Knowledge Representation and Reasoning in NMR and Extended LP

Jürgen Dix

University of Koblenz-Landau
Dept. of Computer Science

and

Technical University of Vienna
Institute for Knowledge-Based Systems

VII. Complexity, Expressibility and Implementation

- Complexity
- Expressibility
- Implementing WFS
- Implementing STABLE
- Implementing D-WFS

1 Complexity

- **First-order case:** Highly undecidable due to *loop-checking*

$$p(x) \leftarrow p(f(x))$$

- **Propositional case:**
  - normal programs: first level of the pol. hierarchy
  - disjunctive programs: second level of the pol. hierarchy

	Complexity	
	1. ord. prog. *(with functions)*	*prop. prog.* *(no variables)*
$\mathbf{M_P}$ *(P is Horn)*	A: Σ_1^0-compl. not A: Π_1^0-compl.	**linear** in $\lvert P\rvert$
COMP	Π_1^1-compl. over $\mathbb{N}$	**co-NP**-compl.
$\mathbf{COMP_3}$	Π_1^1-compl. over $\mathbb{N}$	**linear** in $\lvert P\rvert$
STABLE	Π_1^1-compl. over $\mathbb{N}$	**co-NP**-compl.
WFS	Π_1^1-compl. over $\mathbb{N}$	**linear** in $\#At \times \lvert P\rvert$

Table 2: Complexity of Non-Disjunctive Semantics

	Complexity	
	1. ord. prog. *(with functions)*	*prop. prog.* *(no variables)*
GCWA *(P is positive)*	A: $\Sigma^{\mathbf{0}}_{\mathbf{1}}$-compl. *not* A: $\Pi^{\mathbf{0}}_{\mathbf{2}}$-compl.	A: **co-NP**-compl. *not* A: $\Pi^{\mathbf{P}}_{\mathbf{2}}$-compl.
D-WFS	$\Pi^{\mathbf{1}}_{\mathbf{1}}$-compl. over $\mathbb{N}$	$\Pi^{\mathbf{P}}_{\mathbf{2}}$-compl.
D-STABLE	$\Pi^{\mathbf{1}}_{\mathbf{1}}$-compl. over $\mathbb{N}$	$\Pi^{\mathbf{P}}_{\mathbf{2}}$-compl.

Table 3: Complexity of Disjunctive Semantics

Systems handling first-order programs must be necessarily incomplete. They can be only complete under additional assumptions on the underlying class of programs like

1. allowed programs without function symbols, also called *safe DATALOG*-programs, or
2. programs, where never a *negative literal with free variables* is selected (to resolve against), so called *non-floundering*-programs.

- WFS is at most quadratic. Thus computing WFS first is cheap and helps in computing STABLE. In fact the following holds:

$$a \in WFS(P):\ STABLE(P) = STABLE(P \cup \{a\})$$

- D-WFS is of the same complexity as D-STABLE. But still the analogue holds

$$a \in D\text{-}WFS(P):\ D\text{-}STABLE(P) = D\text{-}STABLE(P \cup \{a\})$$

2 Expressibility

Distinguish between

- **EDB-relations** (relations that do not appear in the head of a program)
- **IDB-relations** (which are contained in some heads).

More generally: fix a countable domain *DOM* and consider

- a relational schema $\mathbf{R}$, which stands for finitely many relation symbols $R_1, \ldots, R_n$.
- An instance $\mathcal{D}$ of $\mathbf{R}$ is then a finite relational structure

$$\mathcal{D} := \langle U^{\mathcal{D}}, R_1^{\mathcal{D}}, \ldots R_n^{\mathcal{D}} \rangle$$

  where $U^{\mathcal{D}} \subseteq DOM$ is finite, called the *universe*.
- We denote by $\mathbf{Inst}(\mathbf{R})$ the set of all instances over $\mathbf{R}$.

Definition 2.1 (Query Mapping)
Let two disjoint relational schemas $\mathbf{R}, \mathbf{S}$ *be given. A query* q *is a partial recursive mapping*

$$q : \mathbf{Inst}(\mathbf{R}) \rightarrow \mathbf{Inst}(\mathbf{S}),$$

i.e. q *maps instances of* $\mathbf{R}$ *to instances of* $\mathbf{S}$*. We assume* $U^{\mathcal{D}} = U^{q(\mathcal{D})}$ *for all* $\mathcal{D} \in \mathbf{Inst}(\mathbf{R})$.

