An Efficient Computation of the Extended Generalized Closed World Assumption by Support–for–Negation Sets

Dietmar Seipel

University of Tübingen
Sand 13, D – 72076 Tübingen, Germany
seipel@informatik.uni-tuebingen.de

Abstract. Closed world assumptions are one of the major approaches for non–monotonic reasoning in artificial intelligence. In [16] this formalism is applied to disjunctive logic programs, i.e. logic programs with positive disjunctive rule heads and positive atoms in the rule bodies. The disjunctive closure operator $\mathcal{T}_P^S$ allows for the derivation of the set $\mathcal{MS}_P$ of all positive disjunctive clauses logically implied by the program P, the *minimal model state*. On the other hand, disjunctive clauses over negated atoms are derived in the *extended generalized closed world assumption* $\mathcal{EGCWA}_P$. Such a clause is the negation of a conjunction of positive atoms. $\mathcal{EGCWA}_P$ contains all conjunctions which are *false* in all minimal Herbrand models of the program.

We present efficient Δ–iteration techniques for computing the closed world assumption $\mathcal{EGCWA}_P$ based on an iterative computation of the set of all support–for–negation sets $SN(C)$ for conjunctions C, i.e. certain sets of clauses which characterize $\mathcal{EGCWA}_P$: $C \in \mathcal{EGCWA}_P$ iff $\mathcal{MS}_P \models SN(C)$. The support–for–negation sets $SN(A)$ for atoms A are easily derived from the minimal model state $\mathcal{MS}_P$. We will propose a *bottom–up* computation deriving the support–for–negation sets of longer conjunctions from shorter ones based on an algebraic formula given by [16]: $SN(C_1 \wedge C_2) = SN(C_1) \vee SN(C_2)$. We will present techniques for the efficient computation of these disjunctions of two clause sets and a Δ–iteration approach for computing the support–for–negation sets of a sequence of growing minimal model states.

For disjunctive normal logic programs, i.e. logic programs with positive disjunctive rule heads and – possibly – negated atoms in the rule bodies, these operators form a basis for computing the *generalized disjunctive well–founded semantics*.

1 Introduction

Deductive databases use logic programming and techniques from non–monotonic reasoning for building a powerful, declarative language for specifying data and queries in relational databases. There has been a great deal of research on the processing of definite Horn clauses with definite facts, cf. [3], [22], [4]. Recursive programs are evaluated iteratively in a set–oriented fashion based on the

primitive operations of relational algebra, which can be executed efficiently on databases. The desired result is the least fixpoint of the definite consequence operator $\mathcal{T}_P$, which equals the minimal Herbrand model $\mathcal{M}_P$, cf. [15]. More advanced applications require the support of disjunctive rules and disjunctive facts: The deduction of independencies in probabilistic reasoning, cf. [17], [21], and the overlapping of rectangles in spatial databases, cf. [1], are examples falling into this category.

In general, a disjunctive normal logic rule has a head given by a disjunction of positive atoms and a body given by a conjunction of positive and negated atoms. Disjunctive logic programs are programs consisting of indefinite (disjunctive) rules without negated atoms in the body. They specify a set of logically implied disjunctive facts (i.e. positive ground clauses), the so–called *minimal model state* $\mathcal{MS}_P$, cf. [16]. Efficient techniques for computing $\mathcal{MS}_P$ iteratively as the least fixpoint of the disjunctive consequence operator $\mathcal{T}_P^S$ are currently investigated. Various semantics for indefinite rules with negation are given in literature: well–founded semantics [23], generalized well–founded semantics [2], generalized disjunctive well–founded semantics [16], stationary semantics [18] and stable model semantics [12], [11]. A classification of these semantics w.r.t. certain criteria is given by [8].

The *generalized disjunctive well–founded semantics* of [16] for disjunctive normal logic programs is based on four fundamental operators for deriving *true* ground disjunctions of only positive atoms or only negated atoms. It is three–valued, since not all disjunctions or conjunctions are assigned a truth value *true* or *false*. Some remain *unknown*.

Two of the operators, the positive and the negative consequence operator $\mathcal{T}_S^D$ and $\mathcal{F}_S^D$, respectively, operate on disjunctive normal logic programs and so–called state pairs $S =< T, F >$, where T and F is a set of disjunctions and conjunctions, respectively, of positive atoms, the disjunctive and the conjunctive Herbrand state. The disjunctions in T are considered *true*, the conjunctions in F are considered *false*. $\mathcal{T}_S^D$ derives some new *true* disjunctions form S and the program P, thus enlarging the disjunctive Herbrand state T. $\mathcal{F}_S^D$ derives new *false* atoms (atomic conjunctions) from S and P, thus enlarging the conjunctive Herbrand state F. In two separate iterations the fixpoints of $\mathcal{T}_S^D$ and $\mathcal{F}_S^D$ are computed, the so–called disjunctive well–founded semantics of P. It is three–valued as well and it generalizes the two–valued well–founded semantics for normal logic programs – with definite rule heads – to disjunctive normal logic programs. $\mathcal{T}_S^D$ and $\mathcal{F}_S^D$ will not be considered further in this paper.

Two other operators act on the disjunctive transformation $P = Dis(P', S)$ of a given disjunctive normal logic program P' w.r.t. an actual state pair S. The disjunctive consequence operator $\mathcal{T}_P^S$ allows for the derivation of the *minimal model state* $\mathcal{MS}_P$. $\mathcal{MS}_P$ can be characterized as the set of all ground disjunctions which are *true* in all Herbrand models of P, or equivalently as the set of all ground disjunctions logically implied by the program P. On the other hand, conjunctive clauses are derived by the *extended generalized closed world assumption*: $\mathcal{EGCWA}_P$ contains all conjunctions which are *false* in all minimal

Herbrand models of the program P. Equivalently, the corresponding disjunctions of negated atoms in the set $\neg\mathcal{EGCWA}_P$ are considered *true*. It is proven in [16], that $\neg\mathcal{EGCWA}_P$ is the maximal set of such disjunctions, which is consistent with the minimal model state $\mathcal{MS}_P$ of the program in the sense, that no new positive disjunctions can be derived from their union.

By theses operators, repeatedly minimal model states and extended generalized closed world assumptions are computed w.r.t. growing state pairs.

