapsto_{\mathrm{all}} P_2$, then P_1 and P_2 have the same well-founded model.

Some of these properties have already been proven in the literature, in particular unfolding has been considered in several papers, see [AD95]. However, we get the possibility of unfolding here immediately from Theorem 4.2, so it is worth mentioning.

Once we know that the WFS has our properties, Corollary 4.3 tells us that the well-founded model of the residual program is equal to the well-founded model of the original program. This result is already known [AD95], but again, we get it immediately in our framework. For residual programs, due to their very simple structure, it is easy to prove that our D-WFS agrees with the standard well-founded semantics. So our semantics D-WFS is indeed a generalization of the well-founded semantics to disjunctive programs.

Theorem 5.2 (D-WFS extends WFS). Let P be a non-disjunctive program and Q be a (positive or negative) ground literal. Then we have:

$Q \in \textit{D-WFS}(P) \iff Q \in \textit{WFS}(P)$.

But this gives us a nice characterization of the standard well-founded semantics:

Corollary 5.1 (Characterization of WFS). WFS is the weakest semantics (for non-disjunctive programs) satisfying GPPE, Elimination of Tautologies, Elimination of Nonminimal Rules and positive and negative Reduction.

In fact, the elimination of nonminimal rules is not even necessary. Of course, the well-founded semantics allows this transformation, but we can compute the well-founded model without it (see Subsection 5.5 below). At this point we can already note the following simple algorithm to compute the well-founded model via the residual program:

Corollary 5.2 (Computation of WFS). Let P be a non-disjunctive program.

1. *For every positive ground literal A:*

$$A \in \textit{WFS}(P) \iff (A \leftarrow \textit{true}) \in \textit{res}(P).$$

2. *For every negative ground literal not A:*

$$\textit{not}\ A \in \textit{WFS}(P) \iff A \notin \textit{heads}\big(\textit{res}(P)\big).$$

5.2. Relation to Disjunctive Stable Semantics

The stable model semantics is also invariant under our transformations:

Theorem 5.3 (STABLE allows our Transformations). For all ground programs P_1 and P_2, if $P_1 \mapsto_{\mathrm{all}} P_2$, then P_1 and P_2 have the same set of stable models.

The stable semantics is in fact a modeltheoretic semantics, which selects some models of the given program, and not directly determines a set of implied pure ground disjunctions. But it can be turned into a semantics as defined in Section 2:

$Q \in \mathrm{STABLE}(P) :\iff I \models Q$ for all stable models I of Q.

Since we know that the stable semantics allows our transformations and we have defined D-WFS as the weakest semantics allowing these transformations, it immediately follows that D-WFS is weaker than (or equal to) STABLE. We can also say that D-WFS is an approximation of STABLE: all consequences of D-WFS are also consequences of STABLE, but STABLE might allow to conclude more (and this is indeed the case, see below):

Corollary 5.3 (D-WFS is weaker than STABLE). Let P be any (disjunctive) logic program. If $Q \in$ D-WFS(P), then Q holds in all stable models of P. (And if P is stratified, these are exactly the perfect models.)

The following example shows that D-WFS is indeed *strictly* weaker than STABLE, even for stratified disjunctive programs:

Example 5.1. [D-WFS is weaker than PERFECT]. Consider the following stratified disjunctive logic program due to ROSS:

$p \vee q.$
$r \leftarrow \mathit{not}\, p.$
$r \leftarrow \mathit{not}\, q.$

Here his S-WFS[3] as well as our semantics D-WFS leave r undefined, because p and q are undefined. In contrast, the perfect model semantics allows to conclude r, which is required for any semantics having a really exclusive or.

However, it depends on the application whether one really wants to conclude notr or not. In a similar example due to PRZYMUSINSKI, p means "go to Australia", q means "go to Europe", and r means "cancel reservation". Here, one can cancel the reservation only after the decision about the journey. □

A direct comparison with the perfect model semantics for disjunctive programs is not possible, because unfolding can destroy a stratification in the case of disjunctive programs. For non-disjunctive programs, this obviously cannot happen.

Example 5.2. [(GPPE destroys Stratification)] Consider the following stratified program:

$p \leftarrow q.$
$p \leftarrow \mathit{not}\, q.$
$q \vee r.$

It has two perfect (Herbrand) models, namely $I_1 := \{p, q\}$ and $I_2 := \{p, r\}$.

[3] The strong well-founded model introduced in [Ros89] at first looked very similar to our D-WFS. However, it can have a strange behaviour as demonstrated by the example $P = \{p \vee q \leftarrow r,\ p \vee r,\ r \leftarrow p\}$. Here the strong well-founded model contains $\mathit{not}\, p$. This is derivable neither by the GCWA nor by the perfect model semantics and is also a counterexample to Theorem 5.2 in [Ros89]. This example furthermore shows that unfolding is not possible in S-WFS. We tried to slightly modify the definition of S-WFS in order to avoid this counterexample (by removing to asymmetry between the deduction rules S1 and S2 in [Ros89]), but the modified semantics nevertheless does not allow unfolding.

But now consider what happens if we apply GPPE to the body literal q in the first rule:

$p \vee r.$
$p \leftarrow \mathit{not}\, q.$
$q \vee r.$

This logic program is not stratified. If we nevertheless naively try to apply the definition of the perfect model, we find that this program has no perfect model.

The stable model semantics does not have these problems, I_1 and I_2 are the two stable models of the first as well as the second program. □

5.3. Relation to Static Semantics

PRZYMUSINKI's static semantics [Prz95] is defined for arbitrary theories in the autoepistemic logic of beliefs (AEB). This is a much larger class than the class of disjunctive logic programs. However, we conjectured for quite some time that the static semantics would be equivalent to our D-WFS on the restricted domain of disjunctive logic programs and pure ground disjunctions as consequences (with each negative literal $\mathit{not}\, A$ translated to $\mathcal{B}(\neg A)$). For instance, the static semantics never directly assumes a disjunction of belief literals — $\mathcal{B}(\neg p) \vee \mathcal{B}(\neg q)$ follows from a disjunctive logic program under the static semantics if and only if already $\mathcal{B}(\neg p)$ or $\mathcal{B}(\neg q)$ follows by itself. In this way, the static semantics behaves very similarly to our D-WFS. Of course, in more general belief theories it is possible to express the exclusive or in the form $\mathcal{B}(\neg p \vee \neg q)$.

But even for the restricted class of belief theories and queries corresponding to disjunctive logic programs it turned out that D-WFS is strictly weaker than the static semantics (the failure of our conjecture was first noted by PRZYMUSINSKI):

Example 5.3. Consider the following logic program:

$p \vee q.$
$q \vee r.$
$s \leftarrow \mathit{not}\, p.$
$s \vee t \leftarrow \mathit{not}\, r.$

This program is a residual program, so in particular $\mathit{not}\, t$ does not follow under our D-WFS. However, it does follow under the static semantics: in all minimal models, p and r are either both true or both false. So the static semantics implies $(\mathit{not}\, p) \leftarrow (\mathit{not}\, r)$. But with this implication it becomes clear that t never needs to be true in any minimal model, because if $\mathit{not}\, r$ is true, then $\mathit{not}\, p$ and therefore s are also true. □

However, the static semantics allows our transformations (*if* the possible consequences are restricted to pure positive and pure negative disjunctions, see above). So D-WFS can at least be seen as an approximation of the static semantics: if $Q \in$ D-WFS(P), then Q is also contained in the least static expansion of P. The relation to the static semantics is more throughly investigated in [BDNP98].

5.4. Relation to GCWA

The generalized closed world assumption GCWA [Min82] is defined only for positive programs (disjunctive programs without negative body literals). Again, for our transformations this is no problem, because if they are applied in the "forward direction", they never introduce new negative literals. It is easy to see that the GCWA allows all our transformations: as the GCWA looks only at minimal models, it satisfies unfolding, elimination of tautologies and elimination of nonminimal rules by Theorem 4.2. Positive and negative reduction are trivially satisfied, because they are never applicable for positive programs. By Corollary 4.3, this means that $\mathrm{GCWA}(P) = \mathrm{GCWA}\big(res(P)\big)$, where $res(P)$ is a set of minimal positive disjunctions. It is easy to see that GCWA in this case agrees with D-WFS:

Theorem 5.4 (D-WFS extends GCWA). Let P be a positive program and Q be a negative ground literal. Then we have

$$Q \in \textit{D-WFS}(P) \iff Q \in \mathrm{GCWA}(P).$$

Why do we require Q being a negative literal and not an arbitrary pure disjunction? The original definition of GCWA in [Min82] only declares a set of certain negative literals to be derivable. The set of derivable positive disjunctions is taken as those that are true in all minimal models (see Section 4.2) Therefore our theorem remains true if Q is any positive ground disjunction.

Can we also compute negative pure disjunctions other than just negative literals? For example it makes sense to derive $\mathit{not}\, p \lor \mathit{not}\, q$ from the program $p \lor q$ because $\mathit{not} p \lor \mathit{not} q$ holds in all minimal models. In fact, this is called the EGCWA [GPP88]. Although we do not follow this idea here, it is quite simple to use the residual program for computing EGCWA. We just declare all those $\mathit{not}\, A_1 \lor \ldots \lor \mathit{not}\, A_n$ to be derivable, such that $A_1 \lor \ldots \lor A_n$ is subsumed by a clause in $res(P)$.

5.5. A Weak Version of the Disjunctive WFS

D-WFS as well as GCWA allow the elimination of nonminimal rules. For instance, in the program

$p \lor q.$
$p.$

it is possible to remove the first disjunction and to conclude $\mathit{not}\, q$. It has been argued that some applications need a really inclusive interpretation of "$\lor$", so that the negation of ground atom A should not be assumed if A appears in the head of an applicable rule. For such applications, the "weak GCWA" was defined [RT88, RLM89].

It is possible to define a similar version of our D-WFS, which we call WD-WFS. We simply replace the reduction operator R by a weaker version which does not remove nonminimal conditional facts:

Definition 5.1. (Weak Reduction Operator). For any set Γ of conditional facts, let

$$R_{\mathrm{weak}}(\Gamma) := \{\mathcal{A} \leftarrow \mathit{not}\, \big(\mathcal{C} \cap \mathit{heads}(\Gamma)\big) \mid \mathcal{A} \leftarrow \mathit{not}\, \mathcal{C} \in \Gamma,\ \text{and there is no } \mathcal{A}' \leftarrow \mathit{true} \in \Gamma \text{ with } \mathcal{A}' \subseteq \mathcal{C}\}.$$

As it is the case for the original reduction operator R, the total number of atoms occurring in Γ is reduced in each step so that the whole process must come to an end after finitely many steps.