(union) q_{union}**:** which maps two relations R_1, R_2 to their union $R_1 \cup R_2$,

(negation) q_{neg} **:** which maps a relation R to its complement $\overline{R}$,

(trans. closure) q_{tc}**:** which maps a relation R to its transitive closure R_{tc},

(even) q_{even}**:** which maps an instance $\mathcal{D}$ of $\mathbf{R}$ to "true" iff the universe $U^{\mathcal{D}}$ has an even number of elements.

Every query q has associated a *recognition problem*: given an instance $\mathcal{D}_{\mathbf{R}}$ of $\mathbf{R}$ and a tuple $\underline{t}$ (matching the format of a relation S of $q(\mathbf{R})$, i.e. the arities correspond and $\underline{t}$ is a sequence of elements of the universe of $U^{\mathcal{D}}$), decide whether

$$\underline{t} \in S^{q(\mathcal{D})}.$$

In our examples: $\mathbf{S} = \{S\}$ (single relation output).

Recognition problems of above queries can be decided in log-SPACE, and therefore also in PTIME.

Queries in classical logic

Let **R** be a schema and a finite set of constants be given. Both sets determine the underlying language $\mathcal{L}$. We define:

C_1: formulae built up from "$\exists, \wedge, =$" in the language $\mathcal{L}$.

C_2: formulae built up from "$\exists, \wedge, \vee, =$" in the language $\mathcal{L}$.

C_3: formulae built up from "$\exists, \wedge, \neg, =$" in the language $\mathcal{L}$. Obviously, $\forall$ and $\vee$ can be built from the other constructs as usual.

Every class C_i of formulae determines in a natural way a class of queries. For a fixed formula φ with free variables $x_1, \ldots, x_n$ and every instance $\mathcal{D}$ of **R**, the set

$$\{(t_1, \ldots, t_n) \mid \mathcal{D} \models \varphi[t_1, \ldots, t_n]\},$$

constitutes an instance of a schema $\mathbf{S} = \{S\}$, namely the interpretation of S. We define the universe of the instance to be the same as the universe of the original instance (the range of the terms t_i).

How can we now define queries with programs under certain semantics?

1. For a given program P we can view any instance $\mathcal{D}$ of the (finite) EDB-relations as an *input* argument and then compute the (finite) IDB-relations (the *output*) under the respective sceptical semantics.

2. Any program P determines a query under a semantics SEM as follows: an instance $\mathcal{D}$ over **R** determines the semantics $SEM(P+\mathcal{D})$, i.e. all IDB relations are determined and thus constitute an instance over the schema consisting of all IDB relations.

We are in particular interested to express all relations of some complexity class. E.g. the relations *inductively definable* over $\mathcal{D}$ (denoted by **IND**), which is a strict subclass of the relations that are *polynomial* over $\mathcal{D}$. These are the fixpoint queries.

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

Table 4: Expressibility of Non-Disjunctive Semantics

	Expressibility *1. ord. prog.* *(no functions)*
GCWA *(P is positive)*	$\subset \Pi_2^{\mathbf{P}}$
WGCWA *(P is positive)*	$\subset \Pi_2^{\mathbf{P}}$
PERFECT *(P is stratified)*	$= \Pi_2^{\mathbf{P}}$
WPERFECT *(P is stratified)*	$= \Pi_2^{\mathbf{P}}$
D-WFS	$= \Pi_2^{\mathbf{P}}$
DSTABLE	$= \Pi_2^{\mathbf{P}}$

Table 5: Expressibility of Disjunctive Semantics

3 Implementing WFS

Why not using our confluent calculus? There are two problems with GPPE:

- Fairness,
- Complexity.

Two cheap instances of GPPE:

Definition 3.1 (Success and Failure)
The success-transformation can be applied to a program P whenever there is a rule $A \leftarrow$ in P. Success then removes a positive occurrence of A in the body of another rule.

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

Theorem 3.2 (Linear Complexity)
The calculus of rules consisting of success, failure and reduction (see Definition 2.1) is also confluent. In addition, it is strongly terminating and the normal form $nf(P)$ *can be computed in linear time. The semantics according to the normal form coincides with the least fixpoint of the Fitting operator* Φ_P *and thus coincides with* $comp_3$*:*

$$comp_3(P) = \{A:\ A \leftarrow\ \ \in nf(P)\} \cup \{not\,A:\ A \textit{ is in no head of nf}(P)\}$$

Definition 3.3 (Loop Detection)
The Loop Detection transformation can be applied to a program P whenever there is a set $S_{\text{unfounded}}$ of ground atoms such that

- *for each rule in $A \leftarrow \mathcal{B} \in P$, if $A \in S_{\text{unfounded}}$ then $\mathcal{B} \cap S_{\text{unfounded}} \neq \emptyset$.*

Loop Detection then removes all rules of P which contain in their body a positive occurrence of an atom in $S_{\text{unfounded}}$.