We present efficient Δ–iteration techniques for computing the closed world assumption $\mathcal{EGCWA}_P$ based on an iterative computation of the set of all support–for–negation sets $SN(C)$ for conjunctions C, which characterize $\mathcal{EGCWA}_P$: $C \in \mathcal{EGCWA}_P$ iff $\mathcal{MS}_P \models SN(C)$. The support–for–negation sets $SN(A)$ for atoms A are easily derived from the minimal model state $\mathcal{MS}_P$ by resolving all disjunctions with $\neg A$. Normally, if only a single support–for–negation set $SN(C)$ is required, it is faster to derive it by SLI–resolution, cf. [16]. Since we need all of them for deriving the whole $\mathcal{EGCWA}_P$, we use an algebraic approach based on the simple formula $SN(C_1 \wedge C_2) = SN(C_1) \vee SN(C_2)$ proven in [16]. It allows for the computation of support–for–negation sets of longer conjunctions from shorter ones in a *bottom–up* computation with the following advantages:

- A lot of *redundancy* is avoided, since the same operations can be shared between many different support derivations in bottom–up computation.
- *Database techniques* like massive joins can be adapted and used.

The rest of this paper is organized as follows: We will review some of the definitions and theoretical foundations for disjunctive logic programming and their fixpoint semantics given in [16]. Following, we will describe an efficient bottom–up Δ–iteration technique with subsumption for disjunctive logic programs that iteratively derives the disjunctive facts of $\mathcal{MS}_P$. Then we will turn to the more general disjunctive normal logic programs. We will describe the extended generalized closed world assumption $\mathcal{EGCWA}_P$ and again some bottom–up Δ–iteration techniques for its computation.

2 Foundations of Disjunctive Logic Programming

Syntactically, a *deductive database* $DDB = (DB, P)$ is given by a set DB of facts, which resemble a relational database, and a logic program P, which consists of a set of rules for deriving new facts from DB.

(i) Given some atoms $p_i(X_i)$ $(1 \leq i \leq m,\ m \in \mathbb{N}_+)$ and some literals $L_j(Y_j)$ $(1 \leq j \leq n,\ n \in \mathbb{N}_0)$. Then $p_1(X_1) \vee \ldots \vee p_m(X_m)$ is a disjunctive *fact* and

$$r: \quad p_1(X_1) \vee \ldots \vee p_m(X_m) \ \text{:-}\ L_1(Y_1), \ldots, L_n(Y_n).$$

is a disjunctive normal *rule*, where the atoms $p_1(X_1), \ldots, p_m(X_m)$ form the rule's head and the literals $L_1(Y_1), \ldots, L_n(Y_n)$ form its body. r is disjunctive, iff all its body literals L_i are positive. r is normal, iff $m = 1$.
The disjunctive normal logic program containing all the *ground instances* of rules in P is denoted by $Ground(P)$.

The *semantics* of a deductive database $DDB = (DB, P)$ is defined according to the bottom–up or naive evaluation algorithms, cf. [16], as the least fixpoint of a consequence operator $\mathcal{T}_P^S$.

(ii) The *disjunctive Herbrand base* DHB_P of a disjunctive logic program P is given by the set of positive ground clauses

$$DHB_P \;=\; \{\; A_1 \vee \ldots \vee A_k \mid k \geq 1,\; A_i \in HB_P,\; 1 \leq i \leq k \;\},$$

where HB_P denotes the Herbrand base of P. A *disjunctive Herbrand state* – or simply Herbrand state – D of P is a subset $D \subseteq DHB_P$. The *canonical form* and the *expansion* of D are

$$can(D) = \{\; C \in D \mid \text{there is no proper subclause } C' \text{ of } C \text{ in } D \;\},$$
$$exp(D) = \{\; C \in DHB_P \mid \text{there is a subclause } C' \text{ of } C \text{ in } D \;\},$$

respectively, i.e. $can(D) \subseteq D \subseteq exp(D)$. D is expanded, iff $exp(D) = D$.

(iii) The *set difference* with subsumption and the *set disjunction* of two Herbrand states D_1 and D_2 is given by

$$D_1 \setminus_s D_2 = \{\; C \in D_1 \mid \text{there is no subclause } C' \text{ of } C \text{ in } D_2 \;\},$$
$$D_1 \vee D_2 = \{\; C_1 \vee C_2 \mid C_1 \in D_1,\; C_2 \in D_2 \;\}.$$

(iv) The *disjunctive consequence operator* $\mathcal{T}_P^S : 2^{DHB_P} \to 2^{DHB_P}$ of a disjunctive logic program P maps Herbrand states to Herbrand states:

$$\mathcal{T}_P^S(D) \;=\; \{\; C \in DHB_P \mid\; C' \text{ :- } B_1, \ldots, B_n \text{ is in } Ground(P),$$
$$B_i \vee C_i \in D,\; 1 \leq i \leq n,\; C \text{ is the smallest factor of } C' \vee C_1 \vee \ldots \vee C_n\}.$$

A *model state* D of P is an expanded Herbrand state of P such that every minimal Herbrand model of D is a Herbrand model of P.

Example 1. A very frequently used logic program P is the one that specifies the transitive closure of a graph given by an *arc*–relation:

$$P = \;\{\; path(X,Y) \text{ :- } path(X,Z), path(Z,Y);$$
$$path(X,Y) \text{ :- } arc(X,Y) \;\}.$$

Although P is definite, when applied to a graph given by indefinite information for *arc*, it is necessary to use the more general disjunctive consequence operator $\mathcal{T}_P^S$ instead of the definite one for deriving all the disjunctive facts implied by P. From the Herbrand state $D_0 \;=\; \{\; arc(a,b) \vee arc(a,c), arc(b,d), arc(c,d) \;\}$, the Herbrand states $D_{i+1} = \mathcal{T}_P^S(D_i)$ $(0 \leq i \leq 2)$ are derived:

$$D_1 = \{\; arc(a,b) \vee path(a,c), path(a,b) \vee arc(a,c), path(b,d), path(c,d) \;\},$$
$$D_2 = \{\; path(a,b) \vee path(a,c), path(a,d) \vee arc(a,c), path(a,d) \vee arc(a,b) \;\},$$
$$D_3 = \{\; path(a,d) \vee path(a,c), path(a,d) \vee path(a,b) \;\}.$$

From the Herbrand state $D_3' \;=\; D_3 \cup \{\; path(b,d), path(c,d) \;\}$ we get $\mathcal{T}_P^S(D_3') \;=$ $\{\; path(a,d) \;\}$. Thus, $\mathcal{T}_P^S$ can also derive definite facts from disjunctive facts.

For every *definite* logic program P the minimal Herbrand model $\mathcal{M}_P$ of P is equal to the following sets, cf. [15]: the logical consequences $A \in HB_P$ of P, the least fixpoint $lfp(\mathcal{T}_P)$ of the definite consequence operator $\mathcal{T}_P$, $\mathcal{T}_P \uparrow \omega$, and the logical consequences $A \in HB_P$ of $\mathcal{M}_P$.