Definition 5.2. (Weak Residual Program). Let P be any logic program, and let $\Gamma_0 := \mathrm{lfp}(T_P)$, $\Gamma_i := R_{\mathrm{weak}}(\Gamma_{i-1})$, and $n \in \mathbb{N}$ with $\Gamma_n = \Gamma_{n-1}$. Then we call $w\text{-}res(P) := \Gamma_n$ the weak residual program of P.

Definition 5.3. (WD-WFS). The semantics WD-WFS is defined by:

$$\mathrm{WD\text{-}WFS}(P) := known\big(w\text{-}res(P)\big).$$

So this is only an operational definition of WD-WFS. It might seem at first that WD-WFS can be characterized as the weakest semantics which allows unfolding, elimination of tautologies, and positive and negative reduction. However, it turns out that WD-WFS as well as WGCWA do not allow the elimination of tautologies:

$p \vee q \leftarrow p.$
$p.$

Here the first rule is a tautology, but it is important for WD-WFS and WGCWA, because it blocks the assumption of the negation *not* q.

The elimination of tautologies is used to show the equivalence of $\mathrm{lfp}(T_P)$ to the original program, so Corollary 4.1 is not applicable to the WGCWA. However, it is easy to show directly $\mathrm{WGCWA}(P) = \mathrm{WGCWA}\big(\mathrm{lfp}(T_P)\big)$. Since the reduction operator does nothing for positive programs, we get immediately:

Theorem 5.5 (WD-WFS extends WGCWA). Let P be a positive program and Q be a positive ground disjunction or negative ground literal. Then we have $Q \in$ WD-WFS$(P) \iff Q \in \mathrm{WGCWA}(P)$.

WD-WFS also agrees with the well-founded semantics on non-disjunctive programs. This shows, that in order to compute the well-founded model, it is not necessary to eliminate nonminimal rules.

Corollary 5.4 (Computation of WFS). Let P be any non-disjunctive program.

1. *For every positive ground literal A:*

$$A \in WFS(P) \iff (A \leftarrow true) \in w\text{-}res(P).$$

2. *For every negative ground literal not A:*

$$not\, A \in WFS(P) \iff A \notin heads\big(w\text{-}res(P)\big).$$

Let $\mapsto := \mapsto_U \cup \mapsto_T \cup \mapsto_P \cup \mapsto_N$. From Theorem 4.1 and Definition 1 it immediately follows that $P \mapsto^* w\text{-}res(P)$ for every ground program P. However, this rewriting system does not have the nice confluence property: from the program

$p \leftarrow p \wedge not\, q.$
$p.$

we can get to the two irreducible programs $P_1 = \{p\}$ and $P_2 = \{p,\ p \leftarrow not\, q\}$ (and P_2 would be the weak residual program).

6. COMPUTATIONAL PROPERTIES

In this section, we will make a few short comments on the computational aspects of our approach. We have investigated the computation of the residual program in more detail in [BD95a] for disjunctive programs and in [Bra96] for non-disjunctive programs.

It seems that every query evaluation algorithm which is able to handle non-stratified programs has to delay negative ground literals under certain conditions. For instance, this is done in [CW93, CSW95, CW95, CW96]. We believe that it is an important feature of our approach that such delaying and the whole computation can be understood on the source code level. Of course, specialized data structures can be useful for improving the efficiency, but they are not necessary for understanding the correctness of the method.

Our algorithm has a strong relation to the classical alternating fixpoint procedure, which was used in [KSS91] for bottom-up computation of the (non-disjunctive) WFS. More precisely, they restrict the conditional facts to the head literal and a one bit indication whether there is a non-trivial body or not. This is done by managing two versions of every predicate: the certainly true facts and the possibly true facts. Of course, this is a loss of information, but it can be compensated by recomputing the conditional facts for every step of the reduction phase. In [KSS91], also the optimization is used that they start with the computation of certainly true facts, and then the first computation of the possibly true facts can be combined already with the first reduction.

Even for non-disjunctive programs, there can be exponentially many derivable conditional facts in some rare circumstances. Since it is known that the WFS can be computed in polynomial time, it is not acceptable to compute the complete residual program first. However, if the residual program is only used as an intermediate step for computing stable models, the possible exponential growth is not a big problem, since already for non-disjunctive programs, the stable semantics is NP-complete. A solution for avoiding the exponential growth, invented by Chen and Warren for their SLG-resolution [CW93, CSW95, CW95, CW96], is to delay not only negative literals, but also positive literals which depend on already delayed negative literals. This idea has been adapted to our approach, and a practical algorithm for computing the well-founded model of non-disjunctive programs has been presented in [BZF97, ZBF97]. The algorithm consists in applying our transformations positive and negative Reduction, Success, Failure, and a Loop-detection rule (which is a special case of GPPE). All these transformations are of linear complexity and the corresponding algorithm is provably better than van Gelder's alternating fixpoint operator. We consider it as an interesting result that our framework also lead us to such an efficient algorithm.

It is interesting that positive and negative reduction and the corresponding operators for the evaluation of delayed positive body literals in trivial cases ("success" and "failure") exactly correspond to the least fixpoint of Fitting's Φ_P operator (see Theorem 2.3) used in some implementations of the WFS [SNV95, NS96]. It must be noted, however, that delaying positive literals leads to the problem of positive loops, and current solutions are neither elegant nor very efficient. In the approach presented here, this problem simply does not occur.

An important optimization of our approach is to interleave the computation of derived conditional facts with the reductions. If we know already that a ground atom A is true, it is not necessary to derive a conditional fact containing *not* A. If we know already that A is false, we can immediately evaluate *not* A to true, i.e. it is not necessary to delay the body literal. The number of ground atoms we know already to be true or false can be improved by ordering the computation according to the predicate dependencies. It is possible to compute the residual program locally for every strongly connected component of the program. For instance, for non-disjunctive stratified programs, it is never necessary to actually delay any negative body literal (as in the standard approach). It follows from our confluence results that optimizations changing the order of the application of the transformations are possible.

Finally, as for any bottom-up algorithm, we need some variant of the magic-set rewriting technique to make it goal-directed. For non-disjunctive programs this should not be very difficult, for disjunctive programs this is currently under research. However, it depends on the semantics whether such a transformation is at all possible. For instance, the STABLE semantics violates the "relevance" property ([DM94b, BD97]), so in general, a goal-directed computation is not possible.

7. CONCLUSIONS

In this paper, we have presented a general approach to define semantics for disjunctive logic programs simply by postulating some properties. Although we applied this framework to a particular set of transformations, our method is not restricted to these: we can also handle comp_3, WFS_C, STABLE and WGCWA (see Subsection 2.4). In [DO97b, DO97a, ADO97] still more transformation-rules are investigated.

The resulting semantics D-WFS turned out to be interesting, because it extends WFS and GCWA, and because of its strong relation to Przymusinski's static semantics, and the similarity to Ross's S-WFS. Furthermore, D-WFS is weaker than the disjunctive stable model semantics, so it gives no unsensible conclusions.

Besides the abstract definition of our semantics, we were also able to develop a bottom-up query evaluation algorithm for it. It is important to note that although our definition of Partial Evaluation was given on instantiated programs, our bottom-up procedure works on non-ground programs. Our computation of the residual program uses only the given semantical properties and can also be used for other semantics having these properties (e.g. the disjunctive stable semantics and the static semantics).

In fact, we proved that the validity of this computation is *equivalent* to the given semantical properties, which is an astonishing result: it was not clear from the beginning, that a particular bottom-up procedure can be linked to a set of declarative transformation rules.

Our approach is based on the notion of conditional facts, developed independently by Bry and Dung/Kanchansut. The delaying of negative body literals is also implicit in many query evaluation algorithms. It is nice that we can do this on the level of *programs*, and not on the level of *implementational data-structures*: we believe that such an approach greatly enhances the general understanding of the algorithms. Once the details are hidden in low-level data-structures, a real understanding is much more difficult.

As a byproduct of our approach, we have a characterization of the standard WFS as the weakest semantics allowing unfolding (GPPE), elimination of tautologies, and positive and negative reduction. Similar results hold for GCWA, STABLE, WFS_C and comp_3. In fact, suitable subsets of our transformations can be applied directly to programs to compute a residual program in a very efficient way (for WFS and comp_3). Queries can then be answered immediately from the residual program by using the identity in Definition 5. The underlying reason for this is the *confluence* of these calculi: a topic which is not investigated in this article (see [BD98a]).

A simple prototype of our approach is available[4]. Of course, the algorithm can be further optimized, this is subject of our future research. We are also interested to apply this framework to other semantics, and have already some results for the disjunctive stable semantics ([BD95a]).

Last but not least we thank the anonymous referees for valuable comments and useful hints.

REFERENCES

[AD95.] Chandrabose Aravindan and Phan Minh Dung. On the correctness of unfold/fold transformations of normal and extended logic programs. *The Journal of Logic Programming*, 24(3):201–218, September 1995.

[ADO97.] J. Arrazola, Jürgen Dix, and Mauricio Osorio. Confluent rewriting systems for logic programming semantics. Technical Report TR 27/97, University of Koblenz, Department of Computer Science, Rheinau 1, October 1997.

[BD95a.] Stefan Brass and Jürgen Dix. A General Approach to Bottom-Up Computation of Disjunctive Semantics. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 927, pages 127–155. Springer, Berlin, 1995.

[BD95b.] Stefan Brass and Jürgen Dix. Disjunctive Semantics based upon Partial and Bottom-Up Evaluation. In Leon Sterling, editor, *Proceedings of the 12th Int. Conf. on Logic Programming, Tokyo*, pages 199–213. MIT Press, June 1995.

[BD97.] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Stable Semantics by Partial Evaluation. *Journal of Logic Programming*, 32(3):207–228, 1997. (Extended abstract appeared in: Characterizations of the Stable Semantics by Partial Evaluation *LPNMR, Proceedings of the Third International Conference, Kentucky*, pages 85–98, 1995. LNCS 928, Springer.).

[BD98a.] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA. *Journal of Automated Reasoning*, 20(1):143–165, 1998. (Extended abstract appeared in: Characterizing D-WFS: Confluence and Iterated GCWA. *Logics in Artificial Intelligence, JELIA '96*, pages 268–283, 1996. Springer, LNCS 1126.).

[BD98b.] Gerhard Brewka and Jürgen Dix. Knowledge representation with extended logic programs. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition, Volume 6, Methodologies*, chapter 6. Reidel Publ., 1998.

