A General Approach to Bottom-Up Computation of Disjunctive Semantics

Stefan Brass[1] and Jürgen Dix[2]

[1] Univ. of Hannover, Inst. f. Informatik, Lange Laube 22, D-30159 Hannover, FRG, sb@informatik.uni-hannover.de

[2] Univ. of Koblenz, Inst. f. Informatik, Rheinau 1, D-56075 Koblenz, FRG dix@informatik.uni-koblenz.de

Abstract. Our goal is to derive bottom-up query-evaluation algorithms from abstract properties of the underlying negation semantics. In this paper, we demonstrate our approach for the disjunctive stable model semantics, but the ideas are applicable to many other semantics as well. Our framework also helps to understand and compare other proposed query evaluation algorithms. It is based on the notion of conditional facts, developed by Bry and Dung/Kanchansut. We start by computing a "residual program" and show that it 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 can already be answered on the basis of the residual program. For the remaining literals, we propose to use an appropriate completion of the residual program, which syntactically characterizes the intended models. In the case of the stable model semantics, we utilize an interesting connection to Clark's completion.

1 Introduction

In this paper, we introduce a framework for bottom-up query evaluation, and apply it to the disjunctive stable model semantics [GL91b, Prz91a]. A unique feature of our approach is that we derive the algorithms directly from abstract properties of the semantics [KLM90, Bra90, Dix92, DM94a, Dix95a, Dix95b, Dix95c]. In our case, these properties require certain elementary program transformations to be possible. Related transformational approaches have been considered in [DM93, CL95, DN95]. 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. The authors of [DN95, KSS91] consider only non-disjunctive programs under the well-founded semantics.

The first step of our approach is to compute the set of implied "conditional facts" [Bry89, Bry90, DK89a, DK89b, HY91]. These are rules without positive body literals, which result from delaying the evaluation of negative body literals (the delayed literals are attached as "conditions" to the derived facts). We show that this transformation preserves the semantics of the original program under very weak conditions, namely the possibility of unfolding (also called the "generalized principle of partial evaluation" GPPE [DM94b, Dix95b, SS94]) and the irrelevance of trivial tautologies (having the same atom in head and body). Of course, such a delaying of negative body literals is implicit in other query evaluation algorithms for non-stratified programs. It is nice, however, that our whole algorithm can be understood as a source level transformation. Lower level data structures are useful for speeding up certain abstract operations, but they are not essential for understanding the correctness of the algorithm.

The second step of the approach is to carry out certain reductions, namely to evaluate negative body literals if their truth value can be trivially determined. The result of this is called the *residual program*. The well-founded model (resp. a disjunctive extension of it) can be directly derived from the residual program. The computation of the residual program is also valid under many other semantics, e.g. the static [Prz95] and stationary [Prz91b] semantics. Of course, it is not necessary to have a strict separation between the first and the second step of the approach. By mixing and optimizing them, we are also able to derive the standard bottom-up query evalution algorithm for stratified non-disjunctive programs and the algorithm proposed in [KSS91] for the WFS of non-disjunctive programs. We believe that our approach can be a good framework to understand and compare other proposed (bottom-up) query evaluation algorithms.

Only the few atoms which are undefined in the WFS are considered in the third step. We propose to compute an appropriate completion of the residual program, which syntactically characterizes the intended models of the given semantics. In our case, we utilize (a disjunctive extension of) CLARK's completion [Cla78], which is equivalent to the stable model semantics for residual programs. Given such a set of formulas, standard theorem proving techniques can be used (we use hyperresolution [CL73] for model construction in an interesting way). More specialized techniques are also possible, but in the case of stable models it seems that not much can be gained. We believe that the availability of such a syntactical characterization greatly simplifies the understanding of query evaluation (compare, e.g. [BL92] and [BL93]). It certainly helps to view more specialized techniques only as possible optimizations of a simple basic approach.

Let us finally mention other proposed query evaluation algorithms for the (non-disjunctive) stable model semantics. Closest to our approach is probably [CW93]: They also compute a "residual program", but this is done top-down, and their processing of the residual program is quite different. The algorithms proposed in [BNNS93] (see also [DM93]) apply the definition or certain characterizations of the stable models quite directly, but use linear programming techniques for an efficient construction of minimal models. This is an interesting idea and could be used to replace our third step (expressing it the other way round, our computation of the residual program can be useful as a preprocessing phase for [BNNS93, DM93]). In [PAA91], a top-down algorithm for credulous query evaluation in extended stable models is presented. There, trees of ground literals are constructed which explain the truth or falsity of atoms in well-founded and stable models. As any top-down algorithm, they have to do some not easily understandable loop checking, which we get for free. On the other hand, top-down algorithms are goal-directed, which would need a "magic set" transformation in our case (this has been recently done by DEGERSTEDT/NILSSON in [DN95] for the well-founded semantics for non-disjunctive programs).

The paper is organized as follows. Section 2 contains the basic definitions, especially of the semantical properties on which the algorithm is based. In Section 3, we define the residual program, and show its equivalence to the original program. We then explain how to efficiently compute the residual program in Section 4. The usage of CLARK's completion to compute stable models is the subject of Section 5. In Section 6, we sketch how the algorithm [KSS91] for the well-founded semantics fits into our framework. We conclude with a summary and an outlook on future work. The proofs are delayed to an appendix.

2 Semantics and Program Transformations

We consider allowed disjunctive DATALOG programs over some fixed function-free finite signature Σ:

Definition 1 (Logic Program).
A program Φ is a finite set of rules of the form

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

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.

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 $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\ \mathcal{C}$.

Since not all proposed semantics are defined for the complete class of disjunctive programs, it is also possible to consider only a subset (which, however, must satisfy certain closure properties). Especially, our results remain valid when restricted to non-disjunctive or positive programs.

The following is a very general view of semantics:

Definition 2 (Semantics).
A semantics SEM is a mapping from the class of logic programs Φ over Σ into some semantic domain, e.g. sets of models or sets of consequences. The only restriction is that SEM is invariant under instantiation, i.e. $\mathrm{SEM}(\Phi) = \mathrm{SEM}(\Phi^*)$, where Φ^* denotes the full instantiation of Φ.

In this paper, we are mainly concerned with the stable model semantics of disjunctive programs [GL91b, Prz91a]:

Definition 3 (Stable Model).
A 2-valued Herbrand interpretation I is a stable model of a logic program Φ iff I is a minimal model of Φ/I, where Φ/I denotes the GELFOND/LIFSCHITZ-Transformation of Φ wrt I (instantiation of Φ and evaluation of negative body literals in I).

But our approach is not restricted to this particular semantics, it can be applied to any semantics satisfying some weak properties. Our properties all have the form that certain elementary transformations on logic programs are possible without changing the meaning of the programs. It suffices to consider instantiated (i.e. variable free) programs here, because we required that the semantics is instantiation-invariant, so it is completely determined by its values on instantiated programs.

Definition 4 (Program Transformation).
A transformation is any relation $\mapsto$ between instantiated logic programs.

Definition 5 (Equivalence Transformation).
A program transformation $\mapsto$ is a SEM-equivalence transformation if for all Φ, Φ' with $\Phi \mapsto \Phi'$: $\mathrm{SEM}(\Phi) = \mathrm{SEM}(\Phi')$.

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

Definition 6 (Elimination of Tautologies).
A semantics SEM allows elimination of tautologies, iff the transformation

$$\text{Delete a rule } \mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\ \mathcal{C} \text{ with } \mathcal{A} \cap \mathcal{B} \neq \emptyset.$$

is a SEM-equivalence transformation.

Another, especially important transformation is partial evaluation in the sense of the "unfolding" operation. It is the "Generalized Principle of Partial Evaluation (GPPE)" of [DM94b, Dix95c, SS94]. In the case of non-disjunctive programs, the idea is simply to replace a positive body literal B by the bodies of all matching rules. This is very natural because the rules with B in their head are usually considered as the only possibilities to derive B. In the disjunctive case, we simply add the disjunctive context to the head of the resulting rule. For instance, if we apply this transformation to q in the body of $p \leftarrow q \wedge r$, and there is the rule $q \vee s \leftarrow t$, then the resulting rule is $p \vee s \leftarrow t \wedge r$.

Definition 7 (Generalized Principle of Partial Evaluation).
A semantics SEM satisfies GPPE iff the following transformation on instantiated logic programs is a SEM-equivalence transformation:

Replace a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\ \mathcal{C}$, where $\mathcal{B}$ contains an atom B, by the n rules

$$\mathcal{A} \cup \bigl(\mathcal{A}_i - \{B\}\bigr) \ \leftarrow\ \bigl(\mathcal{B} - \{B\}\bigr) \cup \mathcal{B}_i \ \wedge\ \mathit{not}\ \bigl(\mathcal{C} \cup \mathcal{C}_i\bigr)$$

where $\mathcal{A}_i \leftarrow \mathcal{B}_i \wedge \mathit{not}\ \mathcal{C}_i$ $(i = 1, \ldots, n)$ are all rules with $B \in \mathcal{A}_i$.