In general a *disjunctive* logic program P does not have a unique minimal model, but a set $\mathcal{MM}_P$ of minimal models. For disjunctive logic programs, model states are the structure corresponding to models for logic programs. They can be characterized based on the disjunctive consequence operator: an expanded Herbrand state D of P is a model state of P, iff $\mathcal{T}_P^S(D) \subseteq D$. As for Herbrand models of definite logic programs, every intersection of model states again is a model state. Thus, there is a unique *minimal model state* $\mathcal{MS}_P$: the intersection $\cap_{i \in J} MS_i$ of the set $\{MS_i\}_{i \in J}$ of all model states of P. $\mathcal{MS}_P$ is expanded, since model states are expanded, and it is equivalent to the following Herbrand states, cf. [16]: the logical consequences $C \in DHB_P$ of P, the expanded least fixpoint $exp(lfp(\mathcal{T}_P^S))$, the expansion $exp(\mathcal{T}_P^S \uparrow \omega)$ of the ω–power $\mathcal{T}_P^S \uparrow \omega$, and the set of all clauses $C \in DHB_P$ implied by all minimal models $M \in \mathcal{MM}_P$ of P.

For a definite logic program P, the minimal Herbrand model $\mathcal{M}_P$ is equal to the canonization $can(\mathcal{MS}_P)$ of the minimal model state.

3 The Disjunctive Δ–Iteration Technique

Semi–naive or Δ–iteration was originally defined for definite logic programs. For extending it to the fixpoint computation of a disjunctive logic program P, we need the Δ–version of the disjunctive consequence operator $\mathcal{T}_P^S$.

Definition 1. The *disjunctive consequence operator* $\mathcal{T}_P^S : 2^{DHB_P} \times 2^{DHB_P} \rightarrow 2^{DHB_P}$ of a disjunctive logic program P is applied to Herbrand states $D, \Delta D$:

$$\begin{aligned}\mathcal{T}_P^S(D, \Delta D) \;=\; \{\; & C \in DHB_P \mid C' \text{ :- } B_1, \ldots, B_n \text{ is in } Ground(P),\ (\ n = 0 \ \ or \\ & (\ B_i \vee C_i \in D \cup \Delta D,\ 1 \leq i \leq n, \text{ and some } B_i \vee C_i \text{ is in } \Delta D\ )\ ), \\ & C \text{ is the smallest factor of } C' \vee C_1 \vee \ldots \vee C_n\ \}.\end{aligned}$$

The algorithm Δ–ITERATION of Fig. 1 formulates the disjunctive Δ–iteration based on the Δ–version of $\mathcal{T}_P^S$. When applied to the empty Herbrand state $D = \emptyset$, it yields the least fixpoint of $\mathcal{T}_P^S$. The basic operations of Δ–ITERATION are the following. From a computational point of view, $\mathcal{T}_P^S$ derives a multiset D_1 of disjunctive facts, which has to be pruned by removing duplicates. The time complexity of *pruning* is $n \log n$, where $n = |D_1|$ is the number of facts in D_1. *Δ–computation* w.r.t. subsumption of the set $D_2 \setminus_s D$ of all disjunctive facts in D_2 which are not subsumed by disjunctive facts in D takes time $n_1 \cdot n_2$, where $n_1 = |D|$, $n_2 = |D_2|$. *Canonization* is the computation of all disjunctive facts in a Herbrand state D_3 which are not properly subsumed by other – i.e. shorter – disjunctive facts in the same state. It takes time n^2, where $n = |D_3|$.

Lemma 2. *Given a disjunctive logic program P and Herbrand states $D, \Delta D$ of P:*

$$can(\ prune(\mathcal{T}_P^S(D, \Delta D)) \setminus_s D\ ) \;=\; can(\ prune(\mathcal{T}_P^S(D, \Delta D))\ ) \setminus_s D.$$

Algorithm Δ–ITERATION (P, D)

{ initialization }
$\Delta D := D;$

{ iteration }
while ($|\Delta D| > 0$)
    $D_1 := \mathcal{T}_P^S(D, \Delta D);$
    $D_2 := prune(D_1);$
    $D_3 := D_2 \setminus_s D;$
    $\Delta D := can(D_3);$
    $D := can(D \cup \Delta D);$
end

Fig. 1. Disjunctive Δ–ITERATION

Based on these considerations, we derive an efficient version of the disjunctive Δ–iteration by suitably arranging the basic operations, cf. [20]. Experiments with Δ–ITERATION, cf. Table 1, revealed that the comparatively cheap pruning operation reduces the multiset D_1 of derived facts to about 30% of its size. Then the applied Δ–computation yields a further reduction to about 1 % of the size of D_1 in time $O(|D_2| \cdot |D|)$, where normally D is much smaller than D_2. Finally, the most expensive operation *can* is delayed as far as possible. It is applied to the smallest Herbrand state D_3 and reduces it to about 0.5% of the size of D_1.

iteration	1	2	3	4	
$\mathcal{T}_P^S$	0.26	2.69	28.71	11.22	42.0%
prune	0.01	0.53	11.46	4.13	12.6%
$\setminus_s$	0.05	1.21	19.68	11.33	32.0%
can	0.27	2.45	1.13	0.01	3.5%
can	0.55	5.17	4.19	0.02	9.9%

iteration	1	2	3	4	
$\mathcal{T}_P^S$	100	2.209	33.406	12.667	100%
prune	50	891	8.612	5.235	31%
$\setminus_s$	44	208	151	0	0.9%
can	44	132	47	0	0.5%
can	0	3	30	0	14%

Table 1. Time in seconds and number of tuples after the different operators in tree–based Δ–ITERATION

Algorithm Δ–ITERATION can be efficiently supported by a data structure for disjunctive facts, the so–called *clause trees*, cf. [20]. Another data structure, the so–called *model tree*, was proposed by [16], [10] for representing the set of all minimal models of a Herbrand state instead of the clauses of the Herbrand state. Both types of trees are used for a fast hyperresolution of further disjunctive facts by $\mathcal{T}_P^S$ and for purposes of subsumption elimination. In this general setting, a clause tree is a set tree for a Herbrand state S and a model tree is a set tree for a set S of models.

