Minimal number of permutations sufficient to compute all extensions a finite default theory

Paweł Cholewiński and Mirosław Truszczyński
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046

pawel|mirek@cs.engr.uky.edu

Abstract

In this paper we analyze an algorithm for generating extensions of a default theory. This algorithm considers all permutations (orderings) of defaults. For each permutation, it constructs a tentative extension incrementally, in each step firing the first applicable default, where the applicability of a default is defined with respect to the part of a tentative extension constructed so far. When no more defaults can be fired, an *a posteriori* consistency check is performed to test whether all defaults that were fired remain applicable with respect to the final theory. If so, this theory is returned as an extension. Otherwise, the next ordering is tried. Straightforward worst case analysis implies that this algorithm may have to inspect all $n!$ permutations in order to be complete (here n is the number of defaults in an input default theory). In this paper we show that this number can be significantly reduced. Namely, we exhibit a set of $\binom{n}{\lfloor n/2 \rfloor} \approx 0.8 \times 2^n/\sqrt{n}$ permutations whose inspection guarantees that the algorithm will detect all extensions. In addition, we present a simple algorithm to generate all these permutations.

1 Introduction

Default logic introduced by Reiter [Rei80] is one of the most widely studied non-monotonic formalisms. It was proposed as a knowledge representation mechanism, as it often offers concise descriptions of various knowledge domains and commonsense reasoning situations. However, in order to be a full-blown knowledge representation tool, algorithmic methods are needed to process default theories and compute their extensions. Several algorithms to perform these tasks were proposed recently

[MT93a, ALS94, Nie95, MNR95]. All these algorithms exploit known characterizations of extensions of default theories in terms of sets of generating defaults, sets of justifications, or ordering of defaults. They search through the space of the appropriate objects and identify those that indeed generate extensions. In the worst case, the size of the search space of each such algorithm is at least exponential. However, some pruning is usually possible and results in substantial speedups — several experimental studies are now underway and preliminary results are being reported [NS95, CMMT95].

In this paper we will analyze one of the basic algorithms to generate extensions. The algorithm works by considering all orderings (permutations) of the set of defaults of a default theory. It was introduced in [MT93a] and further studied in [ALS94]. We will show that by some simple modifications, the size of the search space for this algorithm can be reduced from $n!$ to $\binom{n}{\lfloor n/2 \rfloor} \approx \sqrt{2/\pi} \times 2^n/\sqrt{n} \approx 0.8 \times 2^n/\sqrt{n}$ (throughout the paper, we assume that the set of defaults in an input default theory has cardinality n). We will also show that this value is best possible.

Hence, the resulting algorithm constructs all extensions of a default theory by considering a search space that at the *start* of the algorithm's execution is guaranteed to have size $o(2^n)$. To the best of our knowledge, this is the only algorithm so far with such property. Other algorithms start with the full search space consisting of all subsets of the set of defaults, or of all subsets of the set of justifications. The size of the former of these search spaces is 2^n, and can be even larger for the latter one (although in this latter case, it can sometimes be smaller).

We will start with a brief description of basic algorithms for generating extensions. Only a general outline of the first two of them will be given here. The third of these algorithms, which is the main subject of this paper, will be described in detail.

Select-defaults-and-check

1. Select a set of defaults $S \subseteq D$.

2. Check if S is a set of generating defaults. If so, output the theory generated by S as an extension.

3. Repeat until all subsets of D are considered or pruned.

This is perhaps the most commonly studied algorithm. For its detailed description see [MT93a]. In the version given above, the algorithm inspects all 2^n subsets of D. However, pruning techniques can be incorporated into this method to dynamically restrict the search space during the execution of the program. For example, once a set S of defaults is found to be generating, its proper subsets and supersets need not be considered. This follows from the fact that extensions are incomparable and therefore their generating sets must be incomparable too (see [MT93a] for details). Still, the initial search space has size 2^n.

Select-justifications-and-check

1. Select a set of justifications $J \subseteq j(D)$.

2. Find the set of defaults S whose justifications belong to $j(D)$.

3. Compute the set of consequences E of W that can be derived by means of defaults in S (a default "fires" if its prerequisite has been derived earlier).

4. If all justifications in J are consistent with E and every default not in S has at least one justification not consistent with E, then output E as an extension.

