Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA

Stefan Brass
Universität Hannover
Institut für Informatik
Lange Laube 22, D-30159 Hannover
Germany, E-mail: `sb@informatik.uni-hannover.de`

Jürgen Dix
Universität Koblenz
Institut für Informatik
Rheinau 1, D-56075 Koblenz
Germany, E-mail: `dix@informatik.uni-koblenz.de`

Abstract. Recently Brass and Dix have introduced the semantics D-WFS for general disjunctive logic programs. The interesting feature of this approach is that it is both *semantically* and *proof-theoretically* founded. Semantically, D-WFS is invariant under some natural declarative principles. Proof-theoretically, any program Φ is associated a normalform $res(\Phi)$, called the *residual* program, by a non-trivial bottom-up construction using least fixpoints of two monotonic operators.

We show in this paper, that the original calculus, consisting of some simple transformations, has a very strong and appealing property: it is *confluent* and terminating. This means that all the transformations can be applied in any order: whenever we arrive at an irreducible program (no more transformation is applicable) this program is already *uniquely* determined and coincides with the normalform $res(\Phi)$. Moreover, for *fair* sequences it is also *strongly terminating*: every fair sequence of transformations leads to normalform $res(\Phi)$. Another feature of our approach is that D-WFS can be read off from $res(\Phi)$ immediately in a very simple way. No proper subset of the calculus has these properties — only when we restrict to certain subclasses of programs.

We also give an equivalent characterization of D-WFS in terms of iterated minimal model reasoning with respect to positive programs. This construction is a generalization of a description of the wellfounded semantics: we introduce a very simple and neat construction of a sequence D_i that eventually stops and represents the set of derivable disjunctions.

Both characterizations open the way for efficient implementations. The first because the ordering of the transformations does not matter: we are free to choose always the "best" transformation, which maximally reduces the program. The second because special methods from Circumscription, in particular sophisticated minimal model reasoner for positive programs, might be useful.

Key words: Logic Programming, Non-Monotonic Reasoning, Disjunctive Semantics.

CR codes: I.2.3

1. Introduction

The success of PROLOG as one of the major programming languages for *Artificial Intelligence* suggests that extensions of Horn-clauses to general clauses with negation and disjunction might become very useful in *Knowl-*

edge Representation and Reasoning. Indeed, in recent years much work has been done in *Logic Programming* to extend semantics based on different forms of negation-as-failure from non-disjunctive to *disjunctive* semantics. Many interrelationships of non-monotonic logics (Default-Logic DL, Circumscription CIRC, Autoepistemic-Logic AEL) with semantics of logic programs have been discovered during the last years [1, 23, 31, 15]. Eiter and Gottlob have shown in [26, 27] that disjunctive semantics are strictly more expressive than non-disjunctive semantics and therefore may be better suited for many knowledge representation tasks (see also [2, 16] for using such semantics in KR).

One of the main problems today is that although we have a quite good understanding of non-disjunctive semantics, our knowledge of disjunctive semantics is quite limited. Besides the main competing approaches of the wellfounded semantics WFS (which is often too weak) and the stable semantics STABLE (which is sometimes too strong) there exist many extensions of WFS that approximate STABLE. Unfortunately, of all these semantics only STABLE has a straightforward extension to disjunctive programs: not even for WFS does there exist a canonical disjunctive version. We claim that our D-WFS is this counterpart of WFS.

The novelty of our approach is that it is not exclusively declarative (like Przymusinski's stationary or static approach[1] [37, 38]) nor exclusively procedural (like the approaches of Minker and his group[2] [4, 3, 29]).

We introduced in [8, 11] a calculus of program transformations. These can be seen as *declarative* because they express precise semantical properties (e.g. *partial evaluation*). But they are also *procedural* because they can be applied to a program and transform it to another one, which is much simpler and sometimes immediately allows us to read off the set of derivable disjunctions (for example, if an atom a does not appear in any head, then its negation must be derivable). We also defined a particular *bottom-up procedure* using two operators R and T that enabled us to associate a normalform

$$res(\Phi)$$

to any program Φ.

In this paper we show how this construction can be avoided and the normalform can be obtained just by applying our transformations directly: they constitute a *confluent* calculus. This means that (almost) any sequence of transformations leads eventually to an irreducible program (where no transformation can be applied) and this irreducible program is *uniquely deter-*

[1] The original static semantics has been defined modeltheoretically in terms of *static expansions*. Of course, suitable characterizations of the least static expansion are procedural in nature but the overall approach is not.

[2] Most of these approaches are defined by certain modifications of model-elimination calculi, but not by modeltheoretical means.

mined. It coincides with the normalform $res(\Phi)$ from which the semantics D-WFS can be easily read off (Theorem 4.2).

This is a deep result that uses in its proof the bottom-up construction very heavily. We also consider subcalculi that are appropriate for restricted classes of programs like the class of positive disjunctive or general nondisjunctive programs. Finally, we give another characterization of D-WFS in terms of sucessive construction of minimal models. Thus our main results are

- Our calculus is *confluent* and *strongly terminating* for fair sequences. The irreducible program coincides with the residual program $res(\Phi)$ of Φ introduced in [8] and immediately determines the D-WFS semantics of Φ (Theorem 4.2).

- D-WFS can be equivalently described using iterated minimal model reasoning, which relates it to circumscription and makes it accessible for special theorem provers that handle positive theories.

We restrict this paper to finite propositional programs, although the original construction of the normalform of a program also holds for the wider class of allowed DATALOG programs. Let us shortly explain this. Such a restriction is not needed for approaches based on models. If we consider the program

$$P(x) \leftarrow P(f(x))$$

or, equivalently, the infinite propositional program

$$p_0 \leftarrow p_1,\ p_1 \leftarrow p_2,\ \ldots, p_i \leftarrow p_{i+1}, \ldots$$

then by looking at all minimal models of these programs, we immediately get $\neg P(t)$ for all terms t, resp. $\neg p_i$ for all $i \in \mathbb{N}$. But no proof-theoretic approach, that is based on efficent program-transformations and associates to any program a normalform in a constructive way (like D-WFS) can "unfold" such an infinite loop and eliminate all these rules. To do this, something like a ω-inference rule is needed, which is not a constructive rule[3]. But if the above loop is finite, this can be recognized in a finite number of steps. Indeed, D-WFS will do this. Therefore a completeness result of a constructive procedure can only hold for the propositional case — or we had to modify the construction of the normalform $res(\Phi)$ and thus departing from constructive methods. The interested reader is referred to [24, 25] where we extend our ideas to the first-order case by introducing *constraint disjunctive programs*.

Our paper is structured as follows. We first introduce some notation and terminology in Section 2. In Section 3, we repeat the very definition of semantics D-WFS (given in [8, 11]) as the *weakest* semantics satisfying certain

[3] Of course, any particular example can be handled by an appropriate rule — a *regularity* condition or constructive *tabling techniques* could help. But since the general problem is undecidable, there is no constructive solution in general.

abstract conditions. In Section 4 we prove one of our main theorems, namely the strong result that the set of transformations is confluent and terminating. The proof is not trivial and uses heavily previous results, in particular the normalform of a program introduced earlier. In Section 5 we consider other semantics and appropriate calculi for them. In particular we consider the stable semantics STABLE and semantics defined on restricted classes of programs like WFS for nondisjunctive and GCWA for positive disjunctive programs. Section 6 contains another equivalence result. We introduce semantics SEM^* and SEM_0 which do not involve any particular technical machinery. They are based on minimal consequences of positive programs and use a generalized Gelfond-Lifschitz transformation. The construction is very direct and easy, but gives no insights into the properties and general behaviour of these semantics. We show that D-WFS is sound w.r.t. both SEM^* and SEM_0 and also complete w.r.t. SEM_0. Therefore, D-WFS coincides with SEM_0 and can be computed by a minimal model reasoner for positive programs. Finally, we conclude with Section 7.