Definition 3 (Set Tree). Given a set $S = \{S_1, \ldots, S_n\}$ of some finite sets S_i $(1 \leq i \leq n,\ n \in \mathbb{N}_0)$. A *set tree* $\mathcal{T} = (V, E)$ with a node labeling $l : V \rightarrow 2^{\mathcal{A}_S}$, where $\mathcal{A}_S = \cup_{i=1}^{n} S_i$, *represents* S, iff

(i) each node $v \in V$ represents a set $set(v)$ as follows: $set(v)$ is the union of the sets $l(v')$ which label the nodes v' on the path from the root to v,

(ii) the sets $set(v)$ of the leaves $v \in V$ are in S, and every set $S_i \in S$ is represented by exactly one leaf of the set tree.

Heuristics for constructing a set tree $\mathcal{T} = (V, E)$ representing a set S in a good way can be based on the graph–theoretic notion of minimal Steiner trees, cf. [20]. The goal is to minimize the sum $weight(\mathcal{T}) = \sum_{e \in E} length(e)$ of the edge lengths, where for an edge $e = (v_1, v_2) \in E$: $length(e) = \mid set(v_2) \setminus set(v_1) \mid$.

Tree–resolution with clause trees has been analyzed for the frequently occurring definite transitive closure rules with disjunctive facts, cf. Table 1. For general disjunctive logic programs involving disjunctive rules the method is applicable as well, and yields even more efficient evaluation.

4 The Extended Generalized Closed World Assumption

The generalized disjunctive well–founded semantics, cf. [16], assigns to a disjunctive normal logic program P a state pair $S =< T, F >$ consisting of a disjunctive Herbrand state T and a conjunctive Herbrand state F, i.e. a set of ground conjunctions. The clauses in T (F) are considered *true* (*false*).

Definition 4. Given a disjunctive normal logic program P.

(i) The *conjunctive Herbrand base* CHB_P of P is

$$CHB_P \ = \ \{\ A_1 \wedge \ldots \wedge A_k \mid k \geq 1,\ A_i \in HB_P,\ 1 \leq i \leq k\ \}.$$

A *conjunctive Herbrand state* F of P is a subset $F \subseteq CHB_P$. The *canonical form* and the *expansion* of F are given by

$$can(F) = \{\ C \in F \mid \text{there is no proper subconjunction } C' \text{ of } C \text{ in } F\ \},$$
$$exp(F) = \{\ C \in CHB_P \mid \text{there is a subconjunction } C' \text{ of } C \text{ in } F\ \}.$$

respectively, i.e. $can(F) \subseteq F \subseteq exp(F)$. F is expanded, iff $exp(F) = F$.

(ii) A *state pair* $S =< T, F >$ of P consists of a disjunctive Herbrand state T and a conjunctive Herbrand state F.

For a disjunctive logic program P, we already know that the minimal model state $\mathcal{MS}_P$ of P is the set of ground disjunctions which are *true* in all (minimal) Herbrand models of P. The *dual approach* is followed by the extended generalized closed world assumption: $\mathcal{EGCWA}_P$ will contain the set of ground conjunctions which are *false* in all minimal Herbrand models of P.

Definition 5. The *extended generalized closed world assumption* of a disjunctive logic program P is given by

$$\mathcal{EGCWA}_P \ = \ \{ \ C \in CHB_P \mid C \text{ is } \textit{false} \text{ in all minimal Herbrand models of } P \ \}.$$

Example 2. The disjunctive logic program $P \ = \ \{ \ c \ :\!\!- \ a, \quad a \vee b \ \}$ derives the minimal model state $\mathcal{MS}_P = \{a \vee b, b \vee c, a \vee b \vee c\}$ and has the set $\mathcal{MM}_P = \{\{a, c\}, \{b\}\}$ of minimal Herbrand models. Thus, the set of all ground conjunctions which are *false* in all $M \in \mathcal{MM}_P$ is given by $\mathcal{EGCWA}_P = \{a \wedge b, b \wedge c, a \wedge b \wedge c\}$.

The extended generalized closed world assumption is *maximal consistent* with the minimal model state $\mathcal{MS}_P$: i.e. it contains exactly the ground conjunctions $A_1 \wedge \ldots \wedge A_k$ which can be assumed to be *false* without making it possible to derive new positive ground disjuncts from $\mathcal{MS}_P$ and the disjunctions $\neg A_1 \vee \ldots \vee \neg A_k$.

The extended generalized closed world assumption will be applied to a certain disjunctive logic program $Dis(P, S)$ obtained from a disjunctive normal logic program P by a transformation w.r.t. a given state pair S. $Dis(P, S)$ is a ground disjunctive logic program which is produced if in the ground instances of the rules of P all negated body atoms are moved to the rule head – yielding positive atoms – and atoms which are *true* or *false* w.r.t. the state pair S are removed from the resulting rules. Finally, rules with *true* heads or *false* bodies, respectively, are removed. Note, that $Dis(P, S)$ is a possibly infinite disjunctive logic program.

Definition 6. Given a disjunctive normal logic program P and a state pair $S = <T, F>$.

(i) The *disjunctive transformation* $Dis(P) \ = \ \{ \ C' \vee B_{m+1} \vee \ldots \vee B_{m+n} \ :\!\!- \ B_1, \ldots, B_m \mid C' \ :\!\!- \ B_1, \ldots, B_m, \neg B_{m+1}, \ldots, \neg B_{m+n} \in P \ \}$ of P is a disjunctive logic program.

(ii) The *disjunctive transformation* $Dis(P, S)$ of P w.r.t. S is a ground disjunctive logic program derived by transforming $Ground(Dis(P))$ as follows. Given a disjunctive rule $r = A_1 \vee \ldots \vee A_k \ :\!\!- \ B_1, \ldots, B_n \in Ground(Dis(P))$:

1. Remove the rule r if $A_{i_1} \vee \ldots \vee A_{i_l} \in T$ for some subdisjunction of its head atoms – i.e. the head is *true* in S – or if $A_1, \ldots, A_k \in F$ – i.e. the head is *false* in S – or if $B_{i_1} \wedge \ldots \wedge B_{i_l} \in F$ for some subconjunction of its body atoms – i.e. the body is *false* in S.
2. Reduce the rule r by removing *true* atoms $B_i \in T$ form the body of r and *false* atoms $A_i \in F$ from the head of r.

(iii) The $\mathcal{S}^{\mathcal{E}}$*-derived state pair* is $\mathcal{S}^{\mathcal{E}}(S) = S \cup < \mathcal{MS}_{Dis(P,S)}, \mathcal{EGCWA}_{Dis(P,S) \cup T} >$, where $S_1 \cup S_2 \ = <T_1 \cup T_2, F_1 \cup F_2>$ for two state pairs $S_i = <T_i, F_i>$.