5. Repeat until all subsets of $j(D)$ are considered or pruned.

Theoretical basis for this algorithm can be traced back to the concept of an argumentation framework [Dun93, BTK93]. Recently, Niemelä and Simmons [Nie95, NS95] presented a detailed description, analysis and implementation of the method and performed an experimental study. They showed that the algorithm performs

surprisingly well. This can be attributed mostly to some interesting pruning techniques used in the implementation. However, at the beginning, the algorithm faces the search space of all subsets of the set of all justifications. The size of this search space can be smaller than 2^n. However, it is easy to give examples of default theories with n defaults, for which the set of justifications has at least n elements and, consequently, the set of all subsets of the set of justifications has size at least 2^n.

Finally, we will describe, this time in more detail, the algorithm which we study in this paper. The main advantage of this algorithm is its close relationship to some basic intuitions behind default reasoning. Namely, the idea of a default reasoning is to process defaults according to our current state of knowledge. We start with W as all that is known. We use this current state of our knowledge to find an applicable default and extend the theory constructed so far. We continue this way until no more applicable defaults remain.

This method is incorporated in the algorithm given below. To break ties between defaults that can be chosen at a given stage we use an ordering of defaults and always choose the first applicable one. Clearly, the problem with this approach is that some facts derived later in the process may "block" defaults used earlier. Therefore a final consistency check is needed (step 5, below).

Select-ordering-and-check

Input: A finite default theory (D, W) with $|D| = n$

Output: A list of all extensions of (D, W).

1. Select a permutation $d_1, \ldots, d_n$ of D.
2. Mark all defaults in D *available*.
3. Set $S := W$.
4. Repeat until no longer possible: find the smallest i such that d_i is marked *available* and is applicable with respect to S (that is, the prerequisite of d_i is

a consequence of S and every justification of d_i is consistent with S). Mark d_i *used* and add its consequent to S.

5. If every justification of every used default is consistent with S, output $Cn(S)$ as an extension.

6. Repeat all these steps until all permutations are used.

The following result was proved in [MT93a].

Theorem 1.1 *The algorithm* **Select-ordering-and-check** *correctly finds all extensions of a given default theory.*

Despite its simplicity, **Select-ordering-and-check** algorithm is computationally complex. It requires inspecting all $n!$ permutations of the set of defaults. In such form it cannot be competitive with the two algorithms described before. In [MT93a], it is shown that it suffices to consider only 2^n permutations to guarantee that all extensions of a default theory will be found (in other words, to guarantee the completeness of the method).

In the next section, we will show that there is a set of permutations X_n of size $\binom{n}{\lfloor n/2 \rfloor} \approx 0.8 \times 2^n/\sqrt{n}$ that also guarantees the completeness of the select-ordering-and-check method. We will describe an algorithm to efficiently construct the set X_n and we will show that X_n cannot in general be further reduced without sacrificing the completeness property (although using pruning techniques, for some default theories it may be possible to find all extensions by considering only a fraction of permutations in the set X_n).

2 Results

We start by recalling a property of the algorithm **Select-ordering-and-check** proved in [MT93a]. It involves the notion of the set of *generating defaults*. For the definition

of this concept, as well as of other concepts from default logic the reader is referred to [MT93a].

Theorem 2.1 *Let (D, W) be a finite default theory. Let $d_1, \ldots, d_n$ be a permutation of defaults in D. If for some k, $0 \leq k \leq n$, $\{d_1, \ldots d_k\}$ is the set of generating defaults for an extension S of (D, W), then the execution of the steps (2) - (5) of* **Select-ordering-and-check** *algorithm will produce and output S.* □

This theorem implies a corollary which is the basis for the results of our paper.

Corollary 2.2 *Let (D, W) be a default theory and let $|D| = n$. Let X be any set of permutations such that*

$(*)$ *each subset of D appears as a prefix (in some ordering) in at least one permutation from X.*

Then, the algorithm **Select-ordering-and-check** *restricted to the set of permutations X correctly finds all extensions of (D, W).*

Proof: It follows from Theorem 1.1 that any output of the algorithm **Select-ordering-and-check** (and, hence, also of its modified version) is an extension. To complete the proof we need to show that every extension of (D, W) will be found by the modified **Select-ordering-and-check** algorithm. To this end, let us consider an extension S of (D, W). Let $G \subseteq D$ be the set of generating defaults for S. By condition $(*)$, there is a permutation σ in X such that the defaults in G form a prefix of σ. By Theorem 2.1, S will be the output produced by the **Select-ordering-and-check** algorithm after it processes the permutation σ. □

Corollary 2.2 yields a method to improve the performance of the algorithm **Select-ordering-and-check**. Namely, we need to find (and use in the algorithm) the minimum size set of permutations A satisfying condition $(*)$.