2. Preliminaries

In this section we present the language $\mathcal{L}$ and introduce all necessary formal notions that we need in the sequel.

As already stated in the introduction, we consider finite, instantiated programs over some fixed finite language $\mathcal{L}$ containing the binary connectives $\vee, \wedge, \rightarrow$, the unary connective $\neg$ and the *falsum* $\perp$. We write $At_{\mathcal{L}}$ for the set of all atoms different from $\perp$.

Definition 2.1 (Logic Program Φ, Possibly True Facts $\mathcal{P}(\Phi)$).
A logic program Φ is a finite set of rules of the form

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

where the $A_i/B_i/C_i$ are $\mathcal{L}$-atoms different from $\perp$, $k \geq 1$, $m \geq 0$, $n \geq 0$. We allow empty conjunctions: they are considered as abbreviations for $\neg\perp$ (the *verum*).

We identify such a rule with the triple consisting of the following sets of atoms $\mathcal{A} := \{A_1, \ldots, A_k\}$, $\mathcal{B} := \{B_1, \ldots, B_m\}$, $\mathcal{C} := \{C_1, \ldots, C_n\}$, and write it as $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C}$. This means, in particular, that we assume the A_i (resp. the B_i, resp. the C_i) to be pairwise distinct.

We write $\mathcal{P}(\Phi)$ for the set of all atoms occurring in rule heads in Φ: these are atoms that are *possibly true*. By *pure* disjunctions we mean disjunctions consisting solely of positive or solely of negative literals.

3. Disjunctive Well-Founded Semantics: D-WFS

A logic program or a deductive database is used by posing and answering *queries* about its contents. While there is no need to consider conjunctive queries (since $\psi_1 \wedge \psi_2$ is answered with "yes" iff both subqueries are answered "yes"), we need disjunctive queries ψ, since the result of $\psi_1 \vee \psi_2$ cannot be derived from the results of the single queries. Because many semantics for disjunctive logic programs do not derive *mixed* disjunctions (i.e. containing negative and positive literals at the same time) we consider the following very general definition:

Definition 3.1 (Operator $\mid\!\sim$, Semantics $\mathcal{S}_{\mid\sim}$).
By a semantic operator $\mid\!\sim$ we mean a binary relation between logic programs and pure disjunctions which satisfies the following three arguably obvious conditions:

1. *Right Weakening:* If $\Phi \mid\!\sim \psi$ and $\psi \subseteq \psi'$[4], then $\Phi \mid\!\sim \psi'$.

2. *Necessarily True:* If $\mathcal{A} \leftarrow \mathit{true} \in \Phi$ for a disjunction $\mathcal{A}$, then $\Phi \mid\!\sim \mathcal{A}$.

3. *Necessarily False:* If $A \notin \mathcal{P}(\Phi)$ for a $\mathcal{L}$-ground atom A, then $\Phi \mid\!\sim \neg A$.

Given such an operator $\mid\!\sim$ and a logic program Φ, by the semantics $\mathcal{S}_{\mid\sim}(\Phi)$ of Φ determined by $\mid\!\sim$ we mean the set of all pure disjunctions derivable by $\mid\!\sim$ from Φ, i.e., $\mathcal{S}_{\mid\sim}(\Phi) := \{\psi \mid \Phi \mid\!\sim \psi\}$.

Note that both *model-based* as well as *completion-based* approaches fit well into this framework, because these approaches provide in a natural way a set of *derivable* disjunctions: we simply take the sceptical view (truth in all intended models or in all models of the completion).

In addition to satisfying the general conditions (1)-(3) listed above, we may want a specific semantic operator $\mid\!\sim$ to be invariant under certain *natural* program transformations.

Definition 3.2 (Invariance of $\mid\!\sim$ under a Transformation).
Suppose that a program transformation *Trans* is given. We view such a transformation as a relation (not neccessarily a function) which specifies under what conditions a program Φ is a transformation of another program Φ'. To facilitate notation we also write such a transformation sometimes as a multi-valued function $\mathit{Trans} : \Phi \mapsto \mathit{Trans}(\Phi)$ mapping logic programs into logic programs. In general, a particular transformation can be applied in different ways to a program so that $\mathit{Trans}(\Phi)$ is not uniquely defined.

[4] I. e. ψ is a subdisjunction of ψ'.

We say that the operator $\mid\!\sim$ is invariant under *Trans* (or that *Trans* is a $\mid\!\sim$-equivalence transformation) iff

$$\Phi \mid\!\sim \psi \iff \mathit{Trans}(\Phi) \mid\!\sim \psi$$

for any pure disjunction ψ and any program Φ.

Why do we only consider *pure* disjunctions? This is not only for simplicity or convenience, but also because we are interested in the sceptical viewpoint and therefore we can not assume that any semantics is given as a set of models[5]. For example SLDNF-like semantics usually are given in proof-theoretic terms. In order to be as general as possible (to cover as many semantics as possible in our framework) we only assume that a semantics provides us at least with the notion of deriving arbitrary conjunctions of either positive or either negative literals and of negations of these conjunctions. But these are exactly pure disjunctions. Some semantics simply do not define what it means to *derive* a mixed disjunction.

We now describe several such natural transformations *Trans* which will be later used to define the D-WFS semantics. By abuse of language (and to facilitate reading) we will simply say "$\mid\!\sim$ satisfies *Trans*" meaning that "$\mid\!\sim$ is invariant under *Trans*". We will illustrate all the transformations on the following running example

Example 3.3 (Running Example Φ_0).

$$\begin{array}{llll} \Phi_0 : & A \vee B & \leftarrow & C,\ \neg C, \neg D \\ & A \vee C & \leftarrow & B \\ & C \vee D & \leftarrow & \quad\ \neg E \\ & B & \leftarrow & \quad\ \neg C, \neg D, \neg E \end{array}$$

We begin with partial evaluation in the sense of the "unfolding" operation. It is the "Generalized Principle of Partial Evaluation (GPPE)" [8, 42]:

Definition 3.4 (GPPE).
Semantics $\mathcal{S}_{\mid\sim}$ satisfies GPPE iff it is invariant under the following transformation: *Replace a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C}$ where $\mathcal{B}$ contains a distinguished atom B by the rules*

$$\mathcal{A} \cup (\mathcal{A}_i - \{B\}) \ \leftarrow\ (\mathcal{B} - \{B\}) \cup \mathcal{B}_i \ \wedge\ \neg(\mathcal{C} \cup \mathcal{C}_i)\ (i = 1, \ldots, n)$$

where $\mathcal{A}_i \leftarrow \mathcal{B}_i \wedge \neg\mathcal{C}_i$ $(i = 1, \ldots, n)$ are all the rules with $B \in \mathcal{A}_i$.

[5] For a semantics given as a set of models, we can immediately define what it means for *mixed* disjunctions to be derivable: just those that are true in all intended models.