[BDNP98.] Stefan Brass, Jürgen Dix, Ilkka Niemelä, and Teodor. C. Przymusinski. A Comparison of the Static and the Disjunctive Well-founded Semantics and its Implementation. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings*

[4] `ftp://ftp.informatik.uni-hannover.de/software/index.html`

of the Sixth International Conference (KR '98). San Francisco, CA, Morgan Kaufmann, May 1998. appeared also as TR 17/97, University of Koblenz.

[BDP96.] Stefan Brass, Jürgen Dix, and Teodor. C. Przymusinski. Super Logic Programs. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*, pages 529–541. San Francisco, CA, Morgan Kaufmann, 1996.

[BL92.] Stefan Brass and Udo W. Lipeck. Generalized bottom-up query evaluation. In Alain Pirotte, Claude Delobel, and Georg Gottlob, editors, *Advances in Database Technology — EDBT'92, 3rd Int. Conf.*, number 580 in LNCS, pages 88–103, Berlin, 1992. Springer.

[BNNS94.] Colin Bell, Anil Nerode, Raymond T. Ng, and V. S. Subrahmanian. Mixed Integer Programming Methods for Computing Non-Monotonic Deductive Databases. *Journal of the ACM*, 41(6):1178–1215, November 1994.

[Bra94.] Stefan Brass. Improved bottom-up query evaluation in positive disjunctive databases. In Bernd Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen,* (IFIP-Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases), pages 75–82, Berlin, 1994. Springer.

[Bra96.] Stefan Brass. Bottom-up query evaluation in extended deductive databases. Habilitationsschrift, Universität Hannover, Fachbereich Mathematik, 1996. `http://www.informatik.uni-hannover.de/~sb/habil.html`.

[Bry89.] François Bry. Logic programming as constructivism: A formalization and its application to databases. In *Proc. of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'89)*, pages 34–50, 1989.

[Bry90.] François Bry. Negation in logic programming: A formalization in constructive logic. In Dimitris Karagiannis, editor, *Information Systems and Artificial Intelligence: Integration Aspects*, pages 30–46, Berlin, 1990. Springer.

[BZF97.] Stefan Brass, Ulrich Zukowski, and Burkhard Freitag. Transformation-based bottom-up computation of the well-founded model. In Jürgen Dix, Luis Moniz Pereira, and Teodor C. Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming (NMELP'96)*, number 1216 in LNAI, pages 171–201. Springer, 1997.

[CL73.] Chin-Liang Chang and Richard C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973.

[CL95.] Stefania Costantini and Gaetano A. Lanzarone. Static Semantics as Program Transformation and Well-founded Computation. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 927, pages 156–180. Springer, Berlin, 1995.

[Cla78.] Keith L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data-Bases*, pages 293–322. Plenum, New York, 1978.

[CSW95.] Weidong Chen, Terrance Swift, and David S. Warren. Efficient Top-Down Computation of Queries under the Well-Founded Semantics. *Journal of Logic Programming*, 24(3):219–245, 1995.

[CW93.] Weidong Chen and David S. Warren. A Goal Oriented Approach to Computing The Well-founded Semantics. *Journal of Logic Programming*, 17:279–300, 1993.

[CW95.] Weidong Chen and David S. Warren. Computing of Stable Models and its Integration with Logical Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 17:279–300, 1995.

[CW96.] Weidong Chen and David S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, January 1996.

[DFN98.] Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In Andrei Voronkov and Alan Robinson, editors, *Handbook of Automated Reasoning*. Elsevier-Science-Press, to appear 1998.

[Dix92.] Jürgen Dix. A Framework for Representing and Characterizing Semantics of Logic Programs. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, pages 591–602. San Mateo, CA, Morgan Kaufmann, 1992.

[Dix95a.] Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.

[Dix95b.] Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.

[Dix95c.] Jürgen Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In Andre Fuhrmann and Hans Rott, editors, *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, pages 241–327. DeGruyter, 1995.

[DK89a.] P. M. Dung and K. Kanchansut. A fixpoint approach to declarative semantics of logic programs. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of North American Conference Cleveland,Ohio, USA*. MIT Press, October 1989.

[DK89b.] P. M. Dung and K. Kanchansut. A natural semantics for logic programs with negation. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of the 9th Conference on Foundations of Software Technology and Theoretical Computer Science*, Berlin, 1989. Springer.

[DM93.] Jürgen Dix and Martin Müller. Implementing Semantics for Disjunctive Logic Programs Using Fringes and Abstract Properties. In Luis Moniz Pereira and Anil Nerode, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Second International Workshop*, pages 43–59, Cambridge, Mass., July 1993. Lisbon, MIT Press.

[DM94a.] Jürgen Dix and Martin Müller. An Axiomatic Framework for Representing and Characterizing Semantics of Disjunctive Logic Programs. In Pascal Van Hentenryck, editor, *Proceedings of the 11th Int. Conf. on Logic Programming, S. Margherita Ligure*, pages 303–322. MIT, June 1994.

[DM94b.] Jürgen Dix and Martin Müller. Partial Evaluation and Relevance for Approximations of the Stable Semantics. In Z.W. Ras and M. Zemankova, editors, *Proceedings of the 8th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1994*, LNAI 869, pages 511–520, Berlin, 1994. Springer.

[DO97a.] Jürgen Dix and Mauricio Osorio. On Well-Behaved Semantics Suitable for Aggregation. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the 1997 International Symposium*, page 505. MIT Press, 1997.

[DO97b.] Jürgen Dix and Mauricio Osorio. Provability Closures in Logic Programming. In Instituto Politectico Nacional, editor, *Proceedings of the International Symposium on Computer Science in Mexico*, pages 313–319, 1997.

[DS98.] Jürgen Dix and Frieder Stolzenburg. A Framework to incorporate Nonmonotonic Reasoning into Constraint Logic Programming. *Journal of Logic Programming*, 36(1,2,3):5—37, 1998. Special Issue on *Constraint Logic Programming*, Guest Editors: Kim Marriott and Peter Stuckey.

[Fit85.] Melvin C. Fitting. A Kripke-Kleene Semantics of logic Programs. *Journal of Logic Programming*, 4:295–312, 1985.

[GL88.] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[GL91.] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–387, 1991. (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.).

[GPP88.] Michael Gelfond, Halina Przymusinska, and Teodor Przymusinski. The extended Closed World Assumption and its Relationship to parallel Circumscription. In *Proceedings 7th Symposion on Principles of Database Systems*, 1988.

[GPP90.] Michael Gelfond, Halina Przymusinska, and Teodor Przymusinski. On the Relationship between CWA, minimal model and minimal Herbrand model Semantics. *International Journal of Intelligent Systems*, 5(5):549–564, 1990.

[HY91.] Yong Hu and Li Yan Yuan. Extended Well-Founded Model Semantics for General Logic Programs. In Koichi Furukawa, editor, *Proceedings of the 8th Int. Conf. on Logic Programming, Paris*, pages 412–425. MIT, June 1991.

[KSS91.] David B. Kemp, Peter J. Stuckey, and Divesh Srivastava. Magic Sets and Bottom-Up Evaluation of Well-Founded Models. In Vijay Saraswat and Kazunori Ueda, editors, *Proceedings of the 1991 Int. Symposium on Logic Programming*, pages 337–351. MIT, June 1991.

[Min82.] Jack Minker. On indefinite databases and the closed world assumption. In *Proceedings of the 6th Conference on Automated Deduction, New York*, pages 292–308, Berlin, 1982. Springer.

[MR90.] Jack Minker and Arcot Rajasekar. A fixpoint semantics for disjunctive logic programs. *Journal of Logic Programming*, 9:45–74, 1990.

[NS96.] Ilkka Niemelä and Patrik Simons. Efficient Implementation of the Well-founded and Stable Model Semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[Prz91a.] Teodor Przymusinski. Semantics of disjunctive logic programs and deductive databases. In Claude Delobel, Michael Kifer, and Yoshifumi Masunaga, editors, *Deductive and Object-Oriented Databases, 2nd Int. Conf. (DOOD'91)*, number 566 in LNCS, pages 85–107, Berlin, 1991. Springer.

[Prz91b.] Teodor Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing Journal*, 9:401–424, 1991. (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.).

[Prz95.] Teodor Przymusinski. Static Semantics For Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357, 1995.

[RLM89.] Arcot Rajasekar, Jorge Lobo, and Jack Minker. Weak Generalized Closed World Assumption. *Journal of Automated Reasoning*, 5:293–307, 1989.

[Ros89.] Kenneth A. Ross. The well-founded semantics for disjunctive logic programs. In *Proceedings of the first International Conference on Deductive and Object Oriented Databases, Kyoto, Japan*, pages 1–22, 1989.

[RT88.] Kenneth A. Ross and Rodney A. Topor. Inferring negative Information from disjunctive Databases. *Journal of Automated Reasoning*, 4:397–424, 1988.

[Sch92.] John S. Schlipf. Formalizing a Logic for Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 5:279–302, 1992.

[Sch95.] John S. Schlipf. The Expressive Powers of the Logic Programming Semantics. *Journal of Computer and System Sciences*, 5(1):64–86, 1995.

[SNV95.] V.S. Subrahmanian, Nau, and Vago. WFS + Branch and Bound = Stable Models. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):362–377, 1995.

[SS94.] Chiaki Sakama and Hirohisa Seki. Partial Deduction of Disjunctive Logic Programs: A Declarative Approach. In *Logic Program Synthesis and Transformation – Meta Programming in Logic*, LNCS 883, pages 170–182, Berlin, 1994. Springer.

[SS97.] Chiaki Sakama and Hirohisa Seki. Partial Deduction in Disjunctive Logic Programming. *Journal of Logic Programming*, 32(3):229–245, 1997.

[VG89.] Allen Van Gelder. 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*, pages 1–10. ACM Press, March 1989.

[VGRS91.] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.

[ZBF97.] Ulrich Zukowski, Stefan Brass, and Burkhard Freitag. Improving the alternating fixpoint: The transformation approach. In J. Dix, U. Furbach, and A. Nerode, editors, *Proc. of the 4th Int. Conf. on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, number 1265 in LNAI, pages 40–59. Springer, 1997.

A. PROOFS OF THEOREMS FROM SECTION 3

Lemma 3.2. The mapping $\mathcal{S}_{res}$ satisfies the conditions of a semantics.

PROOF.

1. Because of the allowedness, the variables are completely instantiated in the computation of conditional facts. So the same conditional facts are computed based on P and on $ground(P)$. Therefore, the input to the second phase of the computation is the same, so the resulting residual program is also identical.