We start by deriving a lower bound for such a set of permutations. Let us consider the set of propositional variables $\{p_1, \ldots, p_n\}$. Define defaults d_i by

$$d_i = \frac{: Mp_i}{p_i}.$$

For a subset P of $\{p_1, \ldots, p_n\}$ define $\varphi_P = \bigvee_{p \in P} \neg p$. Let $k = \lfloor n/2 \rfloor + 1$. It is easy to check that the default theory $(\{d_i : 1 \leq i \leq n\}, \{\varphi_P : |P| = k\})$ has exactly $\binom{n}{\lfloor n/2 \rfloor}$ extensions, each corresponding to a subset of the set of defaults of size $\lfloor n/2 \rfloor$. Hence, in order to guarantee the completeness of the algorithm **Select-ordering-and-check**, the set of permutations it uses must have at least $\binom{n}{\lfloor n/2 \rfloor}$ elements.

Notice that the default theory described above has its objective part, W, of size exponential in n — the cardinality of the set of defaults. This raises an interesting question: is there a default theory of size (measured as the total number of occurrences of propositional letters) polynomial in n and having $\binom{n}{\lfloor n/2 \rfloor}$ extensions? We have only partial results on this subject. Namely, assume that $n = rt$ for some integers r and t. Partition the set of propositional atoms $\{p_1, \ldots, p_n\}$ into r disjoint subsets $Q_1, \ldots, Q_r$, each of size t. For each of these disjoint sets Q_i of atoms apply the construction described above. Denote by (D_{Q_i}, W_{Q_i}) the resulting default theory. Clearly, each (D_{Q_i}, W_{Q_i}) has $\binom{t}{\lfloor t/2 \rfloor}$ extensions and the size of W_{Q_i} is $\binom{t}{\lfloor t/2 \rfloor + 1}(\lfloor t/2 \rfloor + 1)$.

Next, define $D = \bigcup_{i=1}^{r} D_{Q_i}$ and $W = \bigcup_{i=1}^{r} W_{Q_i}$. Since the sets Q_i are disjoint, it follows that the number of extensions of (D, W) is

$$\binom{t}{\lfloor t/2 \rfloor}^r$$

and the size of W is

$$\binom{t}{\lfloor t/2 \rfloor + 1}(\lfloor t/2 \rfloor + 1)r.$$

Notice now that if t is selected so that $\binom{t}{\lfloor t/2 \rfloor} \approx n$, then the size of W is $O(n^2)$ and the number of extensions of (D, W) is $2^{c_n n}$, where $c_n < 1$ and $c_n \longrightarrow 1$. Hence, our construction yields a default theory with n defaults, the total size polynomial in n, and with the number of extensions that is close to the desired bound of $\binom{n}{\lfloor n/2 \rfloor} \approx 0.8 \times 2^n/\sqrt{n}$.

In the remaining part of the paper we will construct the set of $\binom{n}{\lfloor n/2 \rfloor}$ permutations satisfying the condition $(*)$. We will study this issue in abstract, combinatorial terms. A collection X of permutation of $\{1, \ldots, n\}$ is called *complete* if each subset of $\{1, \ldots, n\}$ appears as a prefix (in some ordering) in at least one permutations from X. Our goal is to construct a complete set of permutations of $\{1, \ldots, n\}$ of size $\binom{n}{\lfloor n/2 \rfloor}$.

We will first provide a simple proof that for every n the minimum size of a complete set of permutations is indeed $\binom{n}{\lfloor n/2 \rfloor}$. Although our proof is inductive and, hence, implies a method to construct a minimum complete set, this method cannot be directly incorporated into **Select-ordering-and-check** algorithm. In the last part of the paper, we will adapt a technique from [Knu73] to describe an alternative method, and show that it can be used to improve the **Select-ordering-and-check** algorithm.

We start with a simple technical lemma – a corollary to the celebrated Hall's theorem [Hal35], see also [Bol78].

Lemma 2.3 *Let $G = (V \cup U, E)$ be a bipartite graph with vertex classes V and U in which all the vertices in V have the same degree, and all the vertices in U have the same degree. If $|U| \geq |V|$, then the maximum matching in G covers all vertices in V.*

Proof: Let us denote $|V| = n_V$, $|U| = n_U$, and let k_V, k_U be the degrees of vertices in V and U respectively. Since $n_V \leq n_U$ and $n_V k_V = n_U k_U$, we have $k_V \geq k_U$.