Note that we are free to select a specific positive occurrence of an atom B and then perform the transformation: this is just to say that the GPPE-transformation is a relation between programs. It is uniquely determined only if we fix a specific occurrence of an atom that we want to replace. The new rules are obtained by replacing B by the bodies of all rules r with head literal B and adding the remaining head atoms of r to the head of the new rule. In our example, we have two possibilities to apply GPPE: one to replace C in the first clause and one to replace B in the second clause. We choose the first possibility and get the following program

$$\begin{array}{llll}\Phi_1: & A \vee B & \leftarrow & B,\ \neg C, \neg D \\ & A \vee B \vee D & \leftarrow & \neg C, \neg D, \neg E \\ & A \vee C & \leftarrow & B \\ & C \vee D & \leftarrow & \neg E \\ & B & \leftarrow & \neg C, \neg D, \neg E\end{array}$$

The next transformation, *Elimination of Tautologies*, states that tautological clauses like $p \leftarrow p$ do not influence the semantics of a logic program. With the help of *Elimination of Non-Minimal Rules* we can get rid of clauses that are subsumed by others. Both transformations together correspond to the "Equivalence" principle of [22]:

Definition 3.5 (Elimination of Tautologies, Non-Minimal Rules). Semantics $\mathcal{S}_{\mid\sim}$ satisfies **a)** the *Elimination of Tautologies*, resp. **b)** the *Elimination of Non-Minimal Rules* iff $\mid\sim$ is invariant under the following transformations:

a) Delete a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C}$ with $\mathcal{A} \cap \mathcal{B} \neq \emptyset$.

b) Delete a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C}$ if there is another rule $\mathcal{A}' \leftarrow \mathcal{B}' \wedge \neg\mathcal{C}'$ with $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{B}' \subseteq \mathcal{B}$, and $\mathcal{C}' \subseteq \mathcal{C}$.

By applying the first transformation we get rid of the first clause and get Φ_2:

$$\begin{array}{llll}\Phi_2: & A \vee B \vee D & \leftarrow & \neg C, \neg D, \neg E \\ & A \vee C & \leftarrow & B \\ & C \vee D & \leftarrow & \neg E \\ & B & \leftarrow & \neg C, \neg D, \neg E\end{array}$$

We get rid of the first clause of Φ_2 by applying the second transformation (because it is subsumed by the last one) and get Φ_3. We now apply GPPE again to eliminate B and get Φ_4:

$$\begin{array}{llll}\Phi_3: & A \vee C & \leftarrow & B \\ & C \vee D & \leftarrow & \neg E \\ & B & \leftarrow & \neg C, \neg D, \neg E\end{array} \qquad \begin{array}{llll}\Phi_4: & A \vee C & \leftarrow & \neg C, \neg D, \neg E \\ & C \vee D & \leftarrow & \neg E \\ & B & \leftarrow & \neg C, \neg D, \neg E\end{array}$$

The last two transformations allow us to do some simple reductions. We want $\neg L$ to be derivable if L appears in no rule head. Therefore, it should be possible to evaluate the body literal $\neg E$ to true, i.e. to delete $\neg E$ from all rule bodies: this is guaranteed by *Positive Reduction*. 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 $\neg A_1 \wedge \cdots \wedge \neg A_k$ is surely false, so the entire rule is useless, and it should be possible to delete it: this gives us *Negative Reduction*.

Definition 3.6 (Positive and Negative Reduction).
Semantics $\mathcal{S}_{\mid\sim}$ satisfies **a)** *Positive*, resp. **b)** *Negative* Reduction iff $\mid\sim$ is invariant under the following transformations:

a) Replace a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C}$ by $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg(\mathcal{C} \cap \mathcal{P}(\Phi))$.

b) Delete a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C}$ if there is a rule $\mathcal{A}' \leftarrow \mathit{true}$ with $\mathcal{A}' \subseteq \mathcal{C}$.

In our example, we can apply *positive Reduction* to obtain Φ_5 and then *negative Reduction* (we have to apply it twice) to obtain Φ_6

$$\Phi_5: \begin{array}{lcl} A \vee C & \leftarrow & \neg C, \neg D \\ C \vee D & \leftarrow & \\ B & \leftarrow & \neg C, \neg D \end{array} \qquad \Phi_6: C \vee D \leftarrow$$

We call a semantics $\mid\sim_1$ weaker than a semantics $\mid\sim_2$ iff for all Φ and ψ "$\Phi \mid\sim_1 \psi$ implies $\Phi \mid\sim_2 \psi$". As shown in [11] there exists the weakest semantics which is invariant under all the natural transformations discussed in this Section.

Theorem 3.7 (D-WFS, [11]).
There exists the weakest semantics $\mathcal{S}_{\mid\sim}$ which is invariant under all the natural transformations introduced in this Section. In other words, there exists the weakest semantics which satisfies the properties of GPPE, Elimination of Tautologies and Non-Minimal Rules, Positive and Negative Reduction. Moreover, this semantics is consistent (i.e. it does not derive a literal A and its negation $\neg A$) and closed under logical consequences (as a set consisting of pure disjunctions). We call it the Disjunctive Well-Founded Semantics, or, briefly, D-WFS.

In our running example, we end up with the program Φ_6

$$\mathrm{Red}^{pos} \circ \mathrm{Red}^{neg} \circ \mathrm{GPPE}_B \circ \text{Non-Min} \circ \mathrm{Taut} \circ \mathrm{GPPE}_C(\Phi_0) = \{C \vee D\}.$$

4. The Confluence of our Transformations

Looking again at Φ_0, it is not clear that a different application of our transformations (in a different ordering) leads to the same result. Our main theorem shows that it is indeed so: at least when we require that the final program is irreducible. The proof is not easy. One way is to use a particular normalform $res(\Phi)$ of a program Φ. The definition of this normalform is somewhat complicated and we refer the reader to [11] for a detailed exposition. In fact, we are able to avoid this construction and show the confluence more directly. But since our transformations will also lead to this normalform, we state the definition of $res(\Phi)$ and the main theorem of [11, 9]:

Definition 4.1 (Residual Program,[11]).
The residual program of a DATALOG-program Φ is $res(\Phi) := R^{\omega}(T^{\omega}_{\Phi}(\emptyset))$.[6]

Theorem 4.2 (Computation of D-WFS wrt $\Phi \mapsto res(\Phi)$,[11]).

$$\Phi \mid\!\sim_{\mathrm{D-WFS}} \psi \iff \begin{array}{l}\text{there is } \mathcal{A} \subseteq \psi \text{ with } \mathcal{A} \leftarrow \mathit{true} \in res(\Phi) \text{ or} \\ \text{there is } \neg A \in \psi \text{ and } A \notin \mathcal{P}(res(\Phi)).\end{array}$$

This theorem is especially important. It tells us that once the residual program $res(\Phi)$ has been produced, the semantics D-WFS can be immediately determined. For our running example, we get that Φ_0 is equivalent to Φ_6 (i.e. Φ_0 can be transformed into Φ_6). Since, by definition of $res(\Phi)$ (see the proof of the next theorem), we also have

$$\Phi_6 = res(\Phi_6) = \{C \vee D\}$$

we immediately get

$$\text{D-WFS}(\Phi_0) = \{\neg A, \neg B, \neg E, C \vee D\}.$$

Note that we can *not* conclude that

$$res(\Phi_0) = \Phi_6.$$

Theorem 4.2 just tells us that under D-WFS both Φ_0 and $res(\Phi_0)$ are equivalent as well as Φ_0 and Φ_6. In general, equivalent programs can of course posess different normalforms (see Section 5 where this is the case for STABLE).

Also if we performed our reductions in a different order then it could be possible to end up in a completely different program. Even the residual program *does not* correspond to the application of our transformations in a particular ordering. Nevertheless, a computation shows that in fact the identity $res(\Phi_0) = \Phi_6$ holds.

[6] Operators R and T will be explained below in the proof of Theorem 4.3.

We emphasize the fact that our transformations were originally introduced to define a semantics with *good* properties. The residual program was then defined to associate any program a normalform from which the semantics can be immediately read off (Theorem 4.2). Our next theorem shows that we can use our transformations not just as abstract properties to ensure *good behaviour* of a semantics, but also to compute the semantics directly.

The program Φ_6 has the property that it can not be further reduced because none of our transformations is applicable. Let us denote by $\Phi \rightarrow \Phi'$ if the program Φ can be transformed into Φ' by applying some of our transformations. We then denote by $\rightarrow^*$ the reflexive and transitive closure of $\rightarrow$. Thus $\Phi \rightarrow^* \Phi'$ means that Φ' can be obtained by Φ by applying a finite number of our transformations in *some* order.