Remember that the extended generalized closed world assumption $\mathcal{EGCWA}_P$ of a disjunctive logic program P is maximal consistent with the minimal model state $\mathcal{MS}_P$. Applied to the disjunctive logic program $Dis(P,S) \cup T$, this means that no new positive ground disjunctions can be derived from the state pair $< \mathcal{MS}_{Dis(P,S)\cup T}, \mathcal{EGCWA}_{Dis(P,S)\cup T} >$ and thus from the weaker state pair $\mathcal{S}^{\mathcal{E}}(S)$.

Example 3. For the disjunctive normal logic program P_1 and the state pair $S = <\{a \vee b\}, \{e, f(a)\}>$ we get the disjunctive transformation $Dis(P_1)$:

$$P_1 = \{\ c \text{ :- } a, \neg e;\ \ a \vee b \vee e;\ \ e \vee f(X) \text{ :- } c, d(X);\ \ a \vee c \text{ :- } e, \neg d(a)\ \},$$
$$Dis(P_1) = \{\ c \vee e \text{ :- } a;\ \ a \vee b \vee e;\ \ e \vee f(X) \text{ :- } c, d(X);\ \ a \vee c \vee d(a) \text{ :- } e\ \}.$$

The disjunctive transformation of P_1 w.r.t. S is $Dis(P_1, S) = \{\ c \text{ :- } a\ \}$. Thus, $Dis(P_1, S) \cup T$ is the disjunctive logic program P from Example 2 and the $\mathcal{S}^{\mathcal{E}}$–derived state pair is given by $\mathcal{S}^{\mathcal{E}}(S) = <\emptyset, \{a \wedge b, b \wedge c, a \wedge b \wedge c\}>$.

5 Generalized Δ–Iteration Techniques

Computing the extended generalized closed world assumption of a disjunctive logic program can be speeded up by tree–based Δ–iteration techniques applied to certain *support–for–negation sets* for the conjunctive facts. These sets can be derived efficiently by an iteration from the minimal model state $\mathcal{MS}_P$, if $can(\mathcal{MS}_P)$ is finite.

During the various $\mathcal{S}^{\mathcal{E}}$–derivation steps in the computation of the generalized disjunctive well–founded semantics of a disjunctive normal logic program, there are repeated computations of the $\mathcal{EGCWA}$ and thus of support–for–negation sets w.r.t. monotonically growing model states. Instead of computing the support–for–negation sets of an enlarged minimal model state from the scratch, we use update formulas for deriving them from the support–for–negation sets w.r.t. the previous minimal model state.

The ground *disjunctive transformation* $Dis(P,S)$ of a disjunctive normal logic program P usually is infinite. Therefore, instead of $Dis(P,S)$ we use the finite, non–ground disjunctive transformation $Dis(P)$ with a special associated consequence operator $\mathcal{T}^S_{P,S}$ defined in terms of $Dis(P)$ and the state pair S. $\mathcal{T}^S_{P,S}$ can be evaluated based on the same Δ–iteration techniques as ordinary disjunctive consequence operators.

5.1 Efficient Computation of Support–for–Negation Sets

The extended generalized closed world assumption $\mathcal{EGCWA}_P$ of a disjunctive logic program P is derived based on an iterative computation of support–for–negation sets w.r.t. the minimal model state $\mathcal{MS}_P$.

Definition 7. Given a disjunctive logic program P. The *support–for–negation set* of a conjunction $A_1 \wedge \ldots \wedge A_k \in CHB_P$ w.r.t. $D \subseteq DHB_P$ is

$$SN(D, A_1 \wedge \ldots \wedge A_k) = \{\ C_1 \vee \ldots \vee C_k \mid A_i \vee C_i \in D \text{ for } 1 \leq i \leq k\ \}.$$

Example 4. For the disjunctive logic program P of Example 2 with the minimal model state $\mathcal{MS}_P = \{a \vee b, b \vee c, a \vee b \vee c\}$ the support–for–negation sets are

$SN(\mathcal{MS}_P, a) = \{b, b \vee c\},\ SN(\mathcal{MS}_P, c) = \{b, a \vee b\},\ SN(\mathcal{MS}_P, b) = \{a, c, a \vee c\},$
$SN(\mathcal{MS}_P, a \wedge b) = SN(\mathcal{MS}_P, b \wedge c) = SN(\mathcal{MS}_P, a \wedge b \wedge c)$
$= \{a \vee b, b \vee c, a \vee b \vee c\},\ SN(\mathcal{MS}_P, a \wedge c) = \{b, a \vee b, b \vee c, a \vee b \vee c\},$

Theorem 8 (Characterization). *For a disjunctive logic program P it holds:*

$$\begin{aligned}\mathcal{EGCWA}_P = \{\ & A_1 \wedge \ldots \wedge A_k \mid (\mathcal{MS}_P \models A_i \vee C_i \text{ for } 1 \leq i \leq k) \\ & \text{implies } (\mathcal{MS}_P \models C_1 \vee \ldots \vee C_k)\ \} \\ = \{\ & C \in CHB_P \mid SN(\mathcal{MS}_P, C) \setminus_s \mathcal{MS}_P = \emptyset\ \}.\end{aligned}$$

We will present a technique for computing $\mathcal{EGCWA}_P$ based on the support–for–negation sets $SN(\mathcal{MS}_P, C)$. In [16] these sets are denoted by $SN(C)$ without explicitly noting the context $\mathcal{MS}_P$. The computation can be expressed more easily with the notion of SN^r–sets, canonized subsets of the SN–sets where all disjunctions, which are subsumed by the Herbrand state D, are removed.

Definition 9. Given a disjunctive normal logic program P and $D \subseteq DHB_P$. The *reduced support–for–negation set* of $C \in CHB_P$ w.r.t. D is

$$SN^r(D, C) \ = \ can(\ SN(D, C) \setminus_s D\ ).$$

The following evaluation technique deals with SN^r–sets, which are subsets of the corresponding SN–sets and can be handled much more efficiently. The theorem of [16] relating the SN–sets to the $\mathcal{EGCWA}$ translates to SN^r–sets:

Theorem 10. *For a disjunctive logic program P it holds:*

$$\mathcal{EGCWA}_P = \{\ C \in CHB_P \mid SN^r(\mathcal{MS}_P, C) = \emptyset\ \}.$$

Example 5. The disjunctive logic program P of example 2 derives the reduced support–for–negation sets

$SN^r(\mathcal{MS}_P, a) = \{b\},\ SN^r(\mathcal{MS}_P, c) = \{b\},\ SN^r(\mathcal{MS}_P, b) = \{a, c\},$
$SN^r(\mathcal{MS}_P, a \wedge b) = SN^r(\mathcal{MS}_P, b \wedge c) = SN^r(\mathcal{MS}_P, a \wedge b \wedge c) = \emptyset,$
$SN^r(\mathcal{MS}_P, a \wedge c) = \{b\},$

and thus $\mathcal{EGCWA}_P = \{a \wedge b, b \wedge c, a \wedge b \wedge c\}$.