The following transformations are used in our algorithm for optimization only, so they are not as essential as the above ones. First, we can certainly evaluate negative body literals if their truth value is trivial:

Definition 8 (Reductions).
A semantics SEM allows positive/negative reduction iff the following are equivalence transformations on instantiated programs:

1. Replace a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}$ by $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \big(\mathcal{C} - \{C\}\big)$ where C appears in no rule head.
2. Delete a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}$ if there is a disjunctive fact $\mathcal{A}' \leftarrow \mathit{true}$ with $\mathcal{A}' \subseteq \mathcal{C}$.

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

Definition 9 (Elimination of Non-Minimal Rules).
A semantics SEM allows elimination of non-minimal rules iff the following is a SEM-equivalence transformation:

Delete a rule $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\, \mathcal{C}$ s.t. there is another rule $\mathcal{A}' \leftarrow \mathcal{B}' \wedge \mathit{not}\, \mathcal{C}'$
with $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{B}' \subseteq \mathcal{B}$, $\mathcal{C}' \subseteq \mathcal{C}$
(where at least one $\subseteq$ is proper).

Let us conclude this section with an overview of the properties of some well-known semantics (see also [BD95a] for interesting characterizations of STABLE):

Properties of Logic-Programming Semantics					
Semantics	Domain	El. Taut.	GPPE	P./N. Red.	Nonmin.
Clark's comp [Cla78]	Nondis.	—	•	•	•
GCWA [Min82]	Pos.	•	•	(trivial)	•
Positivism [BH86]	Dis.	—	•	•	•
WGCWA [RLM89][1]	Pos.	—	•	(trivial)	—
Stable [GL91b, Prz91a]	Dis.	•	•	•	•
WFS [VGRS91]	Nondis.	•	•	•	•
Strong WFS [Ros90]	Dis.	—	—	•	—
Stationary [Prz91b]	Dis.	•	•	•	•
Static [Prz95]	Dis.	•	•	•	•

3 Computing the Residual Program

In this section we explain how to compute a normal form of the given program, called the residual program, and prove that it is equivalent to the original program under very weak conditions on the semantics. Our approach is based

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

on the notion of "conditional facts", as developed by BRY [Bry89, Bry90] and DUNG/KANCHANSUT [DK89a, DK89b] (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. Because of the allowedness restriction on the programs, conditional facts are ground:

Definition 10 (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 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:

$$\begin{array}{ccccccc}
p(a) & \leftarrow & & & \neg s(b) & \wedge & \neg r(b). \\
\uparrow & & & & \uparrow & & \uparrow \\
p(X) & \leftarrow & q_1(X) & \wedge & q_2(X,Y) & \wedge & \neg r(Y). \\
 & & \uparrow & & \uparrow & & \\
 & & q_1(a) & & q_2(a,b) \leftarrow \neg s(b) & &
\end{array}$$

In the disjunctive case, one applies the "hyperresolution" operator, by adding to every fact also a disjunctive context [CL73, MR90, Bra95]. An implementation with database techniques has been suggested in [BL92]. Formally, the immediate consequence operator is generalized to conditional facts as follows:

Definition 11 (Immediate Consequences with Conditions).
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 there are only finitely many predicates and constants to build ground atoms occuring in conditional facts, and there are only finitely many subsets of all these atoms (corresponding to head and body). The transformation from the original logic program to the set of all derived conditional facts does not change the semantics of the program:

Theorem 12 ($\Phi \mapsto \mathrm{lfp}(T_\Phi)$ is a SEM-Equivalence Transformation).
Let SEM be a semantics which satisfies GPPE, and allows the elimination of tautologies. Then $SEM(\Phi) = SEM\big(\mathrm{lfp}(T_\Phi)\big)$.

This is an important and non-trivial strengthening of a similar theorem from [BD95b]. Intuitively, it might seem very natural, because GPPE and T_Φ are rather similar. However, T_Φ works bottom-up, whereas GPPE works top-down. In fact, the formal proof is quite long (see the appendix). The earlier proof needed in addition the possibility to eliminate non-minimal rules, and was much simpler.

The next step of our algorithm is to reduce the derived conditional facts by evaluating the conditions where possible. For the computation of stable models, this step is not strictly necessary, but it is an important optimization. In the case of non-disjunctive programs, we simply evaluate *not* C to true if C appears in no head, and to false, if C is given as an unconditional fact. In the general case, the reduction is defined as follows:

Definition 13 (Reduction of Conditional Facts).
Let $\mathcal{P}(\Gamma)$ be the set of atoms occuring in some rule head in Γ. Then we define:

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

We again iterate this operator until nothing changes (the number of occurring literals decreases in each step). We call the result the "residual program", because it is similar to the residual program computed in [CW93]. However, our version also contains the true facts, not only conditions for the undefined facts.

Definition 14 (Residual Program).
The residual program of Φ is

$$\hat{\Phi} := R^{\omega}\big(\mathrm{lfp}(T_\Phi)\big).$$

Corollary 15 ($\Phi \mapsto \hat{\Phi}$ is a SEM-Equivalence Transformation).
Let SEM be a semantics satisfying all properties defined in Section 2. Then $SEM(\hat{\Phi}) = SEM(\Phi)$ for every logic program Φ and its residual program $\hat{\Phi}$.

So we now have a logic program equivalent to the original one under STABLE, WFS, and many other semantics. What we have gained during this computation is first that the rules have a very particular form (no positive body literals, no variables), and second that the truth value of most atoms can be trivially read from this program (because the atom is explicitly given as an unconditional fact or it does not appear in any rule head). There is usually only a very small number of proper conditional facts remaining, and this is where the various semantics differ. For instance, it follows from our results in [BD95b] that the well-founded semantics simply makes all the remaining "critical atoms" undefined.

4 Optimizations and Implementation Hints

In the last section, we gave a theoretical basis for computing the residual program. Now let us consider implementation issues in greater detail. Let us start with a trivial ("naive") implementation of the fixpoint computation:

$\Gamma_{last} := \emptyset$;
$\Gamma := \{\mathcal{A} \leftarrow not\ \mathcal{C} \mid \mathcal{A} \leftarrow not\ \mathcal{C}\ \in \Phi\}$;
while $\Gamma \neq \Gamma_{last}$ **do**
  $\Gamma_{last} := \Gamma$;
  foreach $\mathcal{A} \leftarrow \mathcal{B} \wedge not\ \mathcal{C}$ **with** $\mathcal{B} \neq \emptyset$ **do**
    $apply(\mathcal{A} \leftarrow \mathcal{B} \wedge not\ \mathcal{C})$; /* $\Gamma := \Gamma \cup T_{\{\mathcal{A} \leftarrow \mathcal{B} \wedge not\ \mathcal{C}\}}(\Gamma_{last})$ */

We of course aim at semi-naive evaluation (see, e.g., [RSS94]). There, we have to distinguish between three sets of conditional facts:

1. The new conditional facts Γ_{new} which are generated during this iteration. We resolve with them only beginning in the next iteration (otherwise we would loose the control which resolution steps actually have been done).
2. The conditional facts Γ_{diff} generated in the last iteration. At least one body literal must be matched with them.
3. Conditional facts Γ_{old} generated before the last iteration. We guarantee that all resolution steps using only these conditional facts have already been done.

We will not consider this further here, since the standard techniques can be used.