2. The possibility of right weakening is obvious from the definition.

3. Let $\mathcal{A} \leftarrow true$ be in P. Then it is also contained in $\mathrm{lfp}(T_P)$ and the only reason why it might not be in the resulting residual program is that there is a disjunctive fact $\mathcal{A}' \leftarrow true$ in the residual program with $\mathcal{A}' \subset \mathcal{A}$. In both cases we get $\mathcal{A} \in \mathcal{S}_{res}(P)$.

4. Let A be a Σ-ground atom with $A \notin heads(P)$. The derived disjunctive facts can contain only head literals already present in P, so $A \notin heads\big(\mathrm{lfp}(T_P)\big)$. The reduction operations also introduce no new head literals, and we get $A \notin heads\big(res(P)\big)$. But this means that $not\,A \in \mathcal{S}_{res}(P)$. $\Box$

Lemma 3.3. For all sets Γ and Γ' of conditional facts:

$\Gamma \mapsto_{MPN} \Gamma' \implies res(\Gamma) = res(\Gamma')$.

I.e. the result of the reduction phase is invariant under elimination of nonminimal rules as well as positive and negative reduction.

PROOF. The proof is by induction on the number of applications n of the operator R to compute $res(\Gamma)$. However, we need to prove a slightly stronger induction hypothesis, namely in addition to $res(\Gamma) = res(\Gamma')$ we also prove that Γ' never needs more reduction steps than Γ.

There is nothing to prove for $n = 0$, because if $res(\Gamma) = \Gamma$, then there is no Γ' with $\Gamma \mapsto_{MPN} \Gamma'$. Let us assume in the following that $res(\Gamma) \neq \Gamma$.

For the inductive step, we first show that $\Gamma \mapsto_{MPN} \Gamma'$ implies $R(\Gamma) \mapsto^*_{MPN} R(\Gamma')$:

1. Suppose that $\Gamma \mapsto_M \Gamma'$ and that the conditional fact $\mathcal{A} \leftarrow not\,\mathcal{C}$ was deleted. If we apply the reduction operator once, $\mathcal{A} \leftarrow not\,\mathcal{C}$ is of course deleted. Otherwise $R(\Gamma)$ and $R(\Gamma')$ agree, except that $heads(\Gamma) \supset heads(\Gamma')$ might hold because of the additional rule $\mathcal{A} \leftarrow not\,\mathcal{C}$ in Γ. This means that the only difference between $R(\Gamma)$ and $R(\Gamma')$ is that some negative body literals have not been evaluated to true because they occurred in $\mathcal{A}$. Therefore, we have $R(\Gamma) \mapsto^*_P R(\Gamma')$.

2. Let $\Gamma \mapsto_P \Gamma'$ and let the body literal $not\,A$ with $A \notin heads(\Gamma)$ be evaluated to true in $\mathcal{A} \leftarrow not\,\mathcal{C}$. Obviously, $not\,A$ is also evaluated to true by the operator R. There are only two possibilities why $R(\Gamma)$ and $R(\Gamma')$ might differ: First, the rule $\mathcal{A} \leftarrow not\,\big(\mathcal{C} - \{not\,A\}\big)$ can eliminate more nonminimal rules than $\mathcal{A} \leftarrow not\,\mathcal{C}$. Second, if $\mathcal{C} = not\,A$, it is possible that in $R(\Gamma')$ rules are deleted containing $\mathcal{A}'$ in the body. Thus, we again have $R(\Gamma) \mapsto^*_{MPN} R(\Gamma')$.

3. Let $\Gamma \mapsto_N \Gamma'$ and let $\mathcal{A} \leftarrow \mathit{not}\,\mathcal{C}$ be deleted because of $\mathcal{A}' \leftarrow \mathit{true}$. Of course, the first application of the reduction operator also deletes $\mathcal{A} \leftarrow \mathit{not}\,\mathcal{C}$, but as in the first case, it is possible that $\mathit{heads}(\Gamma) \supset \mathit{heads}(\Gamma')$. Thus we again have $R(\Gamma) \mapsto_P^* R(\Gamma')$.

So we now have $R(\Gamma) \mapsto_{MPN}^* R(\Gamma')$. Let $\Gamma_0 := R(\Gamma)$, $\Gamma_{i-1} \mapsto_{MPN} \Gamma_i$, and $\Gamma_m = R(\Gamma')$. By applying the induction hypothesis the first time, we get $\mathit{res}(\Gamma_0) = \mathit{res}(\Gamma_1)$ and that Γ_1 does not need more reduction steps than Γ_0. So we can apply the induction hypothesis iteratively and finally get $\mathit{res}(\Gamma_0) = \mathit{res}(\Gamma_1) = \cdots = \mathit{res}(\Gamma_m)$, and thus $\mathit{res}(\Gamma) = \mathit{res}(\Gamma')$.

By applying the inductive hypothesis iteratively, we also get that $\Gamma_m = R(\Gamma')$ does not need more applications of R than $\Gamma_0 = R(\Gamma)$. Since $R(\Gamma) = \Gamma$ is excluded, we have also proven the second half of the inductive proposition. □

Lemma 3.4. The operator D_Γ is monotonic (on the standard powerset lattice).

PROOF. Let $\Delta_1 \subseteq \Delta_2$ and let $Q \in D_\Gamma(\Delta_1)$.

1. If Q is a positive disjunction, this means there is $\mathcal{A} \leftarrow \mathit{not}\,\mathcal{C} \in \Gamma$ such that $\mathcal{C} \subseteq \Delta_1^-$. But then also $\mathcal{C} \subseteq \Delta_2^-$ holds.
2. If Q is a negative disjunction, this means that there is $\mathit{not}\,A \in Q$ such that for every $\mathcal{A} \leftarrow \mathit{not}\,\mathcal{C} \in \Gamma$ with $A \in \mathcal{A}$ (at least) one of the following two cases holds:
   - There is $\mathcal{A}' \in \Delta_1^+$ with $\mathcal{A}' \subseteq \mathcal{C}$. But then $\mathcal{A}'$ is also contained in Δ_2^+.
   - There is $\mathcal{A}' \leftarrow \mathit{not}\,\mathcal{C}' \in \Gamma$ with $\mathcal{A}' \subseteq \mathcal{A} - A$ and $(\mathcal{C}' - \Delta_1^-) \subseteq (\mathcal{C} - \Delta_1^-)$. I.e. for every $A' \in \mathcal{C}'$ with $A' \notin \Delta_1^-$, we have $A' \in \mathcal{C}$. But this condition gets only weaker if we replace Δ_1 by the superset Δ_2. □

Theorem 3.1. Let P be any (finite) logic program, and $\Gamma := \mathrm{lfp}(T_P)$. Then:

$\mathcal{S}_{res}(P) = \mathrm{lfp}(D_\Gamma)$ *and* $\mathit{res}(P) = \Gamma / \mathrm{lfp}(D_\Gamma)$.

PROOF. Let $\Gamma_0 := \Gamma/\emptyset$ and $\Gamma_{i+1} := \Gamma/\mathit{known}(\Gamma_i)$. Obviously, we have that $\Gamma \mapsto_{MPN}^* \Gamma_0$ and $\Gamma_i \mapsto_{MPN}^* \Gamma_{i+1}$. But then it is clear that the number of occurring literals decreases in every step, so we must get a Γ_n which is irreducible with respect to $\mapsto_{MPN}$, so $\Gamma_{n+1} = \Gamma_n$. By Lemma 3.3, we have $\mathit{res}(\Gamma) = \mathit{res}(\Gamma_n)$. Since Γ_n is irreducible, we have $\mathit{res}(\Gamma_n) = \Gamma_n$. It follows that $\mathit{res}(\Gamma) = \Gamma_n$ and thus $\mathcal{S}_{res}(P) = \mathit{known}(\Gamma_n)$.

The standard construction of $\mathrm{lfp}(D_\Gamma)$ is (in this finite case): $\Delta_0 := \emptyset$, $\Delta_{i+1} := \mathit{known}(\Gamma/\Delta_i)$ until a fixpoint is reached. This means that $\Gamma_i = \Gamma/\Delta_i$ and $\Delta_{i+1} = \mathit{known}(\Gamma_i)$ for all $i \geq 0$. But since $\Gamma_{n+1} = \Gamma_n$, this also means $\Delta_{n+2} = \Delta_{n+1}$, so $\mathrm{lfp}(D_\Gamma) = \Delta_{n+1} = \mathit{known}(\Gamma_n) = \mathit{known}\big(\mathit{res}(\Gamma)\big)$. This proves $\mathcal{S}_{res}(P) = \mathrm{lfp}(D_\Gamma)$.

Since $\Gamma_{n+1} = \Gamma/\mathit{known}(\Gamma_n) = \Gamma_n = \mathit{res}(P)$, and $\mathit{known}(\Gamma_n) = \mathrm{lfp}(D_\Gamma)$, it follows that $\mathit{res}(P) = \Gamma/\mathrm{lfp}(D_\Gamma)$. □

B. PROOF OF THEOREM 4.1

Our goal is to show that it is possible to reach $\mathrm{lfp}(T_P)$ from P by a series of unfolding, weak unfolding and "elimination of tautology"-steps. The proof would be much simpler if we did not care about nonminimal conditional facts. However, since this restriction is not needed, and there are semantics, for which nonminimal rules are relevant, we prove the stronger and more elegant result. Let us first look at some examples to illustrate the problems:

Example B.1. We once hoped that if P' results from P by unfolding, then $\mathrm{lfp}(T_P) = \mathrm{lfp}(T_{P'})$. This, however, does not hold. Let P be:

$p \vee q \leftarrow r.$
$r \leftarrow q.$
$q \vee r.$

For this program, we get $\mathrm{lfp}(T_P) = \{q \vee r, p \vee q, r, p \vee r\}$. Now if we unfold r in the first rule, we replace this rule by

$p \vee q \leftarrow q.$
$p \vee q.$

Note that $p \vee q \leftarrow q$ is a tautology, and in fact, this tautology is the source of the problem: with it we can derive $p \vee q \vee r$ which is not in $\mathrm{lfp}(T_P)$. $\square$

Example B.2. In the above example, it would have been possible to simply eliminate the tautology before further unfolding steps in order to avoid the derivation of the critical conditional fact. But this is not always correct, because if the program P itself contains a tautology, e.g.