By Hall's theorem, G has a matching which covers V if and only if for any $J \subseteq V$, $|\Gamma(J)| \geq |J|$ ($\Gamma(J)$ denotes the set of vertices in U, adjacent to at least one vertex of J).

Suppose that there is a subset $J \subseteq V$ for which $|\Gamma(J)| < |J|$. There are at least $k_V|J|$ edges incident to $\Gamma(J)$. Since $|\Gamma(J)| < |J|$, there must be at least one vertex in $\Gamma(J)$ which has degree at least $k_V + 1$. But all vertices in U have degree $k_U \leq k_V$, so a contradiction follows. □

We will use now Lemma 2.3 to show that the minimum size of a complete set of permutations of n elements is $\binom{n}{\lfloor n/2 \rfloor}$.

Proposition 2.4 *Let $N = \{1, 2, \ldots, n\}$ and X_n be a minimum cardinality of a complete set of permutations of N. Then*

$$|X_n| = \binom{n}{\lfloor n/2 \rfloor}.$$

Proof: Since $N = \{1, 2, \ldots, n\}$ contains $\binom{n}{\lfloor n/2 \rfloor}$ different subsets of cardinality $\lfloor n/2 \rfloor$ and no two different subsets of the same cardinality can be prefixes of the same permutation, $|X_n| \geq \binom{n}{\lfloor n/2 \rfloor}$.

We have to show that, in fact, a complete set of permutations of size $\binom{n}{\lfloor n/2 \rfloor}$ exists. In the proof we will show how to construct such a set of permutations. Let A be an array with $\binom{n}{\lfloor n/2 \rfloor}$ rows and n columns. We will fill the entries in A with integers from $\{1, 2, \ldots, n\}$ so that each row of A will contain a permutation of N and, for every subset Y of $\{1, 2, \ldots, n\}$, there will be a row $r(Y)$ in A in which elements of Y appear in the first $|Y|$ positions.

First, consider all subsets $Y_1, Y_2, \ldots, Y_l$ of N of cardinality equal to $\lceil n/2 \rceil$. Clearly, $l = \binom{n}{\lceil n/2 \rceil} = \binom{n}{\lfloor n/2 \rfloor}$. Insert the elements of Y_i, in any order, in the first $\lceil n/2 \rceil$ positions of row i.

The partially filled matrix A has the property that every subset of N of cardinality $\lceil n/2 \rceil$ appears in the first $\lceil n/2 \rceil$ positions of some row of A. Consider an integer j, $1 \leq j \leq \lceil n/2 \rceil - 1$. Assume that the elements of A are arranged so that every subset of N of cardinality k, where $j + 1 \leq k \leq \lceil n/2 \rceil$, appears in the first k positions in some row of A. We will show how to rearrange the elements of A so that the same holds for every subset of N of cardinality k, where $j \leq k \leq \lceil n/2 \rceil$.

Let $A_1, A_2, \ldots, A_l$, $l = \binom{n}{j+1}$, be all the subsets of N of cardinality $j + 1$, and let $B_1, B_2, \ldots, B_m$, $m = \binom{n}{j}$, be all the subsets of N of cardinality j. Consider the bipartite graph $G = (U \cup V, E)$, where $U = \{A_1, A_2, \ldots, A_l\}$, $V = \{B_1, B_2, \ldots, B_m\}$ and $\{A_i, B_j\} \in E$ if and only if $B_j \subset A_i$. Clearly all vertices in U have degree $j + 1$, and all vertices in V have degree $n - j$. Since $n - j \geq j + 1$, Lemma 2.3 implies that G contains a matching, say M, covering V.

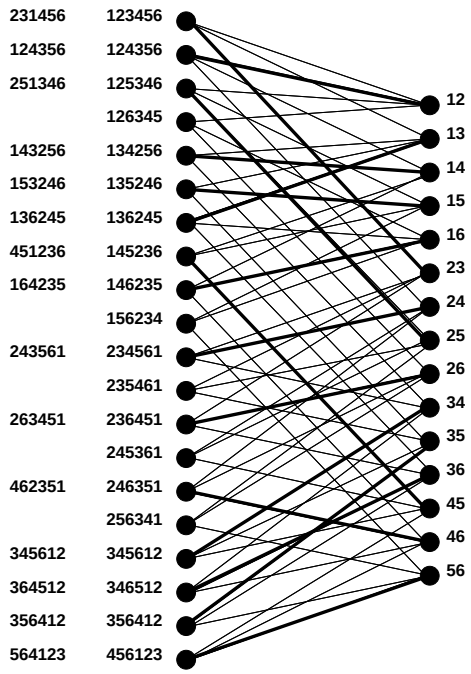

Figure 1: Finding permutations covering all subsets of size 2 for $n = 6$.