The procedure *apply* evaluates a given rule against the current set of conditional facts. For simplicity, we show an implementation only for the case of two positive body literals. The generalization to a variable number of body literals is straightforward:

procedure $apply(\mathcal{A} \leftarrow B_1 \wedge B_2 \wedge not\ \mathcal{C})$:
  foreach $\mathcal{A}_1 \leftarrow not\ \mathcal{C}_1\ \in \Gamma_{last}$ **and** $A_1 \in \mathcal{A}_1$ **do**
    if there is θ_1 with $B_1\theta_1 = A_1$ **then**
      foreach $\mathcal{A}_2 \leftarrow not\ \mathcal{C}_2\ \in \Gamma_{last}$ **and** $A_2 \in \mathcal{A}_2$ **do**
        if there is θ_2 with $B_2(\theta_1 \circ \theta_2) = A_2$ **then**
          $\theta := \theta_1 \circ \theta_2$;
          $\mathcal{A}' := \mathcal{A}_0\theta \cup \big(\mathcal{A}_1 - \{B_1\theta\}\big) \cup \big(\mathcal{A}_2 - \{B_2\theta\}\big)$;
          $\mathcal{C}' := \mathcal{C}_0\theta \cup \mathcal{C}_1 \cup \mathcal{C}_2$;
          $insert(\mathcal{A}' \leftarrow not\ \mathcal{C}')$;
            /* $\Gamma := \Gamma \cup \{\mathcal{A}' \leftarrow not\ \mathcal{C}'\}$ */

There is a large body of knowledge on possible improvements upon this basic algorithm. Obviously, indexes are needed to speed up the search for $\mathcal{A}_i \leftarrow not\ \mathcal{C}_i$ and $A_i \in \mathcal{A}_i$ matching the body literal $B_i(\theta_1 \circ \cdots \circ \theta_{i-1})$. Note that the management of substitutions for allowed rules is very simple: The first occurrence of a variable assigns a value, later occurrences act only as a test. So if our internal representation of rules distinguishes between first and later occurrences, we do not explicitly need to undo a binding (e.g., by means of a trail).

The order of the body literals is relevant to make the best possible use of the existing indexes (and to fail a rule application as soon as possible). We should also employ "intelligent backtracking" (going back to the body literal which introduced the binding responsible for a later failed literal). Furthermore, there are of course other join techniques besides our "nested loop/index join".

The next point to consider is the procedure *insert*. Its goal is to add a newly computed conditional fact $\mathcal{A} \leftarrow not\,\mathcal{C}$ to the current set Γ of conditional facts. In doing so, it at least has to eliminate duplicates, otherwise even simple tautologies like $p(X) \leftarrow p(X)$ will lead to non-termination. But we can do even more. After computing the set of implied conditional facts, we will apply our reductions. So why not apply the reductions right now? Let us consider the three types of reductions in detail:

1. Elimination of non-minimal conditional facts: Suppose Γ contains a conditional fact $\mathcal{A}' \leftarrow not\ \mathcal{C}'$ with $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$. Then of course $\mathcal{A} \leftarrow not\ \mathcal{C}$ will later be deleted, but also all facts derived using $\mathcal{A} \leftarrow not\ \mathcal{C}$ will be deleted: Let $A \in \mathcal{A}$ be matched with a body literal. If $A \in \mathcal{A}'$, we can as well use the stronger conditional fact. If $A \notin \mathcal{A}'$, the conditional fact $\mathcal{A}' \leftarrow not\,\mathcal{C}'$ is already stronger than the derived conditional fact. So in either case, non-minimal facts as input yield non-minimal facts as output. This allows us to delete them at once when they are generated — we do not have to wait until the reduction phase.
2. Positive reduction: This has to be delayed, because when we evaluate $not\ C$ to true, we have to be sure that C will not later be derived.
3. Negative reduction: This can be done at once, because whenever $\mathcal{A} \leftarrow not\,\mathcal{C}$ is used in a resolution step, the resulting conditional fact contains also the condition $not\ \mathcal{C}$ (plus possibly more). So if we know already that $not\ \mathcal{C}$ is false, we only get superfluous conditional facts, which will later anyway be deleted. Therefore, we can delete them at once and avoid the unnecessary resolution steps.

But not only the newly computed conditional fact can be thrown away due to the reductions — we also may have to delete a member of Γ:

procedure *insert*($\mathcal{A} \leftarrow not\ \mathcal{C}$):
  foreach $\mathcal{A}' \leftarrow not\ \mathcal{C}' \in \Gamma$ **with** $\mathcal{A} \subset \mathcal{A}'$, $\mathcal{C} \subseteq \mathcal{C}'$ (or $\subseteq$, $\subset$) **do**
    $\Gamma := \Gamma - \{\mathcal{A}' \leftarrow not\ \mathcal{C}'\}$;
  if $\mathcal{C} = \emptyset$ **then**
    foreach $\mathcal{A}' \leftarrow not\ \mathcal{C}' \in \Gamma$ **with** $\mathcal{A} \subseteq \mathcal{C}'$ **then**
      $\Gamma := \Gamma - \{\mathcal{A}' \leftarrow not\ \mathcal{C}'\}$;
  if there is no $\mathcal{A}' \leftarrow not\ \mathcal{C}' \in \Gamma$ with $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{C}' \subseteq \mathcal{C}$ **then**
    if there is no $\mathcal{A}' \leftarrow true \in \Gamma$ with $\mathcal{A}' \subseteq \mathcal{C}$ **then**
      $\Gamma := \Gamma \cup \{\mathcal{A} \leftarrow not\ \mathcal{C}\}$;

Deleting derived facts makes the immediate consequence operator nonmonotonic wrt $\subseteq$, however, it is monotonic in the lattice based on

$$\Gamma_1 \sqsubseteq \Gamma_2 \;:\Longleftrightarrow\; \text{for every } \mathcal{A}_1 \leftarrow not\ \mathcal{C}_1 \in \Gamma_1 \text{ there is } \mathcal{A}_2 \leftarrow not\ \mathcal{C}_2 \in \Gamma_2$$
$$\text{such that } (\mathcal{A}_2 \subseteq \mathcal{A}_1 \text{ and } \mathcal{C}_2 \subseteq \mathcal{C}_1) \text{ or } (\mathcal{C}_2 = \emptyset \text{ and } \mathcal{A}_2 \subseteq \mathcal{C}_1).$$

Note also that this deletion is even compatible with semi-naive evaluation: When we delete an old conditional fact, we will insert a new "better" one for it. So there will be at least one further iteration, where we can use the new fact instead of the old one (and any rule application which uses this new fact is allowed in the next semi-naive step).

Of course, the feasibility of these reductions relies on an efficient algorithm for finding subsets/supersets. As experiments of SEIPEL [Sei94] have shown, the elimination of duplicate and subsumed disjunctive facts can take more than half of the total running time of query evaluation. The algorithm we are going to present is quite efficient; its time complexity for finding conditional facts in Γ which are affected by a newly computed fact $\mathcal{A} \leftarrow not\ \mathcal{C}$ is

$$O(\text{number of occurrences of literals from } \mathcal{A} \leftarrow not\ \mathcal{C} \text{ in } \Gamma).$$

Further improvements are subject of future research (for instance, it might be possible to generalize data structures for partial-match queries on relations).

Let us define the length of a conditional fact $\mathcal{A} \leftarrow not\ \mathcal{C}$ as the number of literals in it, i.e. $|\mathcal{A}| + |\mathcal{C}|$. The overlap of two disjunctive facts is the number of common literals, i.e. $|\mathcal{A}_1 \cap \mathcal{A}_2| + |\mathcal{C}_1 \cap \mathcal{C}_2|$. Now it is clear that $\mathcal{A}_1 \leftarrow not\ \mathcal{C}_1$ subsumes $\mathcal{A}_2 \leftarrow not\,\mathcal{C}_2$ (i.e. $\mathcal{A}_1 \subseteq \mathcal{A}_2$ and $\mathcal{C}_1 \subseteq \mathcal{C}_2$) iff the overlap of the two conditional facts is equal to the length of $\mathcal{A}_1 \leftarrow not\,\mathcal{C}_1$. The length of a conditional fact can be computed when it is constructed, we will assume that it is stored in an array *length*. Furthermore, we will use an array *overlap* to manage the overlap with the current conditional fact $\mathcal{A} \leftarrow not\,\mathcal{C}$. Of course, we cannot initialize it to zero every time when we want to check a new conditional fact. Therefore we use "lazy initialization": Every test is assigned a unique number *testno*, and we store in the array *lastset* the last time when the corresponding entry in *overlap* was initialized. So we refine the implementation of the procedure *insert* as follows:

> $testno := 0$;
> **procedure** $insert(\mathcal{A} \leftarrow not\ \mathcal{C})$:
> $testno := testno + 1$;
>   **foreach** $A \in \mathcal{A}$ **do**
>   **foreach** $\mathcal{A}' \leftarrow not\ \mathcal{C}' \in \Gamma$ **with** $A \in \mathcal{A}'$ **do**
>   **if** $inc\_overlap(\mathcal{A}' \leftarrow not\ \mathcal{C}',\ |\mathcal{A}| + |\mathcal{C}|)$ **then**
>   **return** ; /* $\mathcal{A} \leftarrow not\ \mathcal{C}$ is non-minimal */
>   **foreach** $C \in \mathcal{C}$ **do**
>   **foreach** $\mathcal{A}' \leftarrow not\ \mathcal{C}' \in \Gamma$ **with** $C \in \mathcal{C}'$ **do**
>   **if** $inc\_overlap(\mathcal{A}' \leftarrow not\ \mathcal{C}',\ |\mathcal{A}| + |\mathcal{C}|)$ **then**
>   **return** ; /* $\mathcal{A} \leftarrow not\ \mathcal{C}$ is non-minimal */
>   . . . /* Negative reduction can be implemented similarly */
> $\Gamma := \Gamma \cup \{\mathcal{A} \leftarrow not\ \mathcal{C}\}$;
> $length[\mathcal{A} \leftarrow not\ \mathcal{C}] := |\mathcal{A}| + |\mathcal{C}|$;
> $lastset[\mathcal{A} \leftarrow not\ \mathcal{C}] := 0$;

Of course, in order to get the mentioned efficiency, our data structures must allow a direct access on the conditional facts $\mathcal{A}' \leftarrow not\,\mathcal{C}'$ with $A \in \mathcal{A}'$ or $C \in \mathcal{C}'$.