$p \vee q \leftarrow q.$
$q \vee r.$

then we of course have to apply the tautology to get the conditional fact $p \vee q \vee r$ contained in $\mathrm{lfp}(T_P)$. $\square$

Example B.3. In fact, tautologies can also occur as intermediate results because unfolding in contrast to hyperresolution evaluates only one body literal at a time. Consider for example the following program:

$p \leftarrow q \wedge r.$
$q \vee r.$
$r \vee s.$

If we unfold the first body literal, we get $p \vee r \leftarrow r$. Then another application of GPPE yields $p \vee r \vee s$, the result of hyperresolution. $\square$

Lemma B.1. Let P be a ground logic program containing the two rules

$$\begin{aligned} \mathcal{A} &\leftarrow B_0 \wedge B_1 \wedge \cdots \wedge B_m \wedge \mathit{not}\, \mathcal{C}. \\ B_0 \vee \mathcal{A}' &\leftarrow \qquad B_1' \wedge \cdots \wedge B_n' \wedge \mathit{not}\, \mathcal{C}'. \end{aligned}$$

Let P' contain in addition the rule which would result from unfolding, i.e.

$$P' := P \cup \big\{(\mathcal{A} \cup \mathcal{A}') \leftarrow B_1 \wedge \cdots \wedge B_m \wedge B_1' \wedge \cdots \wedge B_n' \wedge \mathit{not}\, (\mathcal{C} \cup \mathcal{C}')\big\}.$$

If the added rule is not a tautology, then $\mathrm{lfp}(T_{P'}) \subseteq \mathrm{lfp}(T_P)$.

Proof. To simplify the notation, we consider only the case without $\mathcal{C}, \mathcal{C}'$. This is no real restriction, since negative body literals can be moved into the heads if they are distinguished by making their predicate symbols disjoint from the positive head- and body-literals.

Now let the new rule be applied to disjunctive facts $\mathcal{A}_i$, $i = 1, \dots, m$, and $\mathcal{A}'_j$, $j = 1, \dots, n$. Then the resulting disjunctive fact is

$$\mathcal{A} \cup \mathcal{A}' \cup \bigcup_{i=1}^{m} \bigl(\mathcal{A}_i - \{B_i\}\bigr) \cup \bigcup_{j=1}^{n} \bigl(\mathcal{A}'_j - \{B'_j\}\bigr). \tag{$*$}$$

Now we have to distinguish several cases to show that this conditional fact is also derivable in P. Each case fills a hole in the preceding case.

1. The natural idea to derive this disjunctive fact would be to first apply the rule $B_0 \vee \mathcal{A}' \leftarrow B'_1 \wedge \dots \wedge B'_n$ and get

   $$\{B_0\} \cup \mathcal{A}' \cup \bigcup_{j=1}^{n} \bigl(\mathcal{A}'_j - \{B'_j\}\bigr).$$

   We then enter this for B_0 into $\mathcal{A} \leftarrow B_0 \wedge B_1 \wedge \dots \wedge B_m$. This results in

   $$\mathcal{A} \cup \bigl(\mathcal{A}' - \{B_0\}\bigr) \cup \bigcup_{j=1}^{n} \bigl(\mathcal{A}'_j - \{B'_j, B_0\}\bigr) \cup \bigcup_{i=1}^{m} \bigl(\mathcal{A}_i - \{B_i\}\bigr).$$

   Of course, $\mathcal{A}'$ does not contain B_0, so this set-difference is effectless and $\bigl(\mathcal{A}' - \{B_0\}\bigr)$ can be simplified to $\mathcal{A}'$. However, we have to assume that the $\mathcal{A}'_j$, $j = 1, \dots, n$, do not contain B_0 (or that $B'_j = B_0$) in order to really get $(*)$. This hole is treated in case 2.

2. Now we assume that $B_0 \in \mathcal{A}'_j$ and $B_0 \neq B'_j$ for at least one $1 \leq j \leq n$. Without loss of generality, we can choose $j = 1$ to simplify the notation. Then we first apply the rule $\mathcal{A} \leftarrow B_0 \wedge B_1 \wedge \dots \wedge B_m$ and enter $\mathcal{A}'_1$ for the body literal B_0. This results in

   $$\mathcal{A} \cup \bigl(\mathcal{A}'_1 - \{B_0\}\bigr) \cup \bigcup_{i=1}^{m} \bigl(\mathcal{A}_i - \{B_i\}\bigr).$$

   Since $B_0 \neq B'_1$, the generated disjunctive fact contains B'_1. Therefore, we can insert it into $B_0 \vee \mathcal{A}' \leftarrow B'_1 \wedge \dots \wedge B'_n$. This gives

   $$\begin{aligned}\{B_0\} \cup \mathcal{A}' \cup \bigl(\mathcal{A} - \{B'_1\}\bigr) \cup \bigl(\mathcal{A}'_1 - \{B_0, B'_1\}\bigr)\\ \cup \bigcup_{i=1}^{m} \bigl(\mathcal{A}_i - \{B_i, B'_1\}\bigr) \cup \bigcup_{j=2}^{n} \bigl(\mathcal{A}'_j - \{B'_j\}\bigr).\end{aligned}$$

   Now let us check that this is equivalent to $(*)$. First, $\{B_0\} \cup \bigl(\mathcal{A}'_1 - \{B_0, B'_1\}\bigr)$ can be simplified to $\mathcal{A}'_1 - \{B'_1\}$, since $B_0 \in \mathcal{A}'_1$ and $B_0 \neq B'_1$. Second, $\mathcal{A}$ cannot contain B'_1, since otherwise the combined rule would be a tautology. However, we have to assume now that $B'_1 \notin \mathcal{A}_i$ or $B'_1 = B_i$ for $i = 1, \dots, m$ (otherwise see case 3). Then the above expression is equivalent to $(*)$.

3. The current situation is as follows: $B_0 \in \mathcal{A}'_j$ and $B_0 \neq B'_j$ for at least one $1 \leq j \leq n$, furthermore $B'_j \in \mathcal{A}_i$ and $B'_j \neq B_i$ for at least one $1 \leq i \leq m$. Again, without loss of generality, we can choose $j = 1$ and $i = 1$. Then we first enter $\mathcal{A}_1$ for B'_1 in the rule $B_0 \vee \mathcal{A}' \leftarrow B'_1 \wedge \cdots \wedge B'_m$. This results in

$$\{B_0\} \cup \mathcal{A}' \cup \big(\mathcal{A}_1 - \{B'_1\}\big) \cup \bigcup_{j=2}^{n} \big(\mathcal{A}'_j - \{B'_j\}\big).$$

Since $B'_1 \neq B_1$, the generated disjunctive fact contains B_1. Therefore, we can insert it for B_1 in the rule $\mathcal{A} \leftarrow B_0 \wedge B_1 \wedge \cdots \wedge B_m$, and insert $\mathcal{A}'_1$ for B_0. The result is

$$\mathcal{A} \cup \big(\mathcal{A}'_1 - \{B_0\}\big) \cup \big(\{B_0\} - \{B_1\}\big) \cup \big(\mathcal{A}' - \{B_1\}\big) \cup \big(\mathcal{A}_1 - \{B'_1, B_1\}\big) \\ \cup \bigcup_{j=2}^{n} \big(\mathcal{A}'_j - \{B'_j, B_1\}\big) \cup \bigcup_{i=2}^{m} \big(\mathcal{A}_i - \{B_i\}\big).$$

Now let us check that this is indeed equivalent to $(*)$. First, we allow no duplicate body literals, so $B_0 \neq B_1$, therefore the term $\{B_0\} - \{B_1\}$ entails that B_0 is contained in the result, and we already know that it is also contained in $(*)$. Second, we have $\mathcal{A}'_1 - \{B_0\}$ instead of $\mathcal{A}'_1 - \{B'_1\}$. But the result already contains B_0, so the set-difference has no effect on the total result, and $(*)$ also contains B'_1 (because $B'_1 \in \mathcal{A}_1$ and $B'_1 \neq B_1$), so it is correct that the result contains B'_1 (since $B'_1 \neq B_0$). Third, $\mathcal{A}'$ cannot contain B_1, since otherwise the combined rule would be a tautology. Fourth, we eliminate B'_1 from $\mathcal{A}_1$, but it is still contained in the result as explained above. Finally, we have to assume that $B_1 \notin \mathcal{A}'_j$ (or else $B_1 = B'_j$) for $j = 2, \ldots, n$ (see case 4).

4. Now, the remaining little hole can be described as follows: as in case 3, we have $B_0 \in \mathcal{A}'_1$, $B_0 \neq B'_1$ and $B'_1 \in \mathcal{A}_1$, $B'_1 \neq B_1$. Now in addition, there is a $2 \leq j \leq n$, with $B_1 \in \mathcal{A}'_j$ and $B_1 \neq B'_j$. To simplify the notation, we choose $j = 2$. In this case, we first apply the rule $\mathcal{A} \leftarrow B_0 \wedge B_1 \wedge \cdots \wedge B_m$ and insert $\mathcal{A}'_1$ for B_0 and $\mathcal{A}'_2$ for B_1. This results in

$$\mathcal{A} \cup \big(\mathcal{A}'_1 - \{B_0\}\big) \cup \big(\mathcal{A}'_2 - \{B_1\}\big) \cup \bigcup_{i=2}^{m} \big(\mathcal{A}_i - \{B_i\}\big).$$

Now we apply the rule $B_0 \vee \mathcal{A}' \leftarrow B'_1 \wedge \cdots \wedge B'_n$ and insert $\mathcal{A}_1$ for B'_1 and the above constructed disjunctive fact for B'_2 (Since $B'_2 \neq B_1$, the constructed disjunctive fact really contains B'_2.). The result is

$$\{B_0\} \cup \mathcal{A}' \cup \big(\mathcal{A}_1 - \{B'_1\}\big) \cup \big(\mathcal{A} - \{B'_2\}\big) \\ \cup \big(\mathcal{A}'_1 - \{B_0, B'_2\}\big) \cup \big(\mathcal{A}'_2 - \{B_1, B'_2\}\big) \\ \cup \bigcup_{i=2}^{m} \big(\mathcal{A}_i - \{B_i, B'_2\}\big) \cup \bigcup_{j=3}^{n} \big(\mathcal{A}'_j - \{B'_j\}\big).$$

Now the result contains

- B_0,
- B_1, since $B_1 \in \mathcal{A}_1$ and $B_1' \neq B_1$,
- B_1' since $B_1' \in \mathcal{A}_1'$, $B_0 \neq B_1'$ and $B_2' \neq B_1'$ (we allow no duplicate body literals).