The support–for–negation sets $SN(D, A)$ for atoms A are easily derived from the Herbrand state D. The support–for–negation sets of longer conjunctions are derived from shorter ones based on an algebraic formula given by [16]. A similar formula also holds for the reduced support–for–negation sets.

Theorem 11 (Recursion Formulas). *Given $D \subseteq DHB_P$, $C_1, C_2 \in CHB_P$:*

$$\begin{aligned}SN(D, C_1 \wedge C_2) \ &= \ SN(D, C_1) \vee SN(D, C_2). \\ SN^r(D, C_1 \wedge C_2) \ &= \ can(\ (SN^r(D, C_1) \vee SN^r(D, C_2)) \setminus_s D\ ).\end{aligned}$$

A *bottom–up computation* of all the reduced support–for–negation sets using Δ–iteration techniques will be proposed. The algorithm Δ–ITERATION–SN given in Fig. 2 computes the generalized closed world assumption $\mathcal{GCWA}_P = \mathcal{EGCWA}_P \cap HB_P$ and the canonization $\mathcal{EGCWA} = can(\mathcal{EGCWA}_P \setminus HB_P)$ for a disjunctive logic program P. It is based on the minimal model state $\mathcal{MS}_P$, which has to be computed before. $\mathcal{GCWA}_P$ can be determined easily as the set $HB_P \setminus \cup_{C \in \mathcal{MS}_P} C$ of atoms in HB_P which do not occur in any clause of $\mathcal{MS}_P$.

Algorithm Δ–ITERATION–SN($\mathcal{MS}_P, \mathcal{GCWA}_P, \mathcal{EGCWA}$)

{ initialization }
$\mathcal{MS}$ $:= can(\ \mathcal{MS}_P\ )$;
$\mathcal{SN}$ $:= \{\ (A, SN(\mathcal{MS}, A)) \mid A \in HB_P$ and $SN(\mathcal{MS}, A) \neq \emptyset\ \}$;
$\mathcal{GCWA}_P := \{\ A \in HB_P \mid SN(\mathcal{MS}, A) = \emptyset\ \}$;

{ recursion }
SNR$(\mathcal{SN}, \mathcal{MS}, \mathcal{SN}^r, \mathcal{EGCWA})$.

Fig. 2. Δ–ITERATION–SN

Procedure SNR, cf. Fig. 3, uses a *divide–and–conquer* approach for computing the support collection $\mathcal{SN}^r$ of all pairs $(C, SN^r(C))$ for conjunctions $C \in CHB_P$, where $SN^r(C) \neq \emptyset$ (non–trivial pairs). Each intermediate support collection $\mathcal{SN}^r$ contains the pairs for all conjunctions C over a certain $I \subseteq HB_P$. The initial set $I = \{A \in HB_P | SN^r(A) \neq \emptyset\}$ is recursively split into two arbitrary subsets $I = I_1 \cup I_2$. Correspondingly, $\mathcal{SN} = \{(A, SN^r(A)) | A \in I\}$ is split into subsets $\mathcal{SN}_i = \{(A_i, SN^r(A_i)) | A_i \in I_i\}$, $i = 1, 2$, from which the support collections $\mathcal{SN}_i^r = \{(C_i, SN^r(C_i)) | C_i \subseteq I_i, SN^r(C_i) \neq \emptyset\}$ over I_i are computed recursively.

Procedure SNR$(\mathcal{SN}, \mathcal{MS}, \mathcal{SN}^r, \mathcal{E})$;

if $|\mathcal{SN}| > 1$
then { divide and conquer }
split $\mathcal{SN}$ into $\mathcal{SN}_1$ and $\mathcal{SN}_2$;
SNR$(\mathcal{SN}_1, \mathcal{MS}, \mathcal{SN}_1^r, \mathcal{E}_1)$; SNR$(\mathcal{SN}_2, \mathcal{MS}, \mathcal{SN}_2^r, \mathcal{E}_2)$;
MERGE$(\mathcal{MS}, \mathcal{SN}_1^r, \mathcal{SN}_2^r, \mathcal{SN}^r, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E})$;

else { base case }
$\mathcal{SN}$ contains exactly one pair $(A, SN(\mathcal{MS}, A))$;
$\mathcal{SN}^r := \{\ (A, SN(\mathcal{MS}, A))\ \}$; $\mathcal{E} := \emptyset$;

Fig. 3. Divide–and–conquer method

The *merging*, cf. Fig. 4, of two derived support collections $\mathcal{SN}_i^r$ over I_i constructs the set disjunctions $SN^r(\mathcal{MS}, C_1) \vee SN^r(\mathcal{MS}, C_2)$ for all support pairs $(C_i, SN^r(\mathcal{MS}, C_i)) \in \mathcal{SN}_i^r$, $i = 1, 2$, removes duplicates (*prune*), and yields

$$SN^r(\mathcal{MS}, C_1 \wedge C_2) \;=\; can(\; prune(\; SN^r(\mathcal{MS}, C_1) \vee SN^r(\mathcal{MS}, C_2)\;)\setminus_s \mathcal{MS}\;).$$

The derived support collection $\mathcal{SN}^r$ contains the non–trivial pairs $(C, SN^r(C))$ for conjunctions C over $I = I_1 \cup I_2$. The minimal conjunctions $C_1 \wedge C_2$, where the derived set $SN^r(\mathcal{MS}, C_1 \wedge C_2)$ is empty, are returned.