Now the procedure *inc_overlap* increments the overlap with a conditional fact $\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'$ and checks whether it is a subset or superset of the given conditional fact $\mathcal{A} \leftarrow \mathit{not}\ \mathcal{C}$:

procedure $\mathit{inc\_overlap}(\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}', l)$: **bool**
  if $\mathit{lastset}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'] = \mathit{testno}$ **then**
    $\mathit{overlap}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'] := \mathit{overlap}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'] + 1;$
  else
    $\mathit{overlap}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'] := 1;$
    $\mathit{lastset}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'] := \mathit{testno};$
  if $\mathit{overlap}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'] = \mathit{length}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}']$ **then**
    return *true*; /* $\mathcal{A} \leftarrow \mathit{not}\ \mathcal{C}$ is non-minimal */
  if $\mathit{overlap}[\mathcal{A}' \leftarrow \mathit{not}\ \mathcal{C}'] = l$ **then**
    $\mathit{delete}(\mathcal{A}' \leftarrow \mathcal{C}');$
  return *false*;

This completes our discussion of the computation of derived conditional facts.

The next step in the algorithm are the reductions. We have already eliminated non-minimal conditional facts, and done negative reduction. However, when we now evaluate negative body literals to "true", the other two reductions can become applicable again. Therefore we use the procedure *insert* for the reduced conditional facts, which automatically performs the other reductions.

So our goal now is to find efficiently occurrences of negative body literals which do not occur in any head. We already above needed data structures to locate all positive and negative occurrences of an atom. Now we suggest to introduce a counter $\mathit{numpos}[A]$ for the number of positive occurrences of A. It can easily be updated during the computation of the derived conditional facts. Furthermore, we also maintain a set *nopos* of those atoms occurring in a body, but not in a head. Then positive reduction can be implemented as follows:

procedure *reduce*:
  while $\mathit{nopos} \neq \emptyset$ **do**
    let $C \in \mathit{nopos}$;
    foreach $\mathcal{A} \leftarrow \mathit{not}\ \mathcal{C} \in \Gamma$ **with** $C \in \mathcal{C}$ **do**
      $\Gamma := \Gamma - \{\mathcal{A} \leftarrow \mathit{not}\ \mathcal{C}\};$
      $\mathit{insert}\big(\mathcal{A} \leftarrow \mathit{not}\ (\mathcal{C} - \{C\})\big);$
    $\mathit{nopos} := \mathit{nopos} - \{C\};$

Probably, it would slightly increase the efficiency if we immediately check the other body literals of this rule, i.e. call $\mathit{insert}\big(\mathcal{A} \leftarrow \mathit{not}\ (\mathcal{C} - \mathit{nopos})\big)$ instead. But in order not to repeat these checks later, we would have to store for every conditional fact the version number of *nopos* with which it was last reduced. The reason for this is that *nopos* is increased through the reductions performed by the call to *insert*.

The procedure *delete* called in *inc_overlap* looks as follows:

procedure $delete(\mathcal{A}' \leftarrow not\ \mathcal{C}')$:
  foreach $A \in \mathcal{A}'$ **do**
    $numpos[A] := numpos[A] - 1$;
    if $numpos[A] = 0$ **then**
      $nopos := nopos \cup \{A\}$;

This does not ensure that A occurs in a negative body literal when it is inserted into *nopos*, but this is not important for correctness and termination.

So we can now compute the residual program with reasonable efficiency. Especially, with the access paths discussed, we *never* have to go through the complete set Γ of conditional facts. There are two really important further optimizations, which, however, need further research in the disjunctive case.

First, it is not necessary to delay positive reduction until we have computed all derivable conditional facts. When we evaluate *not* C to *true*, we must only be sure that C will not occur in any head. In the case of Horn programs, it is known how to order the rules according to the predicate dependencies, and iterate only strongly connected components. We can do the same for conditional facts, at least if the program is non-disjunctive (for a generalization to disjunctive facts see [Bra95]). But this allows us to do positive reduction for all predicates defined in "lower" SCCs. Therefore, if the program should be stratified, we can evaluate all negative body literals immediately (and thus get the standard bottom-up query evaluation algorithm for stratified databases). Furthermore, if there are only a few parts in the program which are not stratified, we have to delay only these negative body literals.

Second, as any bottom-up approach, our technique would profit from a "magic set" transformation. However, this is only possible for semantics satisfying the "relevance" condition [Dix92, DM94c, DM94b], which the two-valued stable semantics does not. So there can be no goal-directed query evaluatation algorithm for it. However, if our method is used for computing, e.g., the WFS, then a magic set transformation would drastically increase the efficiency. In the non-disjunctive case, standard techniques may be appropriately adjusted (see [DN95, KSS91]). We only have to ensure that if we derive a conditional fact containing *not* C in its body, then we also get $magic[C]$, so that rule instances possibly yielding C are not blocked. In the disjunctive case, no real magic set techniques are known, although ideas of [Dem91, Roy90, Bra95] might be used.

Finally, let us note that in the non-disjunctive case it is often possible (namely, when no predicate depends on itself positively and negatively at the same time) to introduce new predicates for conditional facts. This would allow to build an implementation on existing deductive databases.

5 Computing the Stable Semantics from the Completion

In the preceding sections, we have shown how to compute the residual program $\hat{\Phi}$. Let us repeat the two important properties of the residual program:

1. It is equivalent to the original program for a wide range of semantics (namely all semantics allowing the elementary transformations of Section 2).

2. It is much simpler than the original program, because it is ground, it contains no positive body literals, and usually only very few rules.

The truth value of most literals is already decided by the residual program, because they either are given as an unconditional fact or do not appear at all in it. We now have to treat the few difficult literals, about which conditions remained after the reductions. It is interesting to note that whatever we do with them, we get a semantics with the good properties of Section 2:

Theorem 16 [BD95b].
Let SEM be a semantics such that $SEM(\Phi) = SEM(\hat{\Phi})$ for every logic program Φ and its residual program $\hat{\Phi}$. Then SEM allows unfolding (GPPE), elimination of tautologies and non-minimal rules, and positive and negative reduction.

Note, however, that it is not guaranteed that any model of the residual program is also a model of the original program. So a model-theoretic semantics can only select certain models of the residual program, for instance, any subset of the supported models. It is subject of our future research to find a maximal class of three-valued models with this property.

But in this paper our main goal is to compute the stable model semantics. There we make use of a result of Fages and Ben-Eliyahu [AB94] that if there are no positive loops, the stable models coincide with the supported models. This is certainly the case for residual programs because they have no positive body literals at all. Furthermore, supported models correspond to Clark's completion [Cla78, BH86]. So if we take Clark's completion of the residual program, we get a syntactical characterization of the stable models. Of course, the previous results considered only the non-disjunctive case. However, the generalization is easy. In order to define Clark's completion for disjunctive rules, we simply view a rule like $p \vee q \leftarrow \mathit{not}\ r$ as a shorthand for the two rules $p \leftarrow \mathit{not}\ q \wedge \mathit{not}\ r$ and $q \leftarrow \mathit{not}\ p \wedge \mathit{not}\ r$.