So set-differences with these atoms have no effect. Next, $\mathcal{A} - \{B_2'\} = \mathcal{A}$, since $B_2' \in \mathcal{A}$ would make the combined rule a tautology. By applying these simplifications, we get:

$$\{B_0\} \cup \mathcal{A}' \cup \mathcal{A}_1 \cup \mathcal{A} \cup \big(\mathcal{A}_1' - \{B_2'\}\big) \cup \big(\mathcal{A}_2' - \{B_2'\}\big) \\ \cup \bigcup_{i=2}^{m} \big(\mathcal{A}_i - \{B_i, B_2'\}\big) \cup \bigcup_{j=3}^{n} \big(\mathcal{A}_j' - \{B_j'\}\big).$$

Note that also $(*)$ contains

- B_0 (since $B_0 \in \mathcal{A}_1'$ and $B_0 \neq B_1'$),
- B_1 (since $B_1 \in \mathcal{A}_2'$ and $B_1 \neq B_2'$), and
- B_1' (since $B_1' \in \mathcal{A}_1$ and $B_1' \neq B_1$).

Therefore, in order to get from the above expression to $(*)$, we only need the following assumptions: $B_2' \notin \mathcal{A}_1'$ and $B_2' \notin \mathcal{A}_i$ (or $B_2' = B_i$) for $i = 2, \ldots, m$ (otherwise see case 5).

5. The current situation is as follows: The resulting disjunctive fact $(*)$ contains

   - B_0 (because $B_0 \in \mathcal{A}_1'$, $B_0 \neq B_1'$),
   - B_1' (because $B_1' \in \mathcal{A}_1$, $B_1' \neq B_1$),
   - B_1 (because $B_1 \in \mathcal{A}_2'$, $B_1 \neq B_2'$), and
   - B_2' (because either $B_2' \in \mathcal{A}_1'$, $B_2' \neq B_1'$ or $B_2' \in \mathcal{A}_i$, $2 \leq i \leq m$, $B_2' \neq B_i$).

   If we apply the rules as in case 1 (i.e. in the most natural way), our only risk is to loose B_0. Let the result be $\hat{\mathcal{A}}_1$. Next, we apply the rules as in case 3. This time, we may loose B_1. Let the result be $\hat{\mathcal{A}}_2$. Since $B_0 \neq B_1'$, $\hat{\mathcal{A}}_1$ contains B_1', and since $B_1 \neq B_2'$, $\hat{\mathcal{A}}_2$ contains B_2'. Therefore, we can insert $\hat{\mathcal{A}}_1$ and $\hat{\mathcal{A}}_2$ for B_1' and B_2' into $B_0 \vee \mathcal{A}' \leftarrow B_1' \wedge B_2' \wedge \cdots \wedge B_n'$. The result surely contains B_0, and also B_1 (from $\hat{\mathcal{A}}_1 - \{B_1'\}$), as well as B_1' (from $\hat{\mathcal{A}}_2 - \{B_2'\}$), and B_2' (from $\hat{\mathcal{A}}_1 - \{B_1'\}$ or directly if $B_2' = B_0$). The rest of $(*)$ was already contained in the intermediate result and simply carries through. □

Lemma B.2. Let P be an instantiated logic program containing a rule

$\mathcal{A} \leftarrow B_0 \wedge B_1 \wedge \cdots \wedge B_m \wedge \mathit{not}\, \mathcal{C}$

and a conditional fact $B_0 \vee \mathcal{A}' \leftarrow \mathit{not}\, \mathcal{C}'$. Let

$P' := P \cup \big\{(\mathcal{A} \cup \mathcal{A}') \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \mathit{not}\, (\mathcal{C} \cup \mathcal{C}')\big\}$.

Then $\mathrm{lfp}(T_{P'}) \subseteq \mathrm{lfp}(T_P)$.

Proof. This is trivial: if something is derivable using the new rule, then we can also use the old rule with the conditional fact. □

Theorem 4.1 (Conditional Facts can be Constructed by Transformations). The set of derived conditional facts can be constructed from the ground instantiation of *a program by applying only unfolding, weak unfolding, and the elimination of tautologies. I.e. $ground(P) \mapsto^*_{UWT} \mathrm{lfp}(T_P)$ holds for all programs P.*

PROOF. We have to construct $\mathrm{lfp}(T_P)$ from the ground instantiation of P by using only unfolding, weak unfolding, and elimination of tautologies. Note that $ground(P)$ is finite because of our restrictions on Σ.

1. First, we apply weak unfolding until nothing changes. However, directly after every application of weak unfolding, we delete every newly generated tautology, which did not result from applying a conditional fact. Therefore, the added rules do not increase the set of derivable conditional facts by Lemma B.1 and Lemma B.2. This means in particular that all generated conditional facts are contained in $\mathrm{lfp}(T_P)$.

   On the other hand, every conditional fact in $\mathrm{lfp}(T_P)$ is generated: for instance, consider the rule $\mathcal{A} \leftarrow B_1 \wedge \cdots \wedge B_n \wedge not\, \mathcal{C}$ applied to conditional facts $\mathcal{A}_i \leftarrow not\, \mathcal{C}_i$. The result contained in $\mathrm{lfp}(T_P)$ is

   $$\mathcal{A} \cup \bigcup_{i=1}^{n} \big(\mathcal{A}_i - \{B_i\}\big) \leftarrow not\, \big(\mathcal{C} \cup \bigcup_{i=1}^{n} \mathcal{C}_i\big).$$

   By applying GPPE to the first body literal, we get

   $$\mathcal{A} \cup \big(\mathcal{A}_1 - \{B_1\}\big) \leftarrow B_2 \wedge \cdots \wedge B_n \wedge not\, \big(\mathcal{C} \cup \mathcal{C}_1\big).$$

   Next, we apply GPPE to B_2 in this rule, and insert $\mathcal{A}_2 \leftarrow not\, \mathcal{C}_2$. And so on.

2. Second, we delete all rules which still have positive body literals by applying unfolding. In order to do this, we first delete all tautologies (we also delete immediately any tautology which is later generated). Since there are no tautologies, unfolding some positive body literal generates only rules without this body literal. And once this body literal has vanished from all rules in the program, it can never be introduced again by unfolding. Therefore, all occurring positive body literals can be eliminated one after the other. Note also that no rules are created which are not already contained in the program due to the first phase. Therefore, the set of conditional facts does not increase, and after all other rules are eliminated, $\mathrm{lfp}(T_P)$ remains. □

5. PROOF OF LEMMA 4.1, LEMMA 4.2, AND THEOREM 4.2

Lemma 4.1. For every program P, the operator T_P is correct and complete in the following sense:

1. *Every conditional fact contained in $\mathrm{lfp}(T_P)$ is a logical consequence of P (here again, P is viewed as a classical theory).*

2. *$\mathrm{lfp}(T_P)$ contains all conditional facts $\mathcal{A} \leftarrow not\, \mathcal{C}$, which are logical consequences of P and which are minimal (where "minimal" means that there is no $\mathcal{A}' \leftarrow not\, \mathcal{C}'$, which is also a logical consequence and satisfies $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$, where at least one "$\subseteq$" is proper).*

Proof. The immediate consequence operator T_P for conditional facts is very similar to the hyperresolution operator. More specifically, it is identical in the case of positive logic programs. Let us define $\tilde{P}$ for a logic program P to be the positive logic program arising from the translation of every rule

$$A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \mathit{not}\, C_1 \wedge \cdots \wedge \mathit{not}\, C_n$$

into

$$A_1 \vee \cdots \vee A_k \vee \tilde{C}_1 \vee \cdots \vee \tilde{C}_n \leftarrow B_1 \wedge \cdots \wedge B_m,$$

where $\tilde{C}_i$ is an atom with a new predicate symbol. Now let $\tilde{\Gamma} := \mathrm{lfp}(T_{\tilde{P}})$, i.e. the disjunctive facts derivable by hyperresolution. Then the translation back into conditional facts gives exactly $\Gamma = \mathrm{lfp}(T_P)$: Since the new atoms did not occur in any rule body, they did not participate in the resolution.

But for hyperresolution we know a correctness and completeness result, namely that it only computes disjunctive facts which are logical consequences of $\tilde{P}$, and that it computes at least all minimal such disjunctive facts (minimal with respect to $\subseteq$ if we view such facts as sets of atoms): See, e.g., [Bra94]. □

Lemma 4.2. Let $P_1 \mapsto_U P_2$. Then P_1 and P_2 have the same minimal models.

Proof. So let $P_1 \mapsto_U P_2$ and suppose that unfolding was applied to the atom B in the rule $\mathcal{A} \leftarrow \big(\mathcal{B} \cup \{B\}\big) \cup \mathit{not}\, \mathcal{C}$.

In order to show that P_1 and P_2 have the same minimal models, we first show that a minimal model of P_1 is also a model of P_2 and vice versa:

- Let I be a model of P_1 (we do not need the minimality in this direction). Since the combined rules are logical consequences of the old rules, I is also a model of P_2.

- Let now I be a minimal model of P_2. The only rule of P_1 which is not also contained in P_2 (and thus could be possibly violated) is $\mathcal{A} \leftarrow \big(\mathcal{B} \cup \{B\}\big) \cup \mathit{not}\, \mathcal{C}$. Suppose that I would violate this rule. Then every body literal including B would be true in I and every head literal would be false. Consider the interpretation I_0 with $I_0 \not\models B$, but which otherwise agrees with I. Since I is a minimal model of P_2 and I_0 is smaller, I_0 cannot be a model of P_2. Since I_0 differs only in the truth value of B from the model I of P_2, it must violate a rule with B in the head (remember that our models assign independent truth values to positive and negative literals, so the truth value of $\mathit{not}\, B$ is the same in I and I_0). This rule must already be contained in P_1 (if it were one of the rules resulting from the unfolding step, we had $B \in \mathcal{A}$, contradicting the assumption that I violates the unfolded rule). So the rule which I_0 violates is one of the rules about B in P_1, say $\mathcal{A}' \leftarrow \mathcal{B}' \wedge \mathit{not}\, \mathcal{C}'$. This means that $\mathcal{B}' \wedge \mathit{not}\, \mathcal{C}'$ is true in I_0 and thus in I. Furthermore $\mathcal{A}'$ is false in I_0, and thus $\mathcal{A}' - \{B\}$ is false in I. But then I violates the combined rule:

  $$\mathcal{A} \cup \big(\mathcal{A}' - \{B\}\big) \leftarrow (\mathcal{B} \cup \mathcal{B}') \wedge (\mathcal{C} \wedge \mathcal{C}').$$

  This is impossible, since I was assumed to be a model of P_2.

Now we show that the minimal models agree. We only show the direction that a minimal model of P_1 is also a minimal model of P_2. The other direction is completely analogous with P_1 and P_2 interchanged.

Let I be a minimal model of P_1. We have shown above that it is a model of P_2. If there were a smaller model I' of P_2, there would also be a minimal model I'' of P_2 smaller than (or equal to) I' and thus smaller than I. But as shown above, I'' would be a model of P_1, contradicting the assumed minimality of I. □

Lemma E.1. Let P_1 and P_2 be distinct ground programs which are both irreducible with respect to unfolding and elimination of nonminimal rules. Then there is an interpretation which is a minimal model of one of these programs, but not of the other.