Procedure MERGE$(\mathcal{MS}, \mathcal{SN}_1^r, \mathcal{SN}_2^r, \mathcal{SN}^r, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E})$;

{ set disjunctions }
for each $(C_1, SN^r(\mathcal{MS}, C_1)) \in \mathcal{SN}_1^r$ **and** $(C_2, SN^r(\mathcal{MS}, C_2)) \in \mathcal{SN}_2^r$
$S_1 := SN^r(\mathcal{MS}, C_1) \vee SN^r(\mathcal{MS}, C_2)$;
$S_2 := prune(S_1)$; $S_3 := S_2 \setminus_s \mathcal{MS}$;
$SN^r(\mathcal{MS}, C_1 \wedge C_2) := can(S_3)$;

{ result assignments}
$\mathcal{SN}^r := \{\; (C_1 \wedge C_2, SN^r(\mathcal{MS}, C_1 \wedge C_2)) \mid SN^r(\mathcal{MS}, C_1 \wedge C_2) \neq \emptyset \;\}$;
$\mathcal{E}_3 := can(\; \{\; C_1 \wedge C_2 \mid SN^r(\mathcal{MS}, C_1 \wedge C_2) = \emptyset \;\}\;)$;
$\mathcal{E} := \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3$

Fig. 4. Merging of SN^r–sets and $\mathcal{EGCWA}$–sets

An *optimized implementation* of MERGE uses the following techniques for the derivation of $SN^r(\mathcal{MS}, C_1 \wedge C_2)$ from $SN^r(\mathcal{MS}, C_i)$, $i = 1, 2$. Firstly, the *reduction* of $D \subseteq DHB_P$ w.r.t. $C \in CHB_P$ removes all C' which contain an atom of C: $reduce(D, C) \;=\; \{\; C' \in D \mid \text{no atom of } C \text{ occurs in } C' \;\}$. Applied to $S_1 = reduce(SN^r(\mathcal{MS}, C_1), C_2)$ and $S_2 = reduce(SN^r(\mathcal{MS}, C_2), C_1)$ we have

$$SN^r(\mathcal{MS}, C_1 \wedge C_2) \;=\; can(\; prune(\; S_1 \vee S_2\;)\setminus_s \mathcal{MS}\;).$$

Secondly, from $can(S_1 \vee S_2) = can(\; ((S_1 \setminus_s S) \vee (S_2 \setminus_s S)) \cup S\;)$, where $S = can(exp(S_1) \cap exp(S_2))$, and $can((S_1 \vee S_2) \setminus_s \mathcal{MS}) = can(S_1 \vee S_2) \setminus_s \mathcal{MS}$ we get

$$SN^r(\mathcal{MS}, C_1 \wedge C_2) \;=\; can(\; prune(\; ((S_1 \setminus_s S) \vee (S_2 \setminus_s S)) \cup S\;)\setminus_s \mathcal{MS}\;).$$

Experiments with Δ–ITERATION–SN, e.g. for the transitive closure program with minimal model states of about 150 disjunctive facts, cf. Table 2, have revealed, that the reduction operation (r) significantly reduces the average number of disjunctive clauses in the (initial) reduced support–for–negation sets and thus reduces the evaluation time. The canonization operation (c) also significantly reduces the average size of the reduced support–for–negation sets, but surprisingly increases the overall computation time. Thus, the version (r,–) using only the reduce operation is faster than the one (r,c) with reduce and can, which in

	reduce	$\vee$	prune	$\setminus_s$	can	total
(r,–)	1.55	4.50	2.54	12.10	––	25.09
(r,c)	1.20	2.87	1.92	7.42	11.84	29.05
(–,–)	––	7.49	4.58	16.78	––	33.03

	initial	reduce	$\vee$	prune	$\setminus_s$	can
(r,–)	57.10	33.53	33.11	24.53	16.39	16.39
(r,c)	29.00	16.99	16.68	15.32	9.69	4.62
(–,–)	57.10	57.10	56.66	42.37	16.39	16.39

Table 2. Time in seconds for Δ–ITERATION–SN and average number of clauses in reduced support–for–negation sets

turn is faster than the one (–,–) with none of them. Note that there is always some additional time (included in total) spend with scheduling operations.

We are also investigating the computation of $can(\mathcal{EGCWA}_P)$ from the set $\mathcal{MM}_P$ of all minimal models based on Definition 5. For the tested examples, however, this derivation technique for $\mathcal{EGCWA}_P$ was slower than Δ–ITERATION–SN, even when optimized with model trees.

5.2 Update Formulas for Support–for–Negation Sets

During the various derivation steps in the computation of the generalized disjunctive well–founded semantics of a disjunctive normal logic program a sequence of monotonically growing state pairs $S =< T, F >$ is generated by alternating applications of the introduced operator $\mathcal{S}^{\mathcal{E}}$ and another operator $\mathcal{S}^{\mathcal{D}}$ given in [16]. It can be shown, that for the disjunctive programs $P' = Dis(P, S) \cup T$ the minimal model states $\mathcal{MS}_{P'}$ are monotonically increasing. Thus, the $\mathcal{S}^{\mathcal{E}}$–steps require the repeated computation of the $\mathcal{EGCWA}_{P'}$ w.r.t. monotonically growing model states $\mathcal{MS}_{P'}$. Instead of computing the support–for–negation sets of an enlarged minimal model state from the scratch, we incrementally use the support–for–negation sets of the smaller minimal model state.

The following theorem gives update formulas for deriving the reduced support–for–negation sets of a the union of two disjunctive Herbrand states from the separate reduced support–for–negation sets of the two disjunctive Herbrand states.

Theorem 12. *Given a disjunctive logic program P, Herbrand states D and ΔD, an atom $A \in HB_P$ and conjunctions $C_1, C_2 \in CHB_P$. For $\Delta SN^r(D, \Delta D, C) = SN^r(D \cup \Delta D, C) \setminus_s SN^r(D, C)$ we get:*

$$\begin{aligned} SN^r(D \cup \Delta D, A) &= can(\ (SN^r(D, A) \cup SN^r(\Delta D, A)) \setminus_s (D \cup \Delta D)\ ), \\ SN^r(D \cup \Delta D, C_1 \wedge C_2) &= (\ can(\ \Delta SN^r(D, \Delta D, C_1) \vee \Delta SN^r(D, \Delta D, C_2)\ ) \\ &\quad \cup\ SN^r(D, C_1 \wedge C_2)\ )\ \setminus_s (\ D \cup \Delta D\ ), \end{aligned}$$

5.3 The Disjunctive Transformation $Dis(P, S)$

The ground disjunctive transformation $Dis(P, S)$ of a disjunctive normal logic program P w.r.t. a state pair $S =< T, F >$ usually is a large – in the presence of variables and function symbols infinite – logic program, which is computationally intractable. Fortunately, we can replace the disjunctive consequence operator $\mathcal{T}^S_{Dis(P,S)}$ by the following, equivalent consequence operator $\mathcal{T}^S_{P,S}$ defined in terms of S and the smaller, non–ground disjunctive transformation $Dis(P)$.