Definition 17 (Completion of a Residual Program).
Let $\hat{\Phi}$ be an instantiated program without positive body literals. Then we define

$$
\begin{aligned}
comp^+(\hat{\Phi}) &:= \bigl\{A_1 \vee \cdots \vee A_m \vee C_1 \vee \cdots \vee C_n \bigm| \mathcal{A} \leftarrow \mathit{not}\, \mathcal{C} \in \hat{\Phi},\\
&\qquad \{A_1, \ldots, A_m\} = \mathcal{A},\\
&\qquad \{C_1, \ldots, C_n\} = \mathcal{C}\bigr\}.\\
comp^-(\hat{\Phi}) &:= \bigl\{\neg A \vee \neg C_1 \vee \cdots \vee \neg C_n \bigm| A \text{ is a ground atom},\\
&\qquad C_i \in \mathcal{C}_i \cup \bigl(\mathcal{A}_i - \{A\}\bigr),\\
&\qquad \text{where } \mathcal{A}_i \leftarrow \mathit{not}\, \mathcal{C}_i \in \hat{\Phi},\ i = 1, \ldots, n\\
&\qquad \text{are all conditional facts with } A \in \mathcal{A}_i\bigr\}.\\
comp(\hat{\Phi}) &:= comp^+(\hat{\Phi}) \cup comp^-(\hat{\Phi}).
\end{aligned}
$$

For instance, consider the following residual program $\hat{\Phi}$ and the completed definitions for p and q in the standard "if and only if"-notation:

$$
\begin{array}{lll}
\hat{\Phi}: & p \vee q \leftarrow \mathit{not}\ r. & \text{completion}: \quad p \leftrightarrow (\neg q \wedge \neg r) \vee (\neg s \wedge \neg t).\\
 & p \leftarrow \mathit{not}\ s \wedge \mathit{not}\ t. & \qquad\qquad\qquad q \leftrightarrow (\neg p \wedge \neg r).
\end{array}
$$

The direction $\leftarrow$ corresponds to the logical information in the given program, and is represented by the positive disjunctions above. The negative disjunctions are the clausal ("multiplied out") forms of the direction $\rightarrow$.

Now this completion of the residual program characterizes the stable models:

Theorem 18 (Completion Characterizes Stable Models).
A two-valued Herbrand interpretation I is a stable model of Φ iff $I \models comp(\hat{\Phi})$, where $\hat{\Phi}$ is the residual program of Φ.

So an answer θ to a query ψ is (skeptically) correct wrt the stable model semantics iff $comp(\hat{\Phi}) \vdash \psi\theta$. Any theorem prover can be used to check that (or generate answers during the proof of ψ). It is also possible to use standard model generating algorithms on $comp(\hat{\Phi})$ to construct the stable models of Φ. However, we can also make use of the very special structure of $comp(\hat{\Phi})$. In order to continue our computation bottom-up, we suggest to compute all positive consequences of $comp(\hat{\Phi})$. This is done by applying again the "hyperresolution" [CL73] technique used already in Definition 11. The negative disjunctions $\neg A_1 \vee \cdots \vee \neg A_n$ are interpreted as rules $\leftarrow A_1 \wedge \cdots \wedge A_n$. The body atoms are matched with the positive disjunctions. The result of the rule application consists of the disjunctive contexts. Note that it is possible that the empty disjunction is computed, which is interpreted as logically false. This happens if the completion is inconsistent, e.g. for the program $p \leftarrow not\ p$.

Definition 19 (Implied Positive Disjunctions).
Hyperresolution with rules from $comp^{-}(\hat{\Phi})$ is defined as follows:

$$H(\mathcal{D}) := \Big\{ \bigcup_{i=1}^{m} (\mathcal{A}_i - \{A_i\}) \;\Big|\; \mathcal{A}_i \in \mathcal{D},\ A_i \in \mathcal{A}_i,\ \neg A_1 \vee \cdots \vee \neg A_m \in comp^{-}(\hat{\Phi}) \Big\}.$$

We start with $\mathcal{D}_0 := comp^{+}(\hat{\Phi})$ and iterate $\mathcal{D}_i := H(\mathcal{D}_{i-1})$ until a fixpoint $\mathcal{D}_n$ is reached. Then let $comp^{*}(\hat{\Phi}) := \{\mathcal{A} \in \mathcal{D}_n \mid \text{for all } \mathcal{A}' \subset \mathcal{A}\text{: } \mathcal{A}' \notin \mathcal{D}_n\}$.

Let us illustrate this with an example. Consider the following residual program $\hat{\Phi}$ and its completion $comp(\hat{\Phi})$ written as facts and rules:

$\hat{\Phi}$:		$comp^{+}(\hat{\Phi})$:	$comp^{-}(\hat{\Phi})$:
$p \vee q.$		$p \vee q.$	$\leftarrow p \wedge q.$
$r \leftarrow not\ p.$		$r \vee p.$	$\leftarrow q \wedge p.$
$r \leftarrow not\ q.$		$r \vee q.$	$\leftarrow r \wedge p \wedge q.$
$s \leftarrow not\ r.$		$s \vee r.$	$\leftarrow s \wedge r.$

The second and the third rule are of course superfluous. We can derive r (with the first rule applied to the second and third disjunctive fact), and after deleting non-minimal disjunctions we get $\{p \vee q,\ r\}$. This directly describes the two stable models $I_1 := \{p, r\}$ and $I_2 := \{q, r\}$. In general, we have the following:

Theorem 20 (Meaning of Implied Positive Disjunctions).
I is a stable model of Φ iff I is a minimal model of $comp^{*}(\hat{\Phi})$.

Obviously, a positive disjunction is skeptically implied by the stable model semantics iff it or a subdisjunction of it is contained in $comp^+(\hat{\Phi})$. A negative ground atom is implied iff it does not occur in $comp^+(\hat{\Phi})$. Arbitrary negative disjunctions require a little more computation, but still can be answered efficiently from $comp^*(\hat{\Phi})$:

Theorem 21 (Computation of Negative Disjunctions).
A negative disjunction $\neg A_1 \vee \cdots \vee \neg A_n$ is skeptically implied by STABLE iff there are not $\mathcal{A}_1, \ldots, \mathcal{A}_n \in comp^*(\hat{\Phi})$ with $A_i \in \mathcal{A}_i$ $(i = 1, \ldots, n)$, and such that neither $\bigcup_{i=1}^{n} (\mathcal{A}_i - \{A_i\})$ nor a subdisjunction of it is contained in $comp^*(\hat{\Phi})$.

It is interesting to note that also the dual approach is possible: If we compute all implied negative disjunctions, the maximal models of the result are the stable models of the original program.

Let us now consider the computation of the stable models themselves. In the special case that there is only a single stable model, $comp^*(\hat{\Phi})$ is the set of atoms true in it, so no further computation is needed. If there are multiple models, we construct them by a case analysis. Note that any atom contained in a proper disjunction of $comp^*(\hat{\Phi})$ can really be true or false (of course, the decision on different atoms is not independent). So we also need no further computation if we are only interested in credulous truth of ground literals. In the general case, we choose a truth value for a ground atom, and then compute the implied positive disjunctions which follow from this new information. We repeat this process until no proper disjunctions remain:

> **procedure** $model\_generation\big(comp^+(\hat{\Phi}),\ comp^-(\hat{\Phi})\big)$:
>   **compute** $comp^*(\hat{\Phi})$;
>   **if** $comp^*(\hat{\Phi})$ contains no proper disjunction **then**
>   **print** $comp^*(\hat{\Phi})$;
>   **else**
>   **let** A be contained in a proper disjunction;
> $model\_generation\big(comp^+(\hat{\Phi}) \cup \{A\},\ comp^-(\hat{\Phi})\big)$;
> $model\_generation\big(comp^+(\hat{\Phi}),\ comp^-(\hat{\Phi}) \cup \{\neg A\}\big)$;

Note that by the completeness of hyperresolution, we can be sure that if we choose A to be true or false, there is really a stable model in which A has this truth value. So our algorithm never runs into "dead ends". This is an important difference to the algorithm used in [CW93] for propagating guessed truth values of atoms. For instance, the following residual program has only a single stable model (namely $\{p, \neg q, r\}$), but it is reduced in the sense of [CW93], so their algorithm has to guess a truth value of some atom:

$$\begin{aligned} p &\leftarrow not\ p. \\ p &\leftarrow not\ q. \\ q &\leftarrow not\ r. \\ r &\leftarrow not\ q. \end{aligned}$$

Of course, this example is quite pathological, and their reduction algorithm works well in the more usual simpler cases. However, it is also not clear how to generalize it to the disjunctive case.

Let us conclude this section by giving an implementation of the computation of $comp^*(\hat{\Phi})$. When we have $\hat{\Phi}$, it is a simple restructuring task to construct $comp^+(\hat{\Phi})$ and $comp^-(\hat{\Phi})$. The interesting point is the hyperresolution. Of course, in principle it would be possible to re-use the implementation of Section 4, but since we now have only ground literals (and no rule heads or negative body literals), a more specialized implementation can be faster.