We call Φ *irreducible*, if there is no $\Phi' \neq \Phi$ with $\Phi \rightarrow^* \Phi'$. The following theorem is a strong result and our main new theorem in this section.

Theorem 4.3 (Confluence of the Full Calculus).
Our set of transformations is confluent, i.e. if $\Phi \rightarrow^* \Phi'$ and $\Phi \rightarrow^* \Phi''$ and both Φ', Φ'' are irreducible, then $\Phi' = \Phi''$. In addition, for any program Φ there is an irreducible Φ' with $\Phi \rightarrow^* \Phi'$. Such an irreducible Φ' is exactly the residual program $res(\Phi)$ of Φ from Definition 4.1.

Let us first show that a simple and direct proof is not so simple. Consider the following program Φ and its partial evaluation $GPPE_A(\Phi)$.

$$\begin{array}{lll} \Phi: A & \leftarrow & A, \neg B \\ A \vee C & \leftarrow & \neg E \end{array} \qquad \begin{array}{lll} GPPE_A(\Phi): A & \leftarrow & A, \neg B \\ A \vee C & \leftarrow & \neg B, \neg E \\ A \vee C & \leftarrow & \neg E \end{array}$$

The example shows that if we first apply the *Elimination of Tautologies* to Φ and then GPPE we will get a different result than first applying GPPE and then Taut:

$$\begin{array}{l} GPPE_A(Taut(\Phi)) = \{A \vee C \leftarrow \neg E\} \\ Taut(GPPE_A(\Phi)) = \{A \vee C \leftarrow \neg B, \neg E,\ A \vee C \leftarrow \neg E\} \end{array}$$

In this particular example, a further application of *Elimination of nonminimal Rules* will do the job:

$$GPPE_A \circ Taut(\Phi) = \{A \vee C \leftarrow \neg E\} = \text{Non-Min} \circ Taut \circ GPPE_A(\Phi).$$

To obtain the residual program however, a further application of *positive Reduction* is needed. In more complicated cases, just applying one particular transformation does not suffice. A general set of identities of the form

$$T_1 \circ GPPE \circ Taut \circ T_2 = Taut \circ GPPE$$

does not seem to hold.

Before giving the proof of our main theorem, we need some preparations. The overall idea is to relate a certain *model set* to each program Φ. Some of these models are partly inconsistent, i.e. there may be ground atoms in the models which are both true and false at the same time. After showing the Lemmata 4.5 and 4.6, we are able to prove our confluence result.

Definition 4.4 (Model Set of a Program).
Let Φ be a program, M be the set of all possible ground atoms, and $M^-, M^+ \subseteq M$. Then, (M^-, M^+) is called *minimal model* of Φ wrt. M^- iff M^+ is minimal wrt. set inclusion such that, for all rules $\mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C}$ in Φ, $\mathcal{B} \subseteq M^+$ and $\mathcal{C} \subseteq M^-$ imply $\mathcal{A} \cap M^+ \neq \emptyset$. The *model set* $\mathcal{M}$ of Φ is the maximal set of minimal models of Φ obtained by repeatedly applying the following procedure:

(1) If a literal $l \notin M^+$ for all $(M^-, M^+) \in \mathcal{M}$, then remove all $(M^-, M^+) \in \mathcal{M}$ with $l \notin M^-$ from $\mathcal{M}$.

(2) If for a set of literals L it is $L \cap M^+ \neq \emptyset$ for all $(M^-, M^+) \in \mathcal{M}$, then remove all $(M^-, M^+) \in \mathcal{M}$ with $L \subseteq M^-$ from $\mathcal{M}$.

Lemma 4.5 (Invariance of the Model Set).
The model set of Φ is the same as the one of $Trans(\Phi)$ for any of our transformations introduced in Section 3.

Proof. It is trivial that an application of *Elimination of Nonminimal rules* or of *Elimination of Tautologies* does not change MM(Φ). GPPE just evaluates a positive occurrence of an atom and does not change the set of minimal models as shown in [10]. The invariance under reductions holds because of the additional two conditions to filter out certain inconsistent models. □

Lemma 4.6 (Model Sets of Irreducible Programs).
Let Φ and Φ' be two irreducible programs. Then, $\Phi = \Phi'$ iff Φ and Φ' have the same model sets.

Proof. We show that different irreducible programs have different model sets (and follow closely the proof of this lemma given in [25]). Since Φ and Φ' are different, we can assume w.l.o.g. (eventually we have to exchange Φ with Φ') that there is a conditional fact $\mathcal{A} \leftarrow \neg\mathcal{C}$ in Φ, such that there is no rule $\mathcal{A}' \leftarrow \neg\mathcal{C}'$ in Φ', where (a) $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$, or (b) $\mathcal{A}' \subseteq \mathcal{C}$ and $\mathcal{C}' = \emptyset$.

Now, there is a minimal model (M^-, M^+) of Φ' where $\mathcal{A} \cup \mathcal{C} \subseteq M^-$ holds, because conditions (a) and (b) ensure that there is a minimal model, taking step (2) of Definition 4.4 into account. The other step (1) is not critical, because it can only increase M^-. However, (M^-, M^+) is not a minimal model of Φ, because $\mathcal{A} \leftarrow \mathcal{C}$ cannot be valid. Therefore, the model sets are different. *Proof.* (of Theorem 4.3)

We first show that applying our transformations can never lead to two different irreducible programs. By Lemma 4.5, the model sets of Φ, Φ_1 and Φ_2 are identical. By Lemma 4.6, it follows $\Phi_1 \equiv \Phi_2$, because Φ_1 and Φ_2 are irreducible.

It remains to show that the irreducible program coincides with the original Definition 4.1. Let us first complete the definition of the residual program. To do this, we have to define the two operators R and T. Let us call a *conditional fact* 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 $k \geq 1$ and $m \geq 0$. For a set Γ of conditional facts we define:

$$T_\Phi(\Gamma) := \Big\{ \Big(\mathcal{A}_0 \cup \bigcup_{i=1}^{m} (\mathcal{A}_i - \{B_i\})\Big) \leftarrow \neg\Big(\mathcal{C}_0 \cup \bigcup_{i=1}^{m} \mathcal{C}_i\Big) \;\Big|\; \text{there are a ground instance } \mathcal{A}_0 \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \neg\mathcal{C}_0 \text{ of a rule in } \Phi \text{ and cond. facts } \mathcal{A}_i \leftarrow \neg\mathcal{C}_i \in \Gamma \text{ with } B_i \in \mathcal{A}_i\ (i = 1, \ldots, m) \Big\}.$$

We compute the smallest fixpoint of T_Φ as usual: We start with $\Gamma_0 := \emptyset$ and then iterate $\Gamma_i := T_\Phi(\Gamma_{i-1})$ until nothing changes. This must happen because our program is finite and propositional.
The R-operator is defined as follows:

$$\begin{aligned} R(\Gamma) := \{\mathcal{A} \leftarrow \neg(\mathcal{C} \cap \mathcal{P}(\Gamma)) \mid \mathcal{A} \leftarrow \neg\mathcal{C} \in \Gamma, \text{ and} \\ (1)\ \text{there is no } \mathcal{A}' \leftarrow \mathit{true} \in \Gamma \text{ with } \mathcal{A}' \subseteq \mathcal{C}, \\ (2)\ \text{there is no } \mathcal{A}' \leftarrow \neg\mathcal{C}' \in \Gamma \text{ with } \mathcal{A}' \subseteq \mathcal{A}, \\ \mathcal{C}' \subseteq \mathcal{C} \text{ where at least one } \subseteq \text{ is proper}\}. \end{aligned}$$