Although checking for such a set $S_{\text{unfounded}}$ can be done in linear time, we may need various applications of it and therefore may reach an overall quadratic behaviour. A worst case estimate is the number of atoms $\#At$ occurring in P.

1. We first compute the *strongly connected components* of the program (as given by the dependency graph). As is well-known, this can be done in linear time (see eg. (Tarjan 1972; Berman, Schlipf, and Franco 1995)).
2. We start with working on the lowest component and propagate the values of atoms to the higher components.

3. In every connected component we iterate the following

   (a) apply *reduction* as well as *success* and *failure* until a fixpoint is reached. As mentioned in Theorem 3.2, this fixpoint corresponds exactly to the fixpoint of the Fitting operator Φ_P.

   (b) apply *loop-detection*, which amounts to computing the maximal unfounded set $S_{\text{unfounded}}$ (Van Gelder, Ross, and Schlipf 1988).

   If there are either *only* positive dependencies or *no* predicate defined in that component depends positively on itself, then no iteration of the two steps above are needed.

Theorem 3.4 (Implementation of WFS)
The above described algorithm is provably better than the alternating fixpoint procedure of Van Gelder (1989), which already works in $\#At \times |P|$, where $\#At$ is the number of atoms and $|P|$ the size of the program.

Moreover, the algorithm is linear for the class of all programs, where in each strongly connected component the following holds: either all the head-atoms defined in that component depend only positively from each other *or* no predicate defined in that component depends positively on itself *(also called* positive-acyclic*). Note that these two conditions can change from one connected component to the other.*

Example 3.5 (Algorithm for WFS using SCC's)
We consider the following program, which contains both negative and positive dependencies.

$$
\begin{array}{lllllllll}
p_1 & \leftarrow & q_1 & p_2 & \leftarrow & not\ q_2 & q_3 & \leftarrow & not\ w_3 \\
q_1 & \leftarrow & r_1 & q_2 & \leftarrow & not\ s_2 & w_3 & \leftarrow & not\ z_3, x_3 \\
r_1 & \leftarrow & q_1, z_3 & s_2 & \leftarrow & t_2 & z_3 & \leftarrow & \\
r_1 & \leftarrow & p_2, not\ q_3 & t_2 & \leftarrow & not\ q_3, t_2 & x_3 & \leftarrow & not\ x_3
\end{array}
$$

4 Implementing STABLE

Stable models are strongly related to extensions of default logic:

$$A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n$$

corresponds to a default rule

$$\frac{B_1 \wedge \cdots \wedge B_m : \neg C_1, \ldots, \neg C_n}{A}$$

We can thus use our method from Chapter III.

Definition 4.1 (P-full Sets)
For a logic program P, a set $\mathrm{F} \subseteq \{not\ A \mid A \in \mathrm{NA}(P)\}$ *is P-full if and only if*

$$\textit{for all } A \in \mathrm{NA}(P),\ \textit{not } A \in \mathrm{F} \textit{ if and only if } A \notin \mathrm{Cn}_{\mathrm{Mon}(P,\mathrm{F})}(\emptyset)$$

where

$$\begin{aligned}\mathrm{Mon}(P,\mathrm{F}) = \{A \leftarrow B_1,\ldots,B_m \mid\ & A \leftarrow B_1,\ldots,B_m, not\ C_1,\ldots,not\ C_n \in P,\\ & \{not\ C_1,\ldots,not\ C_n\} \subseteq \mathrm{F}\}\end{aligned}$$

and $\mathrm{Cn}_{\mathrm{Mon}(P,\mathrm{F})}(\emptyset)$ *is the deductive closure of the rules* $\mathrm{Mon}(P,\mathrm{F})$.