For each edge $\{A_i, B_j\} \in M$ find the row in A which contains elements of A_i in its first $j+1$ positions. Next, permute these elements so that the elements of B_j occur in the first j positions. Since M is a matching, no row of A will be permuted twice and the whole operation is well-defined. Observe also that permuting rows of A in this fashion preserves the property that every subset of N of cardinality k, $j+1 \leq k \leq \lceil n/2 \rceil$, appears as a prefix in some row of A. This operation only forces the same property to hold for sets of cardinality j. In Figure 1 we illustrate this construction for $n = 6$ and $j = 2$. Vertices on the right correspond to all subsets of size 2. Vertices on the left correspond to permutations covering all subsets of size 3. The edges of the matching (shown in bold) indicate the permutations which need to be modified in order to guarantee that all subsets of size 2 are covered, as well. The results of the modifications are shown in the leftmost column.

By applying the above procedure for $j = \lceil n/2 \rceil - 1, \lceil n/2 \rceil - 2, \ldots, 1$, we obtain a rearrangement of the matrix A such that every subset of N of cardinality k, $1 \leq k \leq \lceil n/2 \rceil$, appears as a prefix in some row of A.

Notice that the last $\lfloor n/2 \rfloor$ positions in each row are not filled in. For each row, fill in these remaining positions with the elements of N which do not appear in this row yet. Clearly, every subset of N of cardinality $\lfloor n/2 \rfloor$ appears in some row of A as its suffix. Proceeding as before, we can rearrange the last $\lfloor n/2 \rfloor$ positions in each row so that every subset of N of cardinality k, $1 \leq k \leq \lfloor n/2 \rfloor$, appears as a suffix in some row of A. This, however, simply means that each subset of N of cardinality k, $\lceil n/2 \rceil < k \leq n$ is a prefix in some row of A.

Hence, after all these steps, A contains in its rows $\binom{n}{\lfloor n/2 \rfloor}$ permutations of N and every subset of N appears as a prefix in some row of A. □

The proof of Proposition 2.4 implies a method to generate a minimum size complete set of permutations. The straightforward implementation of this method is based on Hopcroft-Karp method for finding matchings. It leads to a complicated algorithm which runs in $\Theta(2^n\sqrt{n})$ space and requires $\Theta(2^{\frac{3n}{2}}n^{\frac{1}{4}})$ time per permutation. Since it constructs all permutations "at once", if added to **Select-ordering-and-check** algorithm it would result in impractical space requirements.

A much better approach is not to compute all the permutations at once but to generate one permutation at a time. This way one can generate all the needed permutations in $O(n)$ space and $O(n)$ time per permutation. A possible way of implementing this task can be based on the following construction described in [Knu73] pp.567-568. We consider a set $\mathcal{P}$ of paths which start in point $(0,0)$ and end in point (n,r), where $r \geq 0$. Each path consists of n segments, with the ith segment joining the point $(i-1,j)$ with $(i,j+1)$ or $(i,j-1)$ (the latter being allowed only if $j \geq 1$). Hence, the path never goes through grid points (i,j) with negative j. There are exactly $\binom{n}{\lfloor n/2 \rfloor}$ such paths and they correspond to all permutations of $\{1,2,\ldots,n\}$ such that every subset of $\{1,2,\ldots,n\}$ appears as prefix of at least one of the permutations.

For each path p from $\mathcal{P}$ we construct a permutation of $\{1,2,\ldots,n\}$ as follows. We use three lists L_1, L_2 and L_3 which are initially empty. For $i = 1,2,\ldots,n$, if the i-th step of p goes up, we put number i into list L_2; if the step goes down, we put i into

list L_1 and move the currently largest element of list L_2 into list L_3. The resulting permutation is equal to the concatenation of the final contents of L_1, L_2 and L_3, each list in increasing order (see [Knu73]).