PROOF. Since unfolding is not applicable to P_1 and P_2, they cannot contain rules with positive body literals. So P_1 and P_2 consist only of conditional facts. Since the programs are distinct, there must be a conditional fact $\mathcal{A} \leftarrow \mathit{not}\ \mathcal{C}$, which is contained only in one program, but not in the other. It is possible to choose $\mathcal{A} \leftarrow \mathit{not}\mathcal{C}$ such that the other program does also not contain a conditional fact $\mathcal{A}' \leftarrow \mathit{not}\,\mathcal{C}'$ with $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$ (if there were such a $\mathcal{A}' \leftarrow \mathit{not}\,\mathcal{C}'$, we would use it as the distinguishing conditional fact: the first program cannot contain $\mathcal{A}'' \leftarrow \mathit{not}\ \mathcal{C}''$ with $\mathcal{A}'' \subseteq \mathcal{A}'$ and $\mathcal{C}'' \subseteq \mathcal{C}'$ since then $\mathcal{A}'' \subseteq \mathcal{A}$ and $\mathcal{C}'' \subseteq \mathcal{C}$ contradicting the irreducibility with respect to the elimination of nonminimal rules.).

Without loss of generality let us assume that P_1 contains $\mathcal{A} \leftarrow \mathit{not}\mathcal{C}$, and P_2 does not contain any conditional fact $\mathcal{A}' \leftarrow \mathit{not}\,\mathcal{C}'$ with $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$. Then let I be the interpretation which $\mathit{not}\,\mathcal{C}$ true, all other negative literals false, and $\mathcal{A}$ false, and all other positive literals true. Obviously, I is a model of P_2: if a conditional fact $\mathcal{A}' \leftarrow \mathit{not}\,\mathcal{C}'$ in P_2 were violated, it would satisfy $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$, but such conditional facts do not occur in P_2.

Now let I_0 be a minimal model of P_2 smaller than (or equal to) I. But I_0 violates the conditional fact $\mathcal{A} \leftarrow \mathit{not}\ \mathcal{C}$ in P_1: the negative body literals are interpreted as in I, so $\mathit{not}\ \mathcal{C}$ is true, and the interpretation of the positive literals can only switch from true to false, so $\mathcal{A}$ remains false. □

Theorem 4.2 (Semantics Looking only at Minimal Models). A semantics $\mathcal{S}$ allows unfolding (GPPE), elimination of tautologies, and elimination of nonminimal rules if and only if if satisfies the following condition:

$\mathcal{S}(P_1) = \mathcal{S}(P_2)$ *for all programs P_1 and P_2*
which have the same set of minimal models.

PROOF.

1. Let $\mathcal{S}$ be a semantics which allows unfolding, elimination of tautologies and elimination of nonminimal rules. Further let P_1 and P_2 be any logic programs which have the same set of minimal models. For $i = 1, 2$, let $\Gamma_i := \mathrm{lfp}(T_{P_i})$ and Γ_i' result from Γ_i by elimination of all nonminimal rules. Then $\mathit{ground}(P_i) \mapsto^*_{\mathrm{mmod}} \Gamma_i'$ by Theorem 4.1. Since minimal models are not changed by unfolding (Lemma 4.2), elimination of tautologies and elimination of nonminimal rules, Γ_1' has the same minimal models as P_1, and Γ_2' has the same minimal models as P_2. But since P_1 and P_2 agree in their minimal models, so do Γ_1' and Γ_2'. But then Lemma E.1 tells us that $\Gamma_1' = \Gamma_2'$. Since $\mathcal{S}$ also allows $\mapsto^*_{\mathrm{mmod}}$, we get $\mathcal{S}(P_1) = \mathcal{S}(\Gamma_1') = \mathcal{S}(\Gamma_2') = \mathcal{S}(P_2)$.

2. Suppose that $\mathcal{S}$ looks only at the minimal models of the programs. Then

it obviously allows unfolding, elimination of tautologies, and elimination of nonminimal rules because these transformations do not change the set of minimal models of a program: for unfolding, this was proven in Lemma 4.2. The other transformations are equivalence transformations, which do not change the models at all. □

F. PROOF OF THEOREM 4.3

Lemma F.1. Let P_1 and P_2 be ground logic programs with $P_1 \mapsto_{\text{mmod}} P_2$ (i.e. P_2 results from unfolding, weak unfolding, elimination of tautologies or elimination of nonminimal rules). Then $res(P_1) = res(P_2)$.

PROOF. Let Γ_1 be the conditional facts derivable from P_1, i.e. $\Gamma_1 := \text{lfp}(T_{P_1})$, and Γ_2 be those derivable from P_2. In the case of unfolding, Lemma 4.2 tells us that P_1 and P_2 have the same minimal models. The other transformations $\mapsto_W$, $\mapsto_T$, $\mapsto_M$ are equivalence transformations and do not change the models at all. Then, by Lemma 4.1, Γ_1 and Γ_2 contain the same minimal conditional facts, although they might differ in nonminimal conditional facts (see Example B.1).

Let Γ_0 result from Γ_1 by removing all nonminimal facts. Then we obviously have $\Gamma_1 \mapsto_M^* \Gamma_0$, and, as shown above, $\Gamma_2 \mapsto_M^* \Gamma_0$. But now Lemma 3.3 gives us $res(\Gamma_1) = res(\Gamma_0) = res(\Gamma_2)$ and therefore $res(P_1) = res(P_2)$. □

Theorem 4.3 (Invariance of Residual Program). For all ground programs P_1, P_2:

$$P_1 \mapsto_{\text{all}} P_2 \implies res(P_1) = res(P_2).$$

I.e. the residual program is invariant under the elementary program transformations considered in this paper.

PROOF. Lemma F.1 contains the proof for $\mapsto_U$, $\mapsto_W$, $\mapsto_T$, and $\mapsto_M$. So only positive and negative Reduction are missing. Let $P_1 \mapsto_N P_2$ or $P_1 \mapsto_P P_2$. Further, let $\Gamma_1 = \text{lfp}(T_{P_1})$ and $\Gamma_2 = \text{lfp}(T_{P_2})$. We show that then $\Gamma_1 \mapsto_N^* \Gamma_2$ resp. $\Gamma_1 \mapsto_P^* \Gamma_2$. The intuitive reason is that negative body literals are attached to every conditional fact which is derived using the rule directly or indirectly. So instead of evaluating it to true or false before the derivation of implied conditional facts, we can also evaluate it later in every resulting conditional fact.

1. Let $P_1 \mapsto_N P_2$ and let $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}$ be the rule deleted because there is a disjunctive fact $\mathcal{A}' \leftarrow \mathit{true}$ in P_1 with $\mathcal{A}' \subseteq \mathcal{A}$. It is easy to show by induction on the number i of derivation steps that every conditional fact $\mathcal{A}'' \leftarrow \mathit{not}\, \mathcal{C}''$ in $T_{P_1} \uparrow i - T_{P_2} \uparrow i$ also satisfies $\mathcal{A}' \subseteq \mathcal{C}''$. On the other hand, we obviously have $T_{P_2} \uparrow i \subseteq T_{P_1} \uparrow i$. Therefore $\Gamma_1 \mapsto_N^* \Gamma_2$.

2. Let $P_1 \mapsto_P P_2$ and let $\mathit{not}\ C$ be the negative literal evaluated to true. It is easy to show by induction on the number i of derivation steps that for every conditional fact $\mathcal{A}_1 \leftarrow \mathit{not}\ \mathcal{C}_1$ in $T_{P_1} \uparrow i$, either $\mathcal{A}_1 \leftarrow \mathit{not}\ \mathcal{C}_1$ or $\mathcal{A}_1 \leftarrow \mathit{not}\ \big(\mathcal{C}_1 - \{C\}\big)$ or both appear in $T_{P_2} \uparrow i$. Vice versa, for every conditional fact $\mathcal{A}_2 \leftarrow \mathit{not}\, \mathcal{C}_2$ in $T_{P_2} \uparrow i$, either itself or $\mathcal{A}_2 \leftarrow \mathit{not}\ \big(\mathcal{C}_2 \cup \{C\}\big)$ or both appear in $T_{P_1} \uparrow i$. Therefore $\Gamma_1 \mapsto_P^* \Gamma_2$.

Now Lemma 3.3 yields $res(\Gamma_1) = res(\Gamma_2)$, and therefore $res(P_1) = res(P_2)$. □

G. PROOF OF LEMMA 4.3 AND THEOREM 4.4

Lemma 4.3. If I is a normal model of $res(P)$, it is also a normal model of P.

PROOF. We show that for all ground programs $P_1 \mapsto_{\text{all}} P_2$ and consistent interpretations I, if I is a normal model of P_2 then it is also a normal model of P_1.

1. Elimination of tautologies and elimination of nonminimal rules and weak unfolding do not change the set of models, and unfolding does not change the set of minimal models by Lemma 4.2. Furthermore, these transformations never introduce new head literals, so the condition that atoms not occurring in the head must be interpreted as false can only become stronger (thus, if it is satisfied for P_2, it is automatically satisfied for P_1). Finally, disjunctive facts are only deleted if there still is a stronger disjunctive fact.

2. Now let us consider positive reduction and let $not\ C$ be the negative body literal deleted from the rule $\mathcal{A} \leftarrow \mathcal{B} \wedge not\ \mathcal{C}$. Positive reduction makes the rule stronger, so I is certainly a model of P_1. Since C does not occur in any rule head in P_1, we also have $C \notin heads(P_2)$ and thus $I \models not\ C$. But then, if there were a smaller model I_0 of P_1, it would also satisfy $I_0 \models not\ C$, and thus it would also be a model of P_2 contradicting the assumed minmimality of I.

   As the above transformations, positive reduction can make the additional restrictions on the interpretation of negative literals only stronger.

3. Finally, suppose that P_2 results form negative reduction and let $\mathcal{A} \leftarrow \mathcal{B} \wedge not\mathcal{C}$ be the rule deleted because there was a disjunctive fact $\mathcal{A}' \leftarrow true$ in P_1 with $\mathcal{A}' \subseteq \mathcal{C}$. Since I is a normal model of P_2, and $\mathcal{A}' \subseteq \mathcal{C}$ is also contained in P_2, we know that I must make the negation of one of the atoms in $\mathcal{A}'$ false, and thus I is automatically a model of P_1. Furthermore, since $P_2 \subseteq P_1$, any smaller model of P_1 would also be a model of P_2 contradicting the minimality of I. Finally, by removing a rule (with non-empty body), the additional restrictions can only become stronger.