The set of rules $\mathrm{Mon}(P,\mathrm{F})$ can be seen as a set of definite clauses and their deductive closure is just the unique least model of the clauses. Now

1. for every P-full F, $\mathrm{Cn}_{\mathrm{Mon}(P,\mathrm{F})}(\emptyset)$ is a stable model of P and
2. for every stable model M, $\mathrm{F} = \{\mathit{not}\, A \mid A \in \mathrm{NA}(P) \setminus M\}$ is the unique corresponding full set such that $M = \mathrm{Cn}_{\mathrm{Mon}(P,\mathrm{F})}(\emptyset)$.

expand(P,FS)

We need the bounds to satisfy:

- For the lower bound $\mathrm{LB}(P,\mathrm{FS})$ it holds that every stable model of P agreeing with FS, agrees with $\mathrm{LB}(P,\mathrm{FS})$.
- For the upper bound $\mathrm{UB}(P,\mathrm{FS})$ it holds that for every stable model M of P which agrees with FS, $M \subseteq \mathrm{UB}(P,\mathrm{FS})$.

Given these bounds, **expand**(P,FS) returns the smallest set of literals FS' extending FS and closed under the following rules:

- if $\varphi \in \mathrm{LB}(P,\mathrm{FS}')$, then include φ in FS' and
- if $A \notin \mathrm{UB}(P,\mathrm{FS}')$ for $A \in \mathrm{NA}(P)$, then include *not* A in FS'.

- P/FS potentially applicable rules (when the positive as well as negative body literals are not in conflict with FS).

$$P/\mathrm{FS} = \{\ A \leftarrow B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n \mid \\ \{not\, B_1, \ldots, not\, B_m, C_1, \ldots, C_n\} \cap \mathrm{FS} = \emptyset\}$$

For the upper bound $\mathrm{UB}(P, \mathrm{FS})$ we use the least model of the definite program obtained by removing all negative literals from P/FS.

We define the lower bound $\mathrm{LB}(P, \mathrm{FS})$ as the smallest set FS' including FS and closed under the rules

1. If for $A \leftarrow L_1, \ldots, L_n \in P/\mathrm{FS}'$, $\{L_1, \ldots, L_n\} \subseteq \mathrm{FS}'$, then $A \in \mathrm{FS}'$
2. If an atom A is not a head of any rule in P/FS', then *not* $A \in \mathrm{FS}'$.
3. If $A \in \mathrm{FS}'$ is the head of only one rule $A \leftarrow L_1, \ldots, L_n$ in P/FS', then $\{L_1, \ldots, L_n\} \subseteq \mathrm{FS}'$.
4. If *not* $A \in \mathrm{FS}'$ and A is the head of a rule $A \leftarrow L_1, \ldots, L_n$ in P/FS' with $\{L_1, \ldots, L_{i-i}, L_{i+1}, \ldots, L_n\} \subseteq \mathrm{FS}'$, then $\mathrm{compl}(L_i) \in \mathrm{FS}'$.

conflict(P,FS)

As the conflict test is performed for a set of assumptions computed by **expand** it is sufficient to implement **conflict** as follows:

conflict(P, FS) returns true if and only if there is some atom A such that $\{A, not\ A\} \subseteq \mathrm{FS}$.

choose implements the search heuristics: $\mathbf{choose}(P, \mathrm{FS})$ returns either A or *not* A for some atom $A \in \mathrm{NA}(P)$ not covered by FS.

- *Start from the lowest strongly connected component in the dependency graph of the program.*
- *Maximum Occurrences in rules of Minimum size (Mom's) heuristics*

  Most successful heuristics for the Davis-Putnam type procedure for propositional satisfiability. It can be adjusted to stable model computation and made quite dynamic as follows (used in **smodels** system (Niemelä and Simons 1996)). Given a set of current assumptions FS, select a negative literal from a body of a potentially applicable rule (i.e. included in P/FS) with the *least number of negative literals not in* FS.

Example 4.2 *We illustrate the method with two examples. Consider the following program P.*

$p \leftarrow not\ q, not\ u$		1. $not\ u$		1. $not\ s$	
$q \leftarrow not\ p, not\ u$		$\swarrow\ \searrow$		2. $not\ u$	
$r \leftarrow p$	2. $not\ t$		3. t	3. $not\ t$	
$s \leftarrow not\ q$	4. s		11. $not\ s$	$\swarrow\ \searrow$	
$s \leftarrow not\ r, q$	$\swarrow\ \searrow$		12. q	4. $not\ p$	5. p
$t \leftarrow not\ s$	5. $not\ r$	6. r	13. $not\ p$	6. q	9. r
$s \leftarrow not\ t$	7. $not\ p$	9. p	14. $not\ r$	7. r	10. $not\ q$
	8. q	10. $not\ q$	15. $not\ q$	8. $not\ r$	11. s
			$\times$	$\times$	$\times$

5 Implementing D-WFS

- Normalform of Confluent Calculus is exponential.
- Reduction to iterated minimal model reasoning is possible.
- First-order version can be done by using *constrants*: Extending CLP by *disjunction and negation*: see (Dix and Stolzenburg 1998).