It is easy to see that this normalform can be obtained by repeated application of GPPE, Elimination of Tautologies and Elimination of Nonminimal rules (see also Theorem 12 of [9]). □

In the last theorem, we proved that every program can be transformed into an irreducible program. However, we did not discuss if *every sequence of program transformations terminates*, i.e. if our calculus is *strongly terminating*. Already the simple program consisting of just the loop "$A \leftarrow A$" shows a problem. Applying GPPE leads to the same program, so GPPE can be applied infinitely often without leading to the residual program. Of course, in this particular case an application of the *Elimination of Tautologies* immediately leads to the residual program. But still another problem can occur[7]:

$$\begin{aligned} \Phi_{loop}:\ A &\leftarrow B \\ B &\leftarrow A \\ C &\leftarrow A, \neg C \end{aligned}$$

[7] brought to our attention by Frieder Stolzenburg

If we apply GPPE_A to the third clause, this clause is replaced by $C \leftarrow B, \neg C$. We can now apply GPPE_B again to this clause and get the original program. So we have an oscillation

$$GPPE_B \circ GPPE_A(\Phi_{loop}) = \Phi_{loop}$$

To summarize, not every sequence of transformations terminates. We therefore introduce the following

Definition 4.7 (Fair Sequences).
We call a sequence of program transformations *fair*, if in the corresponding sequence of programs

1. every positive body-atom is eventually removed (either by removing the whole clause using a suitable transformation or by an application of GPPE), and

2. every tautology clause is eventually removed (either by applying *Elimination of Tautologies* or another suitable transformation).

As the following theorem shows, the above exceptions are the only ones:

Theorem 4.8 (Strong Termination for Fair Sequences).
Our calculus of transformations is strongly terminating for fair sequences of transformations. Such fair sequences therefore always lead to the residual program.

Proof. Of all our transformations, only the GPPE adds new clauses to the program, all others *decrease* the program clauses.

Our fairness condition implies that for every atom X, every positive occurrence of X in a body of a clause is eventually replaced. This means that for every X, there is a natural number n such that all programs P_i with $i \geq n$ in our fair sequence do not contain X in their bodies. Since we are dealing with finite propositional programs, from some n' on we have only negative bodies in our rules. Then the GPPE can not be applied and the other transformations strictly decrease the number of clauses. □ A simple method to get

a fair sequence of transformations is to apply GPPE only to non-tautological clauses and to ensure that whenever GPPE has been applied to get rid of an occurrence of X, then all such occurrences of X (in other clauses) have also been removed, before applying GPPE to another occurrence of $Y \neq X$.

5. Calculi for Other Semantics

An important question is: *Are all our transformations necessary, or are some superfluous?* In fact, simple counterexamples show that if any one of our transformations is skipped, then the remaining transformations do not suffice to transform any program Φ into the normalform $res(\Phi)$. But it could still be possible that a strict subset of our transformations constitutes a confluent calculus. In fact, for positive disjunctive programs we have

Corollary 5.1 (GCWA).
The calculus consisting of GPPE, Elimination of Tautologies and Elimination of Non-minimal Rules is confluent for positive disjunctive programs. In addition, for any positive disjunctive program Φ there is an irreducible Φ' with $\Phi \rightarrow^* \Phi'$ such that all rules of Φ' have empty bodies. Such a Φ' is exactly the residual program $res(\Phi)$ of Φ.

It turns out that the above three transformations also form a confluent calculus for *arbitrary disjunctive programs with negation*. But the normalform is not identical with the residual program, as can be easily seen (for example we can not skip negative literals $\neg E$ that do not occur in any head).

Corollary 5.2 (GPPE, Taut, Non-Min).
The calculus consisting of GPPE, Elimination of Tautologies, and Elimination of Non-minimal Rules is confluent. No proper subset of this calculus is confluent. Although there is for any program Φ an irreducible Φ' with $\Phi \rightarrow^* \Phi'$, such a Φ' is in general different from the residual program $res(\Phi)$.

Proof. The example given at the beginning of the proof of Theorem 4.3 shows that every calculus containing at least GPPE and Elimination of Tautologies has also to contain Elimination of Non-minimal Rules in order to be confluent. The proof goes exactly as the proof of Theorem 4.3 where the lemmas 4.5 and 4.6 were used. In fact, the underlying notion of the *model set* $\mathcal{M}$ of a program Φ is even simpler: it consists just of all minimal models (M^-, M^+). The additional conditions in the proof of Theorem 4.3 were just needed in order to capture positive and negative Reduction.

The program $A \leftarrow \neg B$ is an example of a program which is in normalform with respect to the smaller calculus, but

$$res(\{A \leftarrow \neg B\}) = \{A \leftarrow \}.$$

□

Corollary 5.3 (WFS for normal programs).
The calculus consisting of (the natural restrictions to non-disjunctive programs of) GPPE, Elimination of Tautologies, Elimination of Non-minimal Rules, Positive and Negative Reduction is confluent.

The residual program gives immediately the wellfounded semantics according to the computation in Theorem 4.2.

5.1. STABLE

The stable semantics is based on a set of models, and therefore it is more appropriate for such semantics not to restrict to the sceptical version, but to take into account all intended models. In fact, in [10] our characterizations are based on the notion of a semantics which is slightly stronger than the one given in Definition 3.1. We defined there a semantics to be any mapping associating to any program a *set of three-valued models*. Similarly to Theorem 3.7, we proved that the disjunctive stable semantics is the weakest semantics satisfying GPPE, *Elimination of Tautologies* and the following, additional transformation:

Definition 5.4 (Elimination of Contradictions).
Semantics $\mathcal{S}_{\mid\sim}$ satisfies the *Elimination of Contradictions* iff it is invariant under the following transformation:

$$\text{Delete a rule } \mathcal{A} \leftarrow \mathcal{B} \wedge \neg\mathcal{C} \text{ with } \mathcal{B} \cap \mathcal{C} \neq \emptyset.$$

This simply means that rules having both an atom A and its negation $\neg A$ in their body can be deleted.

Theorem 5.5 (STABLE as Weakest Semantics, (Cor. 4.2 in [10])).
There exists the weakest semantics which satisfies the properties of GPPE, Elimination of Tautologies and Contradictions. This is exactly the stable semantics STABLE.

The reason why we do not need our positive and negative reduction is because the underlying notion of a semantics (three-valued models of the program) is strong enough to imply these properties when GPPE, and Elimination of Contradictions are present.

Since Red^{pos} and Red^{neg} hold for STABLE, we can simply extend our calculus by the *Elimination of Contradictions* rule — we call this set of transformations the *extended* calculus. This extended calculus, however, is not confluent.

Example 5.6 (Extended Calculus is not Confluent).

$$\begin{array}{llll}
\Phi_{st}: & A \leftarrow & B, & \neg B \\
 & B \leftarrow & & \neg A \\
 & & & \\
Contra(\Phi_{st}): & B \leftarrow & & \neg A \\
 & & & \\
GPPE_B(\Phi_{st}): & A \leftarrow & & \neg A, \neg B \\
 & B \leftarrow & & \neg A
\end{array}$$

While $GPPE_B(\Phi_{st})$ is already irreducible, $Contra(\Phi_{st})$ can be made so by applying *Positive Reduction*: we finally get the program "$B \leftarrow$ " which correctly constitutes the unique stable model $\{\neg A, B\}$ of Φ_{st}.

It seems that applying GPPE "in the wrong way" (namely to obtain Φ_{st} from $GPPE_B(\Phi_{st})$) in order to apply *Elimination of Contradictions* is more suitable for STABLE. It might also help to define a unique normal form other than the residual program. In the last example, the residual program of Φ_{st} is $GPPE_B(\Phi_{st})$ and thus it does not give us immediately the stable semantics (like the computation of D-WFS based on Theorem 4.2).