All the paths from $\mathcal{P}$ can be generated recursively. This leads to an algorithm which searches a binary tree T. This tree T has depth n and $\binom{n}{\lfloor n/2 \rfloor}$ leaves. Each path from $\mathcal{P}$ corresponds to a root-to-leaf path in T. Therefore, the whole algorithm can be implemented to run in $O(n)$ space and will require $O(1)$ time per each generated path from $\mathcal{P}$. Since the algorithm converting each path into a permutation runs in linear time, this approach requires $O(n)$ time per each permutation. This method of generating permutations from X_n can easily be incorporated into the algorithm **Select-ordering-and-check**. Namely, after generating a permutation from X_n, the steps (2) - (5) of the algorithm **Select-ordering-and-check** should be executed. Then, the next permutation from X_n should be generated, the steps (2) - (5) executed, and the whole process should be repeated until all permutations from X_n are considered.

3 Conclusions

In this paper we presented an algorithm that computes all extensions of a default theory after searching through the search space of permutations (orderings) of defaults of size $\approx 0.8 \times 2^n/\sqrt{n}$. This is the first algorithm by date, guaranteeing that all extensions will be found, and considering in the worst case significantly less candidates than 2^n.

Versions of the algorithm **Select-ordering-and-check** and the corresponding analogues of Theorem 1.1 are known for other objects considered as models of belief sets determined by a default theory: rational extensions and constraint extensions [MT93b, MT95], as well. The main result of this paper holds for these other cases, too. That is, the set of permutations constructed in the previous section guarantees the completeness of the select-ordering-and-check method for computing weak, rational

and constraint extensions.

It is also interesting to note that the methods used in this paper can be generalized. Let t be an integer such that $0 \leq t \leq n$. Modifying the arguments used in the paper one can produce a set of permutations of the size $\binom{n}{t}$ such that every subset of $\{1, \ldots, n\}$ of cardinality not greater than t appears as a prefix in at least one of the permutations in the set. This yields an algorithm to compute extensions of a default theory if bounds on the number of generating defaults are known.

References

[ALS94] G. Antoniou, E. Langetepe, and V. Sperschneider. New proofs in default logic theory. *Annals of Mathematics and Artificial Intelligence*, 12:215–230, 1994.

[Bol78] B. Bollobás. *Extremal Graph Theory.* Academic Press, 1978.

[BTK93] A. Bondarenko, F. Toni, and R.A. Kowalski. An assumption-based framework for non-monotonic reasoning. In A. Nerode and L. Pereira, editors, *Logic programming and non-monotonic reasoning. Proceedings of the Second International Workshop*, pages 171–189. MIT Press, 1993.

[CMMT95] P. Cholewiński, W. Marek, A. Mikitiuk, and M. Truszczyński. Default reasoning system — an implementation of default reasoning. *In preparation.*, 1995.

[Dun93] P.M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming (extended abstract). In *Proceedings International Joint Conference on Artificial Intelligence*, pages 852–859, Los Altos, CA, 1993. Morgan Kaufmann. To appear in *Artificial Intelligence.*

[Hal35] P. Hall. On representatives of subsets. *J. of London Math. Soc.*, 10:26–30, 1935.

[Knu73] D. E. Knuth. *The Art of Computer Programming.* Addison-Wesley, 1973.

[MNR95] W. Marek, A. Nerode, and J. B. Remmel. Rule systems, well-orderings and forward chaining. *Submitted for publication.*, 1995.

[MT93a] W. Marek and M. Truszczyński. *Nonmonotonic logics; context-dependent reasoning.* Berlin: Springer-Verlag, 1993.

[MT93b] A. Mikitiuk and M. Truszczyński. Rational default logic and disjunctive logic programming. In A. Nerode and L. Pereira, editors, *Logic programming and non-monotonic reasoning. Proceedings of the Second International Workshop*, pages 283–299. MIT Press, 1993.

[MT95] A. Mikitiuk and M. Truszczyński. Constrained and rational default logics. In *Proceedings of IJCAI-95*. Morgan Kaufmann, 1995.

[Nie95] I. Niemelä. Towards efficient default reasoning. In *Proceedings of IJCAI-95*, pages 312–318. Morgan Kaufmann, 1995.

[NS95] I. Niemelä and P. Simmons. Evaluating an algorithm for default reasoning. In *Proceedings of the IJCAI-95 Workshop on Applications and Implementations of Nonmonotomic Reasonigs Systems*, 1995.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.