Definition 5.1 (NAF-**Consistency**)
A model I for a disjunctive logic program P (viewed as a set of atoms and formulae not (ϕ)*) is called* NAF-consistent *if the following holds*

$$\textit{If } P \models A_1 \vee \cdots \vee A_n, \textit{ then for some } i \in \{1, \ldots, n\}, \textit{not } (A_i) \notin I.$$

$\models$ denotes standard prop. conseq. relation. P is seen as a set of clauses and the *not* (ϕ) atoms as new atoms. Hence, e.g., $\{p \leftarrow \textit{not } q, p \leftarrow q\} \models p$ does not hold.

For example, $\{p, r, \textit{not } p, \textit{not } q\}$ is not a NAF-consistent model of

$$P = \{p \vee q \leftarrow, r \leftarrow \textit{not } p \wedge \textit{not } q\}$$

but $\{p, r, \textit{not } p\}$ is. However, the latter is not a minimal model of P.

In the following we will be interested in *truth in all* NAF-*consistent minimal models*: $\models_{\mathrm{min(N)}}$:

$P \models_{\mathrm{min(N)}} \neg A$ *if and only if* all NAF-consistent min. models of P satisfy $\neg A$.

Definition 5.2 ($\mathbf{SEM}_{\mathrm{NAF}}$)

Let P be a disjunctive logic program over $\mathcal{L}$. We define a set $\mathrm{Dis}_{\mathrm{min}}(P)$ of pure disjunctions as follows

$$\mathrm{Dis}_{\mathrm{min}}(P) := \quad \mathrm{Dis}^{+}(P) \cup \mathrm{Dis}^{-}(P) \textit{ where}$$

$$\begin{array}{lllllll}
\mathrm{Dis}^{+}(\mathrm{P}) := & \{ & A_1 \vee \ldots \vee A_k & : & P & \models & A_1 \vee \ldots \vee A_k & \} \\
\mathrm{Dis}^{-}(P) := & \{ & \neg A & : & P & \models_{\mathrm{min(N)}} & \neg A & \}
\end{array}$$

Here, the series D_i defined by $D_0 := \emptyset$, $D_{n+1} := \mathrm{Dis}(P/D_n)$ grows monotonically and eventually gets constant. We define $SEM_{\mathrm{NAF}}(P)$ to be the limit of this series.

What does P/D_n mean? Intuitively, the program obtained from *prog* by reducing it using disjunctions in D_n. More precisely, P/Dis is obtained by doing the following reductions on P for all *not* C and $C_1 \vee \ldots \vee C_k$:

- if $\neg C \in \mathrm{Dis}$, then remove all ocurrences of *not* C,
- if $C_1 \vee \ldots \vee C_k \in \mathrm{Dis}$ then remove all rules that contain $\{\mathit{not}\, C_1, \ldots, \mathit{not}\, C_k\}$ in their bodies.

P/Dis is obviously a slight generalization of the Gelfond-Lifschitz transformation. Our P/Dis still contains negative literals in general.

Theorem 5.3 *D-WFS*$(P) = \mathrm{SEM}_{\mathrm{NAF}}(P)$.

1. Implement D-WFS as an iterative reduction on DLP's.

2. Start with the original program P and get a reduced program P^* s.t. every query can be answered from this program with *one call* to $\models_{\mathrm{min(N)}}$.

3. The reduced program is obtained as a fixed point of a reduction operator $\mathrm{R_D}(\cdot)$ mirroring the operator in $\mathrm{SEM_{NAF}}(P)$.

$$\begin{aligned}\mathrm{R_D}(P) \quad = \quad & \{\mathcal{A} \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{C}' \mid \mathcal{A} \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{C} \in P, \\ & P/\mathrm{At}_{\mathcal{L}} \not\models C_1 \vee \cdots \vee C_n \text{ where} \\ & \mathcal{C} = \{C_1, \ldots, C_n\}, \\ & \mathcal{C}' = \{C \in \mathcal{C} \mid P \not\models_{\mathrm{min(N)}} \neg C\}\}\end{aligned}$$

Now let P^* be the limit of the (monotonically decreasing) series of programs $P_0, P_1, \ldots$ where $P_0 = P$ and $P_{i+1} = \mathrm{R}_{\mathrm{D}}(P_i)$. Hence, the reduced program is obtained by

- removing each rule such that $C_1, \ldots, C_n$ are all the negative body literals and $C_1 \vee \cdots \vee C_n$ is a classical consequence of $P/\mathrm{At}_{\mathcal{L}}$ and
- removing all *not* (C) from the bodies of the remaining rules for which $\neg C$ is minimally entailed by the remaining program

until no reduction is possible.