6. An Equivalent Characterization of D-WFS

We first present the general idea which leads, in its simplest form, to a semantics SEM*. This semantics, however, is too strong as shown by Example 6.3 below (nevertheless, it satisfies all our transformations). This example leads to the modified version SEM$_0$ that turns out to be identical to D-WFS.

Definition 6.1 (Φ/Dis).
Let Φ be a disjunctive logic program over $\mathcal{L}$ and let Dis be a set of pure disjunctions over $\mathcal{L}$. Let Φ/Dis be the program obtained from Φ by doing the following reductions for all $\neg C$ and $C_1 \vee \ldots \vee C_k$

- if $\neg C \in Dis$, then remove all occurrences of $\neg C$ in bodies,
- if $C_1 \vee \ldots \vee C_k \in Dis$ then remove all rules that contain $\{\neg C_1, \ldots, \neg C_k\}$ in their bodies.

Φ/Dis is obviously a slight generalization of the Gelfond-Lifschitz transformation. While the latter is defined relative to a set $N \subseteq At_{\mathcal{L}}$ in such a way that Φ/N is always *positive*, our Φ/Dis still is a disjunctive program containing possibly negative literals. In fact, the GL-transform can be obtained from our transform by setting $\Phi/N = \Phi/Dis_N$ where

$$Dis_N := N \cup \{\neg X : X \in At_{\mathcal{L}} - N\}.$$

The intuition behind Φ/Dis is that we start with a program Φ and first compute a set Dis of disjunctions that are *obviously derivable* from Φ. We then reduce Φ by simplifying it according to Dis and arrive at a possibly much simpler program Φ/Dis.

The underlying idea of the following definition of a semantics SEM* is to use $\Phi/At_{\mathcal{L}}$ (i.e. we just delete all rules containing negative literals) for deriving positive disjunctions and to use $\Phi/\neg At_{\mathcal{L}}$ (i.e. we set all negative literals to true) for deriving negative literals:

*Definition 6.2 (SEM**).*
Let Φ be a disjunctive logic program over $\mathcal{L}$. We define a set $Dis^*(\Phi)$ of pure disjunctions as follows

$$\begin{array}{rllll} Dis^*(\Phi) := & \{\ A_1 \vee \ldots \vee A_k & : \Phi/At_{\mathcal{L}} & \models & A_1 \vee \ldots \vee A_k\ \} \\ \cup & \{\ \neg A & : \Phi/\neg At_{\mathcal{L}} & \models_{min} & \neg A\ \} \end{array}$$

The semantics SEM* of a program Φ is defined as the limit of the following growing sets D_i^* of disjunctions (this series eventually gets constant after some finite number of steps). We start with $D_0^* := Dis^*(\Phi)$ and set[8]

$$D_{n+1}^* := D_n^* \cup Dis^*(\Phi/D_n^*).$$

The overall idea behind SEM* is the following. In one step we reduce the current program Φ into two programs without any negative literals:

– $\Phi/At_{\mathcal{L}}$ is used to derive positive,

– $\Phi/\neg At_{\mathcal{L}}$ is used to derive negative

information. Note that the positive atoms derivable from $\Phi/At_{\mathcal{L}}$ are certainly also true when the rules also contain negative literals. Dually, the negative literals derivable from $\Phi/\neg At_{\mathcal{L}}$ are certainly also true when there are additional rules containing negative literals. Therefore it seems that only those information is derived that should be derivable. Although SEM* uses $\models_{min}$ (which is more complex than just $\models$) it only uses it for the derivation of negative literals from positive disjunctive programs.

Note that although this is a very handy and easy definition, it does not give us any insights of SEM*. In addition, SEM* is already very strong and not even closed under logical consequence.

*Example 6.3 (Behaviour of SEM**).*

$$\begin{array}{llll} \Phi^* : & A \vee B & \leftarrow & \neg C \\ & B & \leftarrow & \neg B \\ & C & \leftarrow & \neg C \end{array} \qquad \begin{array}{llll} \Phi^{**} : & A \vee B & \leftarrow & \\ & B & \leftarrow & \neg B \end{array}$$

We have $\text{SEM}^*(\Phi^*) = \{\neg A\}$ and $\text{SEM}^*(\Phi^{**}) = \{\neg A,\ A \vee B\}$. In the first case, the derivation of $\neg A$ seems to be very strong (B, C are undefined). In the second case, we not only have $\neg A$ but also $A \vee B$ without having B. Of course, we could modify the definition so that it is closed under logical consequence, but then the derivation of B would be very unintuitive.

[8] Simply setting $D_{n+1}^* := Dis^*(\Phi/D_n^*)$ does not produce a monotonically growing series D_n. For the program consisting of $A \vee B \leftarrow \neg A$, $B \leftarrow \neg B, \neg C$ and C, we get $D_0 = \{\neg A, C\}$, $D_1 = \{A \vee B,\ C\}$.

The reason for the shortcoming of SEM* is the derivation of negative literals $\neg A_1$. We therefore modify SEM* by weakening this condition.

Definition 6.4 (SEM$_0$).
Let Φ be a disjunctive logic program over $\mathcal{L}$. We define a set $Dis(\Phi)$ of pure disjunctions as follows

$$Dis(\Phi) := \quad Dis^+(\Phi) \,\cup Dis^-(\Phi) \text{ where}$$

$$\begin{array}{lllll} Dis^+(\Phi) := \{ & A_1 \vee \ldots \vee A_k & : \Phi/At_{\mathcal{L}} & \models & A_1 \vee \ldots \vee A_k \quad \} \\ Dis^-(\Phi) := \{ & \neg A & : \Phi/Dis_N & \models_{min} & \neg A \text{ for all } N \subseteq At_{\mathcal{L}} \\ & & & & \text{s. t.}^9 N \models Dis^+(\Phi) \ \} \end{array}$$

Here, the series D_i defined by $D_{n+1} := Dis(\Phi/D_n)$ grows monotonically and eventually gets constant. We define SEM$_0(\Phi)$ to be the limit of this series.

So instead of deleting *all* negative literals at a blow ($\Phi/\neg At_{\mathcal{L}}$) and looking at all minimal models of $\Phi/\neg At_{\mathcal{L}}$, we compute Φ/N for any $N \subseteq At_{\mathcal{L}}$ that is compatible ($N \models \Phi/At_{\mathcal{L}}$) and we only derive those $\neg A$ that are true in all minimal models of such Φ/N. In the Example 6.3, this prevents us from deriving $\neg A$. Therefore SEM$_0(\Phi^*) = \emptyset$ and SEM$_0(\Phi^{**}) = \{A \vee B\}$.

The following lemma is the key to establish the soundness of D-WFS with respect to SEM* and its completeness with respect to SEM$_0$. We recall from the last section that $\Phi \rightarrow^* \Phi'$ means that Φ can be transformed into Φ' by applying any (finite) sequence of the transformations introduced in Section 3.

Lemma 6.5. Let $\Phi \rightarrow^* \Phi'$. Then the following holds:

- $Dis(\Phi) = Dis(\Phi')$.
- $\Phi/Dis(\Phi) \rightarrow^* \Phi'/Dis(\Phi')$.

The same properties hold for Dis^* instead of Dis.

Proof. It suffices to show these results only for our basic transformations $\Phi \mapsto \Phi'$ (because then it also holds for any sequence of them).