It is interesting that hyperresolution can also be done by "pushing the facts through the rules", whereas in Section 4 the rules had the control. By using this idea, we immediately get a semi-naive behaviour (doing no derivation twice). The approach is also natural because we now often have more rules than facts. In the following algorithm, we use the set Q to "queue" the disjunctions we still have to treat. The set $\mathcal{D}$ is the result, i.e. at the end it will be $comp^*(\hat{\Phi})$:

> $\mathcal{D} := \emptyset$;
> $Q := comp^+(\hat{\Phi})$;
> **while** $Q \neq \emptyset$ **do**
>   **let** $\mathcal{A} \in Q$;
> $Q := Q - \{\mathcal{A}\}$;
> $ins\_dis(Q, \mathcal{A})$; /* $\mathcal{D} := \mathcal{D} \cup \{\mathcal{A}\}$ */
> $conseq(\mathcal{A})$;
> **print** $\mathcal{D}$;

Besides inserting a disjunctive fact into a set, the procedure *ins_dis* also eliminates non-minimal disjunctions from $\mathcal{D} \cup Q$ (as shown for *insert* in Section 4). The procedure *conseq* computes the immediate consequences using the disjunctive fact $\mathcal{A}$ and possibly other facts in $\mathcal{D}$:

> **procedure** $conseq(\mathcal{A})$:
>   **foreach** $A \in \mathcal{A}$ **do**
>   **foreach** $\leftarrow \mathcal{B} \in comp^-(\hat{\Phi})$ **with** $A \in \mathcal{B}$ **do**
> $resolve(\mathcal{B} - \{A\},\ \mathcal{A} - \{A\})$;

The procedure *resolve* matches the remaining body literals with disjunctive facts from $\mathcal{D}$:

> **procedure** $resolve(\mathcal{B}, \mathcal{A})$:
>   **if** $\mathcal{B} = \emptyset$ **then**
> $ins\_dis(Q, \mathcal{A})$; /* $Q := Q \cup \{\mathcal{A}\}$ */
>   **else**
>   **let** $A \in \mathcal{B}$;
>   **foreach** $\mathcal{A}' \in \mathcal{D}$ **with** $A \in \mathcal{A}'$ **do**
> $resolve\big(\mathcal{B} - \{A\},\ \mathcal{A} \cup (\mathcal{A}' - \{A\})\big)$;

6 Comparison With Other Bottom-Up Algorithms

We claimed that our approach can also be used as a framework for understanding and comparing other query evaluation algorithms. Let us consider for example the algorithm proposed in [KSS91] for computing the WFS of non-disjunctive programs. The well-founded semantics follows from the residual program by making all difficult atoms undefined:

Theorem 22 (Computation of WFS) [BD95b].
The well-founded model of a non-disjunctive program Φ can be derived from its residual program $\hat{\Phi}$ as follows:

1. *If $A \leftarrow true \in \hat{\Phi}$, then A is true in the well-founded model.*
2. *If A does not occur in $\hat{\Phi}$, then A is false in the well-founded model.*
3. *All other ground atoms are undefined in the well-founded model.*

The algorithm of [KSS91] is based on VAN GELDER's alternating fixpoint method. In order to understand it in our framework, the main difference is that they store not the complete conditional facts, but only their head literals and one further bit indicating whether there is a non-trivial body or not. This is done by introducing two versions of every IDB-predicate: The predicate p contains the facts currently known to be true (the conditional facts with body "*true*"), while the predicate p' contains the facts currently not known to be false (the heads of all conditional facts). It is of course a loss of information to code the rule bodies in a single bit, but this can be compensated by recomputing them in each step of the reduction phase.

Let us consider as an example the following rule (similar to the classical "winning state" example, but the "move" relation is computed by a join):

$$p(X) \leftarrow q(X,Y) \wedge r(Y,Z) \wedge \neg p(Z).$$

We assume that q and r are EDB-relations. In our approach, we start by computing the conditional facts as follows:

$$p\_if\_not\_p(X,Z) \leftarrow q(X,Y) \wedge r(Y,Z).$$

In order to simplify the comparision, we have introduced a new predicate to store the conditional facts. Obviously, $p\_if\_not\_p(X,Z)$ represents $p(X) \leftarrow \neg p(Z)$.

An optimization applied in [KSS91] is to precompute the surely true facts and to combine the first derivation of conditional facts already with the first reduction, but in this example p is simply initialized to the empty relation (since there are no purely positive rules about p). Then the first step of [KSS91] is

$$p'(X) \leftarrow q(X,Y) \wedge r(Y,Z) \wedge \neg p(Z).$$

Since $\neg p(Z)$ is always true, this exactly corresponds to our first step, only the bodies of the derived conditional facts are forgotten.

Now in our approach, we apply iteratively the reduction operator R. On the one hand, this produces facts which are known to be true:

$$p(X) \leftarrow p\_if\_not\_p(X, Z) \wedge \neg p\_if\_not\_p(Z, \_).$$

The same facts are computed in the next step of the [KSS91]-approach:

$$p(X) \leftarrow q(X, Y) \wedge r(Y, Z) \wedge \neg p'(Z).$$

On the other hand, we reduce the pending conditional facts by the following deletion:

$$p\_if\_not\_p(X, Z) \mathrel{-=} p\_if\_not\_p(X, Z) \wedge p(Z).$$

Again, the same is done in [KSS91]:

$$p'(X) \leftarrow q(X, Y) \wedge r(Y, Z) \wedge \neg p(Z).$$

Both approaches iterate these two steps until a fixpoint is reached.

To summarize, the difference between both approaches is that we explicitly represent the conditional facts, whereas [KSS91] recompute them in each step. It is also interesting to note that our final set of conditional facts still contains enough information to compute the stable models, whereas this information is lost by reducing them only to their heads.

7 Conclusions

We proposed a general framework for computing semantics of disjunctive logic programs. Especially important for our approach is the residual program which can be computed by delaying negative body literals unless their truth value is trivial. We have shown that it is equivalent to the original program for a wide range of semantics. We also gave a detailed explanation how to compute the residual program bottom-up (including many interesting optimizations).

We illustrated our method in the case of the disjunctive stable semantics, where we used a disjunctive extension of CLARK's completion to characterize the stable models syntactically. Furthermore, we also considered the non-disjunctive well-founded semantics, and compared our approach to the algorithm of [KSS91].

A first prototype implementation of our approach can be obtained from "ftp://ftp.informatik.uni-hannover.de/software/index.html". We plan to further develop this program and investigate additional optimizations.

Furthermore, we will of course look at other semantics, especially we are working on PRZYMUSINSKI's stationary and static semantics. It is also interesting what happens when we use a three-valued completion instead of the two-valued CLARK's completion (we will in general not get the three-valued stable models, since these can violate GPPE in the disjunctive case).

Acknowledgement

The comments of Hendrik Decker, Udo Lipeck, and Teodor Przymusinski have been very helpful. We would like to thank also Dirk Hillbrecht and Michael Salzenberg for implementing our approach.

References

[AB94] Krzysztof R. Apt and Roland N. Bol. Logic programming and negation: A survey. *The Journal of Logic Programming*, 19/20:9–71, 1994.

[BD95a] Stefan Brass and Jürgen Dix. Caracterizations of the Stable Semantics by Partial Evaluation. In A. Nerode, W. Marek, and M. Truszczyński, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Third International Conference*. Springer LNCS, to appear, June 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*. MIT, June 1995.

[BH86] 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.

[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. Springer-Verlag, 1992.

[BL93] Stefan Brass and Udo W. Lipeck. Bottom-up query evaluation with partially ordered defaults. In *Proceedings of the 3rd International Conference on Deductive and Object-Oriented Databases (DOOD'93)*, pages 253–266, LNCS 769. Springer, 1993.

[BNNS93] Colin Bell, Anil Nerode, Raymond Ng, and V. S. Subrahmanian. Implementing stable semantics by linear programming. In Luís Moniz Pereira and Anil Nerode, editors, *Logic Programming and Non-monotonic Reasoning, Proc. of the Second Int. Workshop (LPNMR'93)*, pages 23–42. MIT Press, 1993.

[Bra90] Stefan Brass. Beginnings of a theory of general database completions. In Serge Abiteboul and Paris C. Kanellakis, editors, *Third International Conference on Database Theory (ICDT'90)*, number 470 in LNCS, pages 349–363. Springer-Verlag, 1990.

[Bra95] Stefan Brass. Bottom-up query evaluation in disjunctive deductive databases. Research report, Institut für Informatik, Universität Hannover, 1995.

[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. Springer, 1990.

[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 Lanzarone. Static Semantics as Program Transformation and Well-founded Computation. this Volume, pages 156–180, 1995.

[Cla78] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data-Bases*, pages 293–322. Plenum, New York, 78.

[CW93] Weidong Chen and David S. Warren. Computation of stable models and its integration with logical query processing. Technical report, SUNY at Stony Brook, 1993.