It can be established that $P_i = P/D_i$ for all $i = 0, 1, \ldots$ Thus we have:

Theorem 5.4 $D\text{-}WFS(P) = \mathrm{SEM}_{\mathrm{NAF}}(P) = \mathrm{Dis}_{\min}(P^*)$.

Theorem 5.5 (Polynomial Space Complexity)
The entailment relation $\models_{\min(\mathrm{N})}$*, and therefore also D-WFS, can be implemented in polynomial space.*

References

Berman, K. A., J. S. Schlipf, and J. V. Franco (1995, June). Computing the Well-Founded Semantics Faster. In A. Nerode, W. Marek, and M. Truszczyński (Eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Third International Conference*, LNCS 928, Berlin, pp. 113–126. Springer.

Brass, S. and J. Dix (1995, September). D-WFS: A Confluent calculus and an Equivalent Characterization. Technical Report TR 12/95, University of Koblenz, Department of Computer Science, Rheinau 1.

Brass, S. and J. Dix (1996). Characterizing D-WFS: Confluence and Iterated GCWA. In L. P. J.J. Alferes and E. Orlowska (Eds.), *Logics in Artificial Intelligence (JELIA '96)*, LNCS 1126, pp. 268–283. Springer. (Extended version will appear in the *Journal of Automated Reasoning* in 1998.).

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

Dix, J. and F. Stolzenburg (1998). A Framework to incorporate Nonmonotonic Reasoning into Constraint Logic Programming. *Journal of Logic Program-*

ming 36(1,2,3), 5—37. Special Issue on *Constraint Logic Programming*, Guest Editors: Kim Marriott and Peter Stuckey.

Eiter, T. and G. Gottlob (1993). Propositional Circumscription and Extended Closed World Reasoning are Π_2^P-complete. *Theoretical Computer Science 144*(2), 231–245, Addendum: vol. 118, p. 315, 1993.

Eiter, T. and G. Gottlob (1995). On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence 15*(3/4), 289–323.

Gottlob, G. (1992). Complexity results for nonmonotonic logics. *Journal of Logic and Computation 2*(3), 397–425.

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

Lifschitz, V. (1989, January). Benchmark Problems for Formal NMR, Version 2.00. In Reinfrank, de Kleer, Ginsberg, and Sandewall (Eds.), *Non-Monotonic Reasoning*, LNAI 346, Berlin, pp. 202–219. Springer.

Lloyd, J. W. (1987). *Foundations of Logic Programming*. Berlin: Springer. 2nd edition.

McCarthy, J. (1986). Applications of Circumscription to formalizing Common-Sense Reasoning. *Artificial Intelligence 28*, 89–116.

Niemelä, I. and P. Simons (1996, September). Efficient Implementation of the Well-founded and Stable Model Semantics. In M. Maher (Ed.), *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, pp. 289–303. The MIT Press.

Reiter, R. (1980). A Logic for Default-Reasoning. *Artificial Intelligence 13*, 81–132.

Schlipf, J. (1986, June). How uncomputable is general circumscription? In *Proceedings of the Symposium on Logic in Computer Science*, Cambridge, USA, pp. 92–95. IEEE Computer Society Press.

Stillman, J. (1992, July). The complexity of propositional default logics. In *Proceedings of the 10th National Conference on Artificial Intelligence*, San Jose, California, USA, pp. 794–800. The MIT Press.

Tarjan, R. (1972). Depth first search and linear graph algorithms. *SIAM Journal on Computing 1*, 146–160.

Van Gelder, A. (1989, March). The Alternating Fixpoint of Logic Programs with Negation. In *Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, pp. 1–10. ACM Press.

Van Gelder, A., K. A. Ross, and J. S. Schlipf (1988). Unfounded Sets and well-founded Semantics for general logic Programs. In *Proceedings 7th Symposion on Principles of Database Systems*, pp. 221–230.