In fact, for if *Trans* is any of *GPPE*, *Elimination of Tautologies*, *Elimination of Non-Minimal Rules* or *Positive Reduction* we have

$$Trans(\Phi)/Dis_N \ = \ Trans(\Phi/Dis_N) \quad (*)$$

for all $N \subseteq At_{\mathcal{L}}$. Therefore $Dis(Trans(\Phi)) = Dis(\Phi)$. Moreover it is straightforward to show

$$Trans(\Phi/Dis(\Phi)) = Trans(\Phi)/Dis(Trans(\Phi))$$

[9] We need to consider only those $N \subseteq At_{\mathcal{L}}$ that are compatible with $Dis^+(\Phi)$. For "$A \vee B,\ C \leftarrow \neg A, \neg B$" we do not want to consider $N_0 := \emptyset$ because then $\neg C$ would not be derivable.

from which the result follows. Exactly the same reasoning holds for Dis^* instead of Dis.

The condition $(*)$ does not hold for *Negative Reduction* Red^-. We refer to our example given in the footnote of Definition 6.4: "$A \vee B,\ C \leftarrow \neg A, \neg B$". For $N_0 := \emptyset$ the left-hand side of $(*)$ consists of "$A \vee B$" while the right-hand side is "$A \vee B,\ C$". But we have

$$Red^-(\Phi)/At_{\mathcal{L}} = Red^-(\Phi/At_{\mathcal{L}})$$

and also

$$Dis(Red^-(\Phi)) = Dis(\Phi).$$

For this last result we use the fact that only those N with $N \models Dis^+(\Phi)$ are considered. It also holds $Red^-(\Phi/Dis(\Phi)) = Red^-(\Phi)/Dis(Red^-(\Phi))$ from which the desired result follows. The proof for Dis^* instead of Dis is literally the same. □

Corollary 6.6 (SEM$_0$ and SEM* are sound w.r.t. $\Phi \mapsto res(\Phi)$).
$\Phi \rightarrow^* \Phi'$ implies SEM$^*(\Phi)$=SEM$^*(\Phi')$ and SEM$_0(\Phi)$=SEM$_0(\Phi')$.

Proof. We show by induction on n, that $D_n = D'_n$ (resp. $D^*_n = {D'_n}^*$) and $\Phi/D_n \rightarrow^* \Phi'/D'_n$ (resp. $\Phi/D^*_n \rightarrow^* \Phi'/{D'_n}^*$). This is immediate by the preceding lemma. □

Theorem 6.7 (D-WFS = SEM$_0$).
D-WFS is complete with respect to SEM$_0$: D-WFS(Φ) = SEM$_0(\Phi)$ for all Φ.

Proof. Let $res(\Phi)$ be the residual program from Φ. The last corollary tells us that SEM$_0(\Phi)$ = SEM$_0(res(\Phi))$. It suffices to show that

$$\begin{array}{rcl} A_1 \vee \ldots \vee A_k \in Dis(res(\Phi)) & \iff & \exists\, \mathcal{A} \leftarrow true \ \in res(\Phi), \\ & & \text{with } \mathcal{A} \subseteq A_1 \vee \ldots \vee A_k, \\ \neg A \in Dis(res(\Phi)) & \iff & A \notin \mathcal{P}(res(\Phi)), \end{array}$$

because then, $res(\Phi)/D_0 = res(\Phi)/Dis(res(\Phi)) = res(\Phi)$ and therefore the sequence $D_0, D_1, \ldots$ immediately ends with D_0. The result follows by Theorem 4.2 b).

We show the first equivalence. Let $A_1 \vee \ldots \vee A_k$ follow from $res(\Phi)/At_{\mathcal{L}}$. Since $res(\Phi)/At_{\mathcal{L}}$ consists of only positive disjunctions, there is a subset of $\{A_1, \ldots, A_k\}$ occurring as a disjunction in $res(\Phi)/At_{\mathcal{L}}$. Because of the definition of $res(\Phi)/At_{\mathcal{L}}$, we are done. The opposite direction is trivial.

Let now $\neg A \in Dis(res(\Phi))$ and suppose A appears in some rule

$$A \vee B_1 \vee \ldots \vee B_m \leftarrow \neg C_1, \ldots, \neg C_n.$$

We have to derive a contradiction. Let $N := At_{\mathcal{L}} - \{C_1, \ldots, C_n\}$. Obviously, $N \models Dis^+(res(\Phi))$, because otherwise there would be a rule $\mathcal{A} \leftarrow$ where $\mathcal{A}$ is a subset of $\{C_1, \ldots, C_n\}$ and $res(\Phi)$ would not be the residual program (because negative Reduction could be applied).

Then $res(\Phi)/Dis_N$ consists of exactly those rule heads of rules $res(\Phi)$ whose bodies are subsets of $\{\neg C_1, \ldots, \neg C_n\}$. By *Elimination of Non-minimal Rules* such a head cannot be a strict subset of $\{A, B_1, \ldots, B_m\}$. Therefore the interpretation $\mathcal{J}$ which makes exactly the B_i false and all other atoms true is a model of $res(\Phi)/Dis_N$. So there is also a minimal model of $res(\Phi)/Dis_N$ below $\mathcal{J}$ which is still smaller. But in this minimal model A is true (because of the disjunction $A \vee B_1 \vee \ldots \vee B_m$). Therefore $res(\Phi)/Dis_N$ does not minimally imply $\neg A$ and we have a contradiction. The opposite direction again is trivial. □

7. Conclusions

In this paper we obtained two main results. First the proof of the confluence of a very simple calculus of program transformations. Second an equivalent formulation in terms of minimal model reasoning. Although the very definition of the semantics D-WFS and the program-transformations have been introduced in [11] the results of this paper are completely new. The confluence of our calculus is not just a simple corollary to the results of [11].

To our knowledge, this is the first time that a confluent calculus has been defined as a proof-theoretical attempt for defining a semantics for logic programs. The confluence of such a calculus is certainly the strongest property one can have. It is worth mentioning that an implementation using the confluence of the calculus can be much more efficient than simply taking a certain ordering of the transformations which is fixed in advance. We are currently experimenting and comparing a fixed bottom-up implementation with one based on the confluence of our calculus.

The equivalence of our semantics with a construction based on minimal model reasoning is also very interesting, because it opens the way to apply methods from this area for efficient implementations. In particular, I. Niemelä suggested very efficient methods for computing minimal models of positive disjunctive programs [34, 33, 35]. Because of our characterization, such methods can be immediately used to implement D-WFS (note that we only need to consider positive disjunctive programs in Definition 6.4). Of course, there are still many open questions left for further research. An important property of our approach is its applicability also to semantics stronger than

[10] WFS$^+$ is shown in [21] to be equivalent to the semantics introduced by SCHLIPF in [43].
[11] WGCWA is equivalent to the semantics introduced by ROSS/TOPOR in [41].

Table I. An Overview of Semantics and Their Properties.

Properties of Logic-Programming Semantics					
Semantics	Domain	Taut.	GPPE	Contra	Non-Min.
Clark's comp [17]	Norm.	no	•	no	•
WFS [44]	Norm.	•	•	no	•
WFS^{+} [20][10]	Norm.	•	no	no	•
GCWA [30]	Pos.	•	•	triv.	•
WGCWA [39][11]	Pos.	no	•	triv.	no
Positivism [6]	Dis.	no	•	no	•
STABLE [28, 36]	Dis.	•	•	•	•
Strong WFS [40]	Dis.	no	no	no	no
STATIONARY [37]	Dis.	•	•	no	•
STATIC [38]	Dis.	•	•	no	•
D-WFS [8]	Dis.	•	•	no	•
REG-SEM [45]	Dis.	•	•	no	•

D-WFS. If only our transformation $\Phi \mapsto res(\Phi)$ is sound for such a semantics (e.g. STATIONARY, STATIC or STABLE) we can use our calculus and already decide many queries without fully computing the semantics. Therefore, putting something on top of $res(\Phi)$ should make it possible to obtain these stronger semantics — we already did this to obtain and implement STABLE in [9]. Similar ideas to obtain STATIC are currently under investigation.