[Dem91] Robert Demolombe. An efficient strategy for non-horn deductive databases. *Theoretical Computer Science*, 78:245–259, 1991.

[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: Proc. of the Third Int. Conf. (KR'92)*, pages 591–602. Morgan Kaufmann, 1992.

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

[Dix95b] Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, XXII: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. Proceedings of the Konstanz Colloquium in Logic and Information (LogIn '92)*, pages 227–313. DeGruyter, 1995.

[DK89a] Phan Minh Dung and Kanchana Kanchansut. A fixpoint approach to declarative semantics of logic programs. In *Proc. NACLP'89*, pages 604–625, 1989.

[DK89b] Phan Minh Dung and Kanchana Kanchansut. A natural semantics of logic programs with negation. In *Proc. of the Ninth Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 70–80, 1989.

[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. Lisbon, MIT Press, July 1993.

[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*, pages 511–520. Springer, Lecture Notes in Artificial Intelligence 869, 1994.

[DM94c] Jürgen Dix and Martin Müller. The Stable Semantics and its Variants: A Comparison of Recent Approaches. In L. Dreschler-Fischer and B. Nebel, editors, *Proceedings of the 18th German Annual Conference on Artificial Intelligence (KI '94), Saarbrücken, Germany*, pages 82–93. Springer, Lecture Notes in Artificial Intelligence 861, 1994.

[DN95] Lars Degerstedt and Ulf Nilsson. Magic Computation of Well-founded Semantics. This volume, pages 127–155, 1995.

[GL91b] 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.).

[HY91] Yong Hu and Li Yan Yuan. Extended well-founded model semantics for general logic programs. In Koichi Furukawa, editor, *Proc. of the 8th Int. Conf. on Logic Programming (ICLP'91)*, pages 412–425. MIT Press, 1991.

[KLM90] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.

[KSS91] David B. Kemp, Peter J. Stuckey, and Divesh Srivastava. Magic sets and bottom-up evaluation of well-founded models. In *Proc. of the 1991 Int. Symposium on Logic Programming*, pages 337–351. MIT Press, 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. Springer, 1982.

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

[PAA91] Luís Moniz Pereira, Joaquim N. Aparício, and José J. Alferes. Derivation procedures for extended stable models. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 863–868, 1991.

[Prz91a] 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.).

[Prz91b] Teodor C. 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. Springer-Verlag, 1991.

[Prz95] Teodor Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 1995. To appear.

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

[Ros90] Kenneth A. Ross. The well founded semantics for disjunctive logic programs. In Won Kim, Jean-Marie Nicolas, and Shojiro Nishio, editors, *Deductive and Object-Oriented Databases, Proc. of the First International Conference (DOOD'89)*, pages 385–402. North-Holland, 1990.

[Roy90] Véronique Royer. Backward chaining evaluation in stratified disjunctive theories. In *Proc. of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'90)*, pages 183–195, 1990.

[RSS94] Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. Rule ordering in bottom-up fixpoint evaluation of logic programs. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):501–517, 1994.

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

[Sei94] Dietmar Seipel. An efficient computation of the extended generalized closed world assumption by support-for-negation sets. In *Proc. Int. Conf. on Logic*

Programming and Automated Reasoning (LPAR'94), number 822 in LNAI, pages 245–259. Springer, 1994.

[SS94] Chiaki Sakama and Hirohisa Seki. Partial Deduction of Disjunctive Logic Programs: A Declarative Approach. In *Fourth International Workshop on Logic Program Synthesis and Transformation (LOPSTR'94)*. Lecture Notes in Computer Science, Springer-Verlag, July 1994.

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

A Proofs

A.1 Proof of Theorem 12

Our goal is to show that it is possible to reach $\mathrm{lfp}(T_\Phi)$ from Φ by a series of GPPE- and "Elimination of Tautology"-steps. In this process, we have to be very careful with non-minimal conditional facts, since we allow semantics for which they are relevant. The problem is illustrated by the following examples:

Example 1. We once hoped that if Φ' results from Φ by an application of GPPE, then $\mathrm{lfp}(T_\Phi) = \mathrm{lfp}(T_{\Phi'})$. This, however, does not hold. Let Φ be:

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

For this program, we get $\mathrm{lfp}(T_\Phi) = \{q \vee r, p \vee q, r, p \vee r\}$. Now if we apply GPPE to 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, it is the source of the problem, because with it it is possible to derive $p \vee q \vee r$ which is not in $\mathrm{lfp}(T_\Phi)$.

Example 2. In the above example, it would have been possible to simply eliminate the tautology before further applications of GPPE to avoid the derivation of the critical conditional fact. But this is not always correct, because if the program Φ 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 $\mathrm{lfp}(T_\Phi)$.

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

$$\begin{aligned}&p \leftarrow q \wedge r.\\&q \vee r.\\&r \vee s.\end{aligned}$$

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

Lemma 23. *Let Φ be an instantiated 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 Φ' contain in addition the result of an unfolding step, i.e.

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

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

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 $\mathcal{A}_i$, $i = 1, \ldots, m$, and $\mathcal{A}_j'$, $j = 1, \ldots, n$. Then the resulting disjunctive fact is

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

Now we have to distinguish several cases to show that this conditional fact is also derivable in Φ. 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 $B_0 \vee \mathcal{A}' \leftarrow B_1' \wedge \cdots \wedge B_n'$ and get

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

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

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

   Of course, $\mathcal{A}'$ does not contain B_0, so this set-difference is effectless and $(\mathcal{A}' - \{B_0\})$ can be simplified to $\mathcal{A}'$. However, we have to assume that the $\mathcal{A}_j'$, $j = 1, \ldots, n$, 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 $\mathcal{A} \leftarrow B_0 \wedge B_1 \wedge \cdots \wedge B_m$ and enter $\mathcal{A}'_1$ for the body literal B_0. This results in

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

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

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

Now let us check that this is equivalent to $(*)$. First, $\{B_0\} \cup (\mathcal{A}'_1 - \{B_0, B'_1\})$ 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 $i = 1, \ldots, m$ (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

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

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

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

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

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

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

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

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:

$$\begin{aligned}\{B_0\} \cup \mathcal{A}' \cup \mathcal{A}_1 \cup \mathcal{A} \cup \bigl(\mathcal{A}'_1 - \{B'_2\}\bigr) \cup \bigl(\mathcal{A}'_2 - \{B'_2\}\bigr) & \\ \cup \bigcup_{i=2}^{m} \bigl(\mathcal{A}_i - \{B_i, B'_2\}\bigr) \cup \bigcup_{j=3}^{n} \bigl(\mathcal{A}'_j - \{B'_j\}\bigr). &\end{aligned}$$

   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 24. *Let Φ 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

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

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

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 12 ($\Phi \mapsto \mathrm{lfp}(T_\Phi)$ is a SEM-Equivalence Transformation). *Let SEM be a semantics which satisfies GPPE, and allows the elimination of tautologies. Then* $SEM(\Phi) = SEM\big(\mathrm{lfp}(T_\Phi)\big)$.

Proof. We have to construct $\mathrm{lfp}(T_\Phi)$ from Φ by using only semantics preserving steps. This works as follows:

1. First, we completely instantiate the given program Φ. By our definition of a semantics, we have $\mathrm{SEM}(\Phi) = \mathrm{SEM}(\Phi^*)$. Note also that Φ^* is finite because of our restriction on Σ.
2. Next, we add the tautological rule $A \leftarrow A$ for every ground atom A. Since tautologies are irrelevat for SEM it is not only possible to delete them, but also to add them without changing the semantics of the given program. Because of these tautologies, applications of GPPE never delete the original rule, they only add the partially evaluated rules to the program.
3. We now apply GPPE until nothing changes. However, directly after every appliction of GPPE, 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 23 and Lemma 24. This means especially that all generated conditional facts are contained in $\mathrm{lfp}(T_\Phi)$. On the other hand, every conditional fact in $\mathrm{lfp}(T_\Phi)$ is generated: For instance, consider the rule $\mathcal{A} \leftarrow B_1 \wedge \cdots \wedge B_n \wedge \mathit{not}\ \mathcal{C}$ applied to conditional facts $\mathcal{A}_i \leftarrow \mathit{not}\ \mathcal{C}_i$. The result contained in $\mathrm{lfp}(T_\Phi)$ is

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

By applying GPPE to the first body literal, we get

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

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

4. Finally, we delete all rules which still have positive body literals by applying GPPE. 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, an application of GPPE to 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 created by an application of GPPE. 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 phase 2. Therefore, the set of conditional facts does not increase, and after all other rules are eliminated, $\mathrm{lfp}(T_\Phi)$ remains. □

A.2 Proof of Theorems 18, 20, 21

In order to prove that Clark's completion of the residual program characterizes the stable models, we use the supported models as an intermediate step. Note that we cannot formally use the result of Fages and Ben-Eliyahu because we consider disjunctive logic programs. First we have to generalize the notion of a supported model:

Definition 25 (Supported Model).
A two-valued Herbrand-interpretation I is a supported model of a logic program Φ iff for every ground atom A with $I \models A$ there is a ground instance $\mathcal{A} \leftarrow \mathcal{B} \wedge \mathit{not}\ \mathcal{C}$ of a rule in Φ such that $I \models \mathcal{B} \wedge \mathit{not}\ \mathcal{C}$ and $I \not\models \mathcal{A} - \{A\}$.