By Corollary 4.2, we have $ground(P) \mapsto^*_{\text{all}} res(P)$. Thus, we can show by induction on the length of the derivation that I is also a normal model of $ground(P)$ (and thus P), where the above result is the inductive step. □

Theorem 4.4 (Closure under Logical Consequences). For every program P and pure ground disjunction Q:

$P \cup D\text{-}WFS(P) \vdash Q \implies Q \in D\text{-}WFS(P)$.

PROOF. We show that if $Q \notin$ D-WFS(P), then $P \cup$ D-WFS$(P) \nvdash Q$. We do this by constructing a normal model I of $res(P)$ with $I \not\models Q$.

1. Let Q be a disjunction of negative ground literals: $not\ A_1 \vee \cdots \vee not\ A_n$. Since $Q \notin$ D-WFS(P), every A_i appears in a rule head in $res(P)$. Let I_0 be the following interpretation:

   - $I_0 \models not\ A$ iff $A \notin heads\big(res(P)\big)$,
   - $I_0 \models A$ iff $A \in heads\big(res(P)\big)$.

Obviously, I_0 is a model of $res(P)$. Let I be a minimal model of $res(P)$ which is smaller than (or equal to) I. Then I is a normal model of $res(P)$, so by Lemma 4.3 it is a model of P. By construction, it is a model of D-WFS(P). Furthermore, $I \not\models \mathit{not}\ A_i$ for $i = 1, \ldots, n$, since $A_i \in \mathit{heads}\bigl(res(P)\bigr)$.

2. Let Q be a disjunction of positive ground literals. Since $Q \notin$ D-WFS(P), there is no disjunctive fact $\mathcal{A} \leftarrow \mathit{true}$ in $res(P)$ with $\mathcal{A} \subseteq Q$. Let I_0 be the following interpretation:

   - $I_0 \models \mathit{not}\ A$ iff $A \notin \mathit{heads}\bigl(res(P)\bigr)$,
   - $I_0 \models A$ iff $A \in \mathit{heads}\bigl(res(P)\bigr) - Q$.

   Since all negative body literals are false in this interpretation, and query is not subsumed by a disjunctive fact, this interpretation is obviously a model of $res(P)$. Let I be again a smaller minimal model of $res(P)$. Since already $I_0 \not\models Q$, of course also $I \not\models Q$. By construction, it is also clear that I is a normal model of $res(P)$, and thus a model of P. Furthermore, it obviously satisfies D-WFS(P). $\square$

H. PROOF OF THEOREMS FROM SECTION 5

Theorem 5.1 (WFS allows our Transformations). For non-disjunctive ground programs P_1 and P_2, if $P_1 \mapsto_{\text{all}} P_2$, then P_1 and P_2 have the same well-founded model.

PROOF.

1. The alternating fixpoint construction of the well-founded model [VG89], immediately shows that the well-founded semantics has the property that it looks only at the minimal models and not at the syntax of the rules (each application of the fundamental $\mathbf{S}_P$-operator in [VG89] constructs one minimal model for a specific interpretation of the negative literals). So Theorem 4.2 becomes applicable and shows that WFS allows unfolding, elimination of tautologies, and elimination of nonminimal rules (plus weak unfolding by Lemma 2.1).

2. Let us show that the well-founded semantics allows positive reduction. Let $P_1 \mapsto_P P_2$ and let $\mathit{not}\ C$ be the negative literal evaluated to true because C appears in no rule head. Note that for any interpretation I satisfying $I \models \mathit{not}\ C$, we have:

   $$I \text{ is a minimal model of } P_1 \iff I \text{ is a minimal model of } P_2.$$

   We prove that any set of negative ground literals I^-, such that the minimal model of P_1 with this interpretation of the negative literals is consistent, satisfies the following:

   $$I^- \text{ is a fixpoint of } \mathbf{A}_{P_1} \iff I^- \text{ is a fixpoint of } \mathbf{A}_{P_2},$$

   where $\mathbf{A}$ is the alternating fixpoint operator defined in [VG89]. Of course, this implies that the least fixpoints agree (the consistency requirement is known to hold for the least fixpoint, i.e. the well-founded model).

So suppose that I^- is a fixpoint of $\mathbf{A}_{P_1}$. Since C does not appear in any rule head, already the first iteration of $\mathbf{A}_{P_1}$ makes *not* C true, so it is certainly contained in I^-. But for such interpretations of the negative literals, P_1 and P_2 have identical minimal models. Therefore, $\mathbf{S}_{P_1}(I^-) = \mathbf{S}_{P_2}(I^-)$. By the consistency requirement, $\mathbf{S}_{P_1}(I^-)$ does not contain C, so the minimal models considered in the next application of $\mathbf{S}_{P_1}$ and $\mathbf{S}_{P_2}$ are again identical. Thus, $\mathbf{A}_{P_2}(I^-) = \mathbf{A}_{P_1}(I^-) = I^-$, i.e. I^- is also a fixpoint of $\mathbf{A}_{P_2}$.

The opposite direction, "I^- is a fixpoint of $\mathbf{A}_{P_2} \implies I^-$ is a fixpoint of $\mathbf{A}_{P_1}$", is shown analogously.

3. The proof for negative reduction is very similar. Note that negative reduction is simpler in the non-disjunctive case than in the general case, since it suffices to consider a single negative literal *not* C. Again, minimal models agree if they only interpret the *not* C as false. □

Theorem 5.2 (D-WFS extends WFS). Let P be a non-disjunctive program and Q be a (positive or negative) ground literal. Then we have:

$$Q \in \textit{D-WFS}(P) \iff Q \in \textit{WFS}(P).$$

Proof. Since the WFS allows our transformations (Theorem 5.1), we can conclude from Corollary 4.3 that the well-founded model of the residual program is equal to the well-founded model of the original program (for this result see also [AD95]). Note that our transformations never introduce new disjunctions, so it is no problem that WFS is only defined on non-disjunctive programs.

Let us write D-WFS$'$ for the set of ground literals contained in D-WFS. Because of the special form of the residual program, it is easy to show D-WFS$'\big(res(P)\big) = WFS\big(res(P)\big)$:

1. The direction $\subseteq$ is obvious: any fixpoint I^- of $\mathbf{A}_{res(P)}$ must contain all *not* A, where $A \notin heads\big(res(P)\big)$. Furthermore, $\mathbf{S}_{res(P)}(I^-)$ certainly contains all atoms A which are facts in $res(P)$.

2. In order to prove that the well-founded semantics makes all other ground atoms A undefined, we show that $I^- := \big\{ not\ A \bigm| A \notin heads\big(res(P)\big)\big\}$ is already a fixpoint of $\mathbf{A}_{res(P)}$ (and then it is certainly the least):

   $\mathbf{S}_{res(P)}(I^-)$ contains only the atoms, which are given as facts in $res(P)$. But then the negations of all other atoms are assumed and this allows us to derive all atoms in $heads\big(res(P)\big)$. Taking the complement gives us again I^-.

So we have for all non-disjunctive programs P: $WFS(P) = WFS\big(res(P)\big) =$ D-WFS$'\big(res(P)\big) =$ D-WFS$'(P)$. □

Theorem 5.3 (STABLE allows our Transformations). For all ground programs P_1 and P_2, if $P_1 \mapsto_{\text{all}} P_2$, then P_1 and P_2 have the same set of stable models.

Proof.

1. The stable model semantics looks only at minimal models: it simply selects those among the minimal models which are total and consistent (i.e. "two-valued"). So Theorem 4.2 becomes applicable and shows that STABLE allows unfolding, elimination of tautologies, and elimination of nonminimal

rules (plus weak unfolding by Lemma 2.1). We have proven these properties already in [BD97]. Unfolding was independently established in [SS97].

2. Positive Reduction: let $P_1 \mapsto_P P_2$ and let *not* C be the negative literal evaluated to true.

   Let I be a stable model of P_1. Since C occurs in no rule head, $I \models \mathit{not}\, C$, and thus I is also a model of P_2. There can be no smaller model, since every model of P_2 is also a model of P_1 and I was assumed to be stable and thus minimal.

   Conversely, let I be a stable model of P_2. It is clear that I is also a model of P_1. Furthermore, $I \models \mathit{not}\, C$, so if there were a smaller model I_0 of P_1, this would also be a model of P_2, contradicting again the minimality of I.

3. Negative Reduction: let $P_1 \mapsto_N P_2$ and let $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}$ be the rule which was deleted because of the fact $\mathcal{A}' \leftarrow \mathit{true}$ with $\mathcal{A}' \subseteq \mathcal{A}$.

   Let I be a stable model of P_1. Since $P_2 \subseteq P_1$, I is also a model of P_2. If there were a smaller model I_0 of P_2, this would also be a model of P_1 since $I \not\models \mathit{not}\, \mathcal{C}$ and thus $I_0 \not\models \mathit{not}\, \mathcal{C}$. This would contradict the assumed minimality of I.

   Let I be a stable model of P_2. Then $I \not\models \mathit{not}\, \mathcal{C}$, thus I is also a model of P_1. If there were a smaller model I_0, this would also be a model of P_2 (since $P_2 \subseteq P_1$), contradicting again the minimality of I. □

Theorem 5.4 (D-WFS extends GCWA). Let P be a positive program and Q be a negative ground literal. Then we have

$Q \in \mathit{D\text{-}WFS}(P) \iff Q \in \mathrm{GCWA}(P)$.

PROOF. Given a positive program P, positive and negative reduction are obviously not needed in the construction of the residual program, so it can be reached by unfolding, weak unfolding, elimination of tautologies and elimination of non-minimal rules. These transformations do not change the set of minimal models and therefore they do not change the negative literals assumed by the GCWA. So it suffices to consider residual programs which in this case are sets of minimal positive disjunctions.

1. Let $\mathit{not}\, A \in$ D-WFS(P). The D-WFS assumes *not* A only if A does not appear in $\mathit{res}(P)$. Then A is of course false in all minimal models, and thus the GCWA assumes *not* A.

2. Let $\mathit{not}\, A \notin$ D-WFS(P). Then A appears in a disjunction in $\mathit{res}(P)$, say $A \vee A_1 \vee \cdots \vee A_n$. Let I be the interpretation which makes $A_1, \ldots, A_n$ false, and all other atoms true. Since $A \vee A_1 \vee \cdots \vee A_n$ is a minimal disjunction, I is a model of $\mathit{res}(P)$. Now let I_0 be a smaller minimal model. Since A must be true in I_0, *not* A cannot be assumed by the GCWA. □