Let us conclude with Table I which shows an overview of the properties of some well-known semantics (see also [10] for interesting characterizations of STABLE): We did not mention positive or negative Reduction because these transformations are satisfied by all semantics.

Finally we would like to mention the DisLoP-project[12], undertaken at the university of Karlsruhe. Our approach has been successfully implemented. In particular, our characterization in Theorem 6.7 has been used to get an implementation of D-WFS in polynomial space ([12]). This is done by using ideas of Niemelä and the notion of Hypertableaux developed in [5].

Acknowledgements

We are grateful to some anonymous referees for their useful comments. We are also indebted to Ilkka Niemelä for pointing out two weaknesses in former versions of Definitions 6.4 and 6.2.

[12] Visit `<http://www.uni-koblenz.de/ag-ki/DLP/>` in the world wide web to get more information on this and related projects.

References

1. Krzysztof R. Apt and Roland N. Bol. Logic Programming and Negation: A Survey. *Journal of Logic Programming*, 19-20:9–71, 1994.
2. Chitta Baral and Michael Gelfond. Logic Programming and Knowlege Representation. *Journal of Logic Programming*, 19-20, 1994.
3. Chitta Baral, Jorge Lobo, and Jack Minker. Generalized Disjunctive Well-founded Semantics for Logic Programs: Declarative Semantics. In Z.W. Ras, M. Zemankova, and M.L Emrich, editors, *Proceedings of the 5th Int. Symp. on Methodologies for Intelligent Systems, Knoxville, TN, October 1990*, pages 465–473. North-Holland, 1990.
4. Chitta Baral, Jorge Lobo, and Jack Minker. WF^3: A Semantics for Negation in Normal Disjunctive Logic Programs. In Z.W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems*, LNAI 542, pages 459–468, Berlin, 1991. Springer.
5. Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. In L.M. Pereira J.J. Alferes and E. Orlowska, editors, *Logics in Artificial Intelligence (JELIA '96)*, LNCS 1126, pages 1–17. Springer, 1996.
6. Nicole Bidoit and Richard Hull. Positivism vs. minimalism in deductive databases. In *Proc. of the 5th ACM Symp. on Principles of Database Systems (PODS'86)*, pages 123–132, 1986.
7. Stefan Brass and Jürgen Dix. Computing Disjunctive Stable Semantics based on Clark's Completed Database. In *Proc. of the 6th GI-Workshop "Grundlagen von Datenbanken", Bad Helmstedt, September 1994*, pages 30–34, 1994.
8. Stefan Brass and Jürgen Dix. A disjunctive semantics based on unfolding and bottom-up evaluation. In Bernd Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen,* (IFIP '94-Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases), pages 83–91, Berlin, 1994. Springer.
9. 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.
10. 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.).
11. Stefan Brass and Jürgen Dix. Semantics of Disjunctive Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, accepted for publication, 1997. (Extended abstract appeared in: Disjunctive Semantics Based upon Partial and Bottom-Up Evaluation, *Proceedings of the 12-th International Logic Programming Conference, Tokyo*, pages 199–213, 1995. MIT Press.).
12. Stefan Brass, Jürgen Dix, Ilkka Niemelä, and Teodor. C. Przymusinski. Comparison and Efficient Computation of the Static and the Disjunctive WFS. In Gerd Brewka, Emil Weydert, and Cees Witteveen, editors, *Proceedings of the third Dutch-German Workshop on Nonmonotonic Reasoning and its Applications*, pages 37–42. February 1997.
13. 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.
14. Stefan Brass, Ulrich Zukowski, and Burkhardt Freitag. Transformation Based Bottom-Up Computation of the Well-Founded Model. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 171–201. Springer, Berlin, 1997.
15. Gerd Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning: An Overview*. CSLI Lecture Notes 73. CSLI Publications, Stanford, CA, 1997.

16. Gerhard Brewka and Jürgen Dix. Knowledge representation with logic programs. Technical report, Tutorial Notes of the 12th European Conference on Artificial Intelligence (ECAI '96), 1996. Also appeared as Technical Report 15/96, Dept. of CS of the University of Koblenz-Landau. Will appear as Chapter 6 in *Handbook of Philosophical Logic*, 2nd edition (1998), Volume 6, Methodologies.
17. Keith L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data-Bases*, pages 293–322. Plenum, New York, 1978.
18. J. Dix, L. Pereira, and T. Przymusinski, editors. *Non-Monotonic Extensions of Logic Programming*, LNAI 927, Berlin, 1995. Springer.
19. J. Dix, L. Pereira, and T. Przymusinski, editors. *Non-Monotonic Extensions of Logic Programming*, LNAI 1216, Berlin, 1997. Springer.
20. 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.
21. Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.
22. Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.
23. 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.
24. Jürgen Dix and Frieder Stolzenburg. Computation of Non-Ground Disjunctive Well-Founded Semantics with Constraint Logic Programming (preliminary report). In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 202–226. Springer, Berlin, 1997.
25. Jürgen Dix and Frieder Stolzenburg. A Framework to incorporate Nonmonotonic Reasoning into Constraint Logic Programming. *Journal of Logic Programming*, 1998. Special Issue on *Constraint Logic Programming*, Guest Editors: Kim Marriott and Peter Stuckey, to appear.
26. Thomas Eiter and Georg Gottlob. Propositional Circumscription and Extended Closed World Reasoning are Π_2^P-complete. *Theoretical Computer Science*, 144(2):231–245, Addendum: vol. 118, p. 315, 1993, 1993.
27. Thomas Eiter, Georg Gottlob, and Heikki Mannila. Expressive Power and Complexity of Disjunctive DATALOG. In *Proceedings of Workshop on Logic Programming with Incomplete Information, Vancouver Oct. 1993, following ILPS' 93*, pages 59–79, 1993.
28. 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.).
29. Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT-Press, 1992.
30. 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.
31. Jack Minker. An Overview of Nonmonotonic Reasoning and Logic Programming. *Journal of Logic Programming, Special Issue*, 17, 1993.
32. Jack Minker. Logic and databases: A 20 year retrospective. In Dino Pedreschi and Carlo Zaniolo, editors, *Proceedings of the International Workshop on Logic in Databases (LID)*, LNCS 1154, pages 3–58. Springer, Berlin, 1996.
33. Ilkka Niemelä. Implementing circumscription using a tableau method. In W. Wahlster, editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 80–84, Budapest, Hungary, August 1996. John Wiley.

34. Ilkka Niemelä. A tableau calculus for minimal model reasoning. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 278–294, Terrasini, Italy, May 1996. LNAI 1071, Springer-Verlag.
35. 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.
36. 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.).
37. Teodor Przymusinski. Stationary Semantics for Normal and Disjunctive Logic Programs. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *DOOD '91, Proceedings of the 2nd International Conference*, Berlin, December 1991. Muenchen, Springer. LNCS 566.
38. Teodor Przymusinski. Static Semantics For Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357, 1995.
39. Arcot Rajasekar, Jorge Lobo, and Jack Minker. Weak Generalized Closed World Assumption. *Journal of Automated Reasoning*, 5:293–307, 1989.
40. 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.
41. Kenneth A. Ross and Rodney A. Topor. Inferring negative Information from disjunctive Databases. *Journal of Automated Reasoning*, 4:397–424, 1988.
42. 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.
43. John S. Schlipf. Formalizing a Logic for Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 5:279–302, 1992.
44. 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.
45. Jia-Huai You and Li-Yan Yuan. Autoepistemic Circumscription and Logic Programming. *Journal of Automated Reasoning*, 10:143–160, 1993.

Address for correspondence: Universität Koblenz, Institut für Informatik, Rheinau 1, 56075 Koblenz, Germany, E-mail: `dix@informatik.uni-koblenz.de`