Lemma 26.

1. *If Φ has no positive body literals, then a supported model of Φ is also stable.*
2. *For any Φ, a stable model of Φ is also supported.*

Proof. 1. Let I be a supported model of Φ. Clearly, I is also a supported model of Φ/I. But Φ/I is a set of disjunctive facts of the form $\mathcal{A} \leftarrow \mathit{true}$. Now the supportedness-condition means that every true atom occurs in a disjunction where all other atoms are false. So no smaller model than I can still be a model of Φ/I.

2. Let I be a stable model of Φ, i.e. a minimal model of Φ/I. Let A be an atom true in I and $I' := I - \{A\}$. So I' is smaller than I and must violate a rule $\mathcal{A} \leftarrow \mathcal{B}$ in Φ/I. This means especially $I' \models \mathcal{B}$, and therefore $I \models \mathcal{B}$. Furthermore, I satisfies also the negative body literals (because the rule is contained in Φ/I). Finally we have $I' \not\models \mathcal{A}$, i.e. $I \not\models \mathcal{A} - \{A\}$, so A is contained in $\mathcal{A}$, and this rule supports it. □

Lemma 27.
Let $\hat{\Phi}$ be an instantiated program without positive body literals. I is a supported model of $\hat{\Phi}$ iff $I \models comp(\hat{\Phi})$.

Proof. 1. Let I be a supported model of $\hat{\Phi}$, but suppose that it does not satisfy one of the constructed negative disjunctions, e.g. $\neg A \vee \neg C_1 \vee \cdots \vee \neg C_n$. So $I \models A$, but A is not supported, since $I \models C_i$ makes the rule $\mathcal{A}_i \leftarrow not\ \mathcal{C}_i$ not applicable as a support for A: Either one of the negative body literals is false, or another head literal is true.
2. Let $I \models comp(\hat{\Phi})$, but suppose that I were not a supported model of $\hat{\Phi}$. Since $I \models comp^{+}(\hat{\Phi})$, I obviously is a model of $\hat{\Phi}$. So it is only possible that the support for some ground atom A is missing in I. Let $\mathcal{A}_i \leftarrow not\ \mathcal{C}_i$ ($i = 1, \ldots, n$) be all rules with $A \in \mathcal{A}_i$. Since none of these supports A, in every rule either a negative body literal must be false in I, or another head literal besides A must be true. So choose the corresponding atoms as C_i. Then I does not satisfy $\neg A \vee \neg C_1 \vee \cdots \vee \neg C_n$, which is a contradiction. □

Theorem 18 (Completion Characterizes Stable Models).
A two-valued Herbrand interpretation I is a stable model of Φ iff $I \models comp(\hat{\Phi})$, where $\hat{\Phi}$ is the residual program of Φ.

Proof. This is now an immediate consequence of Theorem 12 and Lemmas 26 and 27. □

Lemma 28.
Let $\hat{\Phi}$ be an instantiated program without positive body literals. An interpretation I is a model of $comp(\hat{\Phi})$ iff it is a minimal model of $comp^{*}(\hat{\Phi})$.

Proof. 1. Let I be a minimal model of $comp^{*}(\hat{\Phi})$, and let $I_1, \ldots, I_m$ be all models of $comp(\hat{\Phi})$. We have to show that I is among the I_i. Suppose it were not. Since I is a minimal model of $comp^{*}(\hat{\Phi})$, and the I_i are also models of $comp^{*}(\hat{\Phi})$ (since $comp(\hat{\Phi}) \vdash comp^{*}(\hat{\Phi})$), there must be ground atoms A_i with $I_i \models A_i$ and $I \not\models A_i$ (if for some I_i there were no such atom, we would have $I_i \preceq I$, and together with $I_i \neq I$ this would imply $I_i \prec I$). Now consider the disjunction $A_1 \vee \cdots \vee A_m$. It (or a subdisjunction of it) is contained in $comp^{*}(\hat{\Phi})$, because all the models of $comp(\hat{\Phi})$ satisfy it, i.e. $comp(\hat{\Phi}) \vdash A_1 \vee \cdots \vee A_m$. However, I does not satisfy it, which is a contradiction.
2. Now let I be a model of $comp(\hat{\Phi})$. Since $comp(\hat{\Phi}) \vdash comp^{*}(\hat{\Phi})$, I is also a model of $comp^{*}(\hat{\Phi})$. Now suppose that there were a smaller model I_1 of $comp^{*}(\hat{\Phi})$, and let I_0 be a minimal model of $comp^{*}(\hat{\Phi})$ with $I_0 \preceq I_1$, i.e. $I_0 \prec I$. By the already proven other direction, we get $I_0 \models comp(\hat{\Phi})$. Let A be a ground atom with $I \models A$ and $I_0 \not\models A$. By Lemma 27, I is a supported model of $\hat{\Phi}$, so let $\mathcal{A} \leftarrow not\ \mathcal{C}$ be the supporting rule for A. Since $I \models not\ \mathcal{C}$, and $I_0 \prec I$, it follows that $I_0 \models not\ \mathcal{C}$. In the same way, $I \not\models \mathcal{A} - \{A\}$ implies $I_0 \not\models \mathcal{A} - \{A\}$. Finally, we know $I_0 \not\models A$. But this means $I_0 \not\models \mathcal{A} \leftarrow not\ \mathcal{C}$, which is a contradiction. □

Theorem 20 (Meaning of Implied Positive Disjunctions).
I is a stable model of Φ iff I is a minimal model of $comp^*(\hat{\Phi})$.

Proof. This follows immediately from Theorem 18 and Lemma 28. □

Theorem 21 (Computation of Negative Disjunctions).
A negative disjunction $\neg A_1 \vee \cdots \vee \neg A_n$ is skeptically implied by STABLE iff there are not $\mathcal{A}_1, \ldots, \mathcal{A}_n \in comp^*(\hat{\Phi})$ with $A_i \in \mathcal{A}_i$ $(i = 1, \ldots, n)$, and such that neither $\bigcup_{i=1}^{n}(\mathcal{A}_i - \{A_i\})$ nor a subdisjunction of it is contained in $comp^*(\hat{\Phi})$.

Proof.
1. Let $\neg A_1 \vee \cdots \vee \neg A_n$ be skeptically implied by STABLE, i.e. true in every minimal model of $comp^*(\hat{\Phi})$. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n \in comp^*(\hat{\Phi})$ with $A_i \in \mathcal{A}_i$ $(i = 1, \ldots, n)$. Since one of the A_i is false in every minimal model of $comp^*(\hat{\Phi})$, the corresponding rest disjunction $\mathcal{A}_i - \{A_i\}$ must be true in that minimal model, so $\bigcup_{i=1}^{n}(\mathcal{A}_i - \{A_i\})$ is true in every minimal model of $comp^*(\hat{\Phi})$, and since it is a positive disjunction, it is also true in every model. By the transitivity of $\vdash$, all positive disjunctions which follow from $comp^*(\hat{\Phi})$, are already represented by a subdisjunction in $comp^*(\hat{\Phi})$.
2. Now let $\neg A_1 \vee \cdots \vee \neg A_n$ be not skeptically implied by STABLE, so there is a stable model I of $\hat{\Phi}$ with $I \models A_i$ $(i = 1, \ldots, n)$. By Lemma 28, I is a minimal model of $comp^*(\hat{\Phi})$, therefore $I_i := I - \{A_i\}$ cannot be a model of it, i.e. there must be a disjunction $\mathcal{A}_i$ with $I_i \not\models \mathcal{A}_i$. Since I satisfies it, $A_i \in \mathcal{A}_i$ follows. However, $\bigcup_{i=1}^{n}(\mathcal{A}_i - \{A_i\})$ is false in I, so neither it nor a subdisjunction of it can be contained in $comp^*(\hat{\Phi})$. □

This article was processed using the LaTeX macro package with LLNCS style