Definition 13. Given a disjunctive normal logic program P and a state pair $S =< T, F >$. The *consequence operator* $\mathcal{T}^S_{P,S} : 2^{DHB_P} \rightarrow 2^{DHB_P}$ of P w.r.t. S maps Herbrand states to Herbrand states:

$$\begin{aligned}\mathcal{T}^S_{P,S}(D) \;=\; \{\; & C \in DHB_P \mid A_1 \vee \ldots \vee A_k \;\text{:-}\; B_1, \ldots, B_n \in Ground(Dis(P)),\\ & \{\; B_1 \vee C_1, \ldots, B_n \vee C_n \;\} \;\subseteq\; can(D \cup (T \cap HB_P)),\\ & A_1 \vee \ldots \vee A_k \notin exp(T),\;\; B_1 \wedge \ldots \wedge B_n \notin exp(F),\\ & \{\; A_1, \ldots, A_k \;\} \setminus F = \{\; A_{i_1}, \ldots, A_{i_l} \;\} \neq \emptyset,\\ & C \text{ is the smallest factor of } A_{i_1} \vee \ldots \vee A_{i_l} \vee C_1 \vee \ldots \vee C_n \;\}.\end{aligned}$$

Theorem 14. *Given a disjunctive normal logic program P and a state pair $S =< T, F >$. Then $\mathcal{T}^S_{Dis(P,S)} \uparrow \omega = \mathcal{T}^S_{P,S} \uparrow \omega$.*

6 Summary and Outlook

We have developed a system called DisLog for the efficient evaluation of disjunctive normal logic programs under the generalized disjunctive well–founded semantics. It is implemented as a meta–interpreter in Prolog.

One basic concept for computing the generalized disjunctive well–founded semantics of a disjunctive, normal logic programs P is the repeated application of the operator $\mathcal{S}^{\mathcal{E}}$ computing the minimal model state and the extended generalized closed world assumption w.r.t. certain disjunctive transformations of P. The basic operators for computing minimal model states are the disjunctive consequence operator $\mathcal{T}^S_P$, the Δ–operator $\setminus_s$ with subsumption and the canonization operator *can* for Herbrand states. Tree–based versions of these operators are applied in a Δ–iteration for computing the minimal model state $\mathcal{MS}_P$ and the extended generalized closed world assumption $\mathcal{EGCWA}_P$ of a disjunctive logic program P. The support–for–negation sets for conjunctive facts can be derived efficiently by an iteration from $\mathcal{MS}_P$.

The operator $\mathcal{S}^{\mathcal{D}}$ for state pairs given in [16], which is based on a positive and a negative consequence operator $\mathcal{T}^D_S$ and $\mathcal{F}^D_S$ is implemented efficiently in DisLog as well. The least fixpoint of the composite operator $\mathcal{S}^{\mathcal{ED}} = \mathcal{S}^{\mathcal{E}} \circ \mathcal{S}^{\mathcal{D}}$ for state pairs is the generalized disjunctive well–founded state pair.

Acknowledgment

The author would like to thank his colleague Helmut Thöne for the many useful comments he has provided.

References

[1] *A. Abdelmoty, M. Williams, N. Paton:* Deduction and Deductive Databases for Geographic Data Handling, Proc. Intl. Symposium on Advances in Spatial Databases 1993 (SSD'93), pp. 443-464.

[2] *C. Baral, J. Lobo, J. Minker:* Generalized Well–Founded Semantics for Logic Programs, Proc. Intl. Conf. on Automated Deduction 1989, pp. 102-116.

[3] *F. Bancilhon, R. Ramakrishnan:* An Amateur's Introduction to Recursive Query Processing Strategies, Proc. ACM SIGMOD 1986, pp. 16-52.

[4] *S. Ceri, G. Gottlob, L. Tanca:* Logic Programming and Databases, Springer, 1990.

[5] *W. Chen, D.S. Warren:* A Goal–Oriented Approach to Computing Well–Founded Semantics, Proc. Joint Intl. Conf. and Symp. on Logic Prog. 1992, pp. 589-603.

[6] *W. Chen, D.S. Warren:* Query Evaluation under the Well–Founded Semantics, Proc. ACM PODS 1993.

[7] *S.K. Das:* Deductive Databases and Logic Programming, Addison–Wesley, 1992.

[8] *J. Dix:* Classifying Semantics of Disjunctive Logic Programs, Proc. Joint. Intl. Conf. and Symp. on Logic Programming 1992, pp. 798-812.

[9] *U. Fuhrbach:* Computing Answers for Disjunctive Logic Programs, Proc. ILPS Workshop on Disjunctive Logic Programs 1991.

[10] *J.A. Fernández, J. Minker:* Theory and Algorithms for Disjunctive Deductive Databases, Programmirovanie J. 1993, Academy of Sciences of Russia.

[11] *M. Fitting:* The Family of Stable Models, Journal of Logic Programming, vol. 17(3,4), 1993, pp. 197-225.

[12] *M. Gelfond, V. Lifschitz:* The Stable Model Semantics for Logic Programming, Proc. Intl. Conf. and Symp. on Logic Programming 1988, pp. 1070-1080.

[13] *U. Güntzer, W. Kießling, R. Bayer:* On the Evaluation of Recursion in (Deductive) Database Systems by Efficient Differential Fixpoint Iteration, Proc. Intl. Conf. on Data Engineering 1987, pp. 120-129.

[14] *G. Köstler, W. Kießling, H. Thöne, U. Güntzer:* The Differential Fixpoint Operator with Subsumption, Proc. Intl. Conf. DOOD 1993, pp. 35-48.

[15] *J.W. Lloyd:* Foundations of Logic Programming, Springer, second edition, 1987.

[16] *J. Lobo, J. Minker, A. Rajasekar:* Foundations of Disjunctive Logic Programming, MIT Press, 1992.

[17] *J. Pearl:* Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, 1988.

[18] *T.C. Przymusinski*: Stationary Semantics for Disjunctive Logic Programs and Deductive Databases, Proc. North–American Conf. of Logic Prog. 1990, pp. 42-59.

[19] *D. Seipel, H. Argenton*: Evaluation Techniques for Disjunctive Logic Programs, Proc. Symposium on Operations Research 1993.

[20] *D. Seipel*: Tree–Based Fixpoint Iteration for Disjunctive Logic Programs, Proc. Workshop on Logic Prog. with Incomplete Inf., Intl. Symp. on Logic Prog. 1993.

[21] *D. Seipel, H. Thöne*: An Application of Disjunctive Logic Programming with Incomplete Information, Proc. Intl. Conf. on Expert Syst. for Development 1994.

[22] *J.D. Ullman*: Principles of Database and Knowledge-Base Systems, Volume I,II, Computer Science Press, 1988/89.

[23] *A. Van Gelder, K.A. Ross, J.S. Schlipf*: The Well–Founded Semantics for General Logic Programs, JACM, vol. 38(3), 1991, pp. 620-650.

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