DisLoP: A Research Project on Disjunctive Logic Programming

Chandrabose Aravindan, Jürgen Dix and Ilkka Niemelä
Department of Computer Science
University of Koblenz
Rheinau 1, D-56075 Koblenz, Germany
{arvind,dix,ini}@uni-koblenz.de

This paper gives a brief high-level description of what has been done in the **Dis**junctive **Lo**gic **P**rogramming-project (funded by **D**eutsche **F**orschungs-**G**emeinschaft), undertaken by the University of Koblenz since July 1995. We present the main ideas, cite the relevant papers and point to the implemented systems and how to access them. This paper also serves as a brief survey of the current status of disjunctive logic programming by highlighting important developments and providing enough pointers for further reading.

Keywords: Disjunctive Logic Programming, Semantics, Non-Monotonic Reasoning

1. Introduction

The research project on Disjunctive Logic Programming, referred to as DisLoP, was initiated in July 1995 at the University of Koblenz, Germany by Jürgen Dix and Ulrich Furbach. An initial funding for two years has been provided by DFG ("**D**eutsche **F**orschungs-**G**emeinschaft" which translates to "German Science Foundation") under grant `Fu 263/3-1`. After an evaluation of the whole project by DFG, a regular two years extension has been granted in July 1997. Two research scientists Chandrabose Aravindan and Ilkka Niemelä have been working exclusively on this project since July 1995.

The key idea of this project is to develop semantics and procedures for dealing with disjunction and non-monotonic negation in logic programming. On the procedural side, instead of starting from scratch, the aim is to exploit the techniques that have been developed for yet another project on *Automated Reasoning* (that was started 3 years ago in Koblenz by Ulrich Furbach as a part of the German "Schwerpunkt-Programm" *Deduction*). One important consequence of this is that the PROTEIN theorem prover [22] developed during the course of the Deduction project can be used for (positive) disjunctive logic programming with little modifications. DisLoP also aims to apply the developed concepts and systems in real world applications such as information management systems. This, by itself, has been conceived as a separate project, and thus there are three inter-related and co-operating projects at the University of Koblenz.

In this report, we highlight the salient aspects of the DisLoP project and summarize the results. Interested readers are welcome to visit the project home page at <URL:http://www.uni-koblenz.de/ag-ki/DLP/> to get more information on this and related projects. Though there are a lot of related works and projects, in this report we concentrate mainly on the DisLoP project and the research carried out by its members. This, in fact, is reflected in the bibliography and the reader is assumed to be familiar with research works on logic programming and non-monotonic reasoning in general. Nevertheless, to make this paper self-contained and also readable to the non-specialist, we have added some remarks and references about classical notions in this field. For more detailed investigations and overview articles we refer the reader to [1, 27, 59, 60, 41, 17, 45, 16, 37, 50, 51].

2. Normal Programs

Although our aim is to create a disjunctive logic programming system, many problems already occur for negation with respect to non-disjunctive programs. There are two main competing approaches to provide semantics for negation in logic programming, namely the *stable model semantics* (STABLE) and the *well-founded semantics* (WFS). STABLE was introduced in [52, 53] and WFS at about the same time in [80, 81].

Let us shortly explain STABLE. An important notion is the *reduct* of a ground program P with respect to

AI Communications 0 () 0
ISSN 0921-7126 / $8.00 © , IOS Press

a set of ground atoms S. This reduct is a (definite) program obtained from P by deleting (i) each rule that has a negative literal $\neg a$ in its body with $a \in S$ and (ii) all negative literals in the bodies of the remaining rules. Now a set of ground atoms S is a *stable model* of a ground program P if and only if S is the unique minimal model of the reduct of P with respect to S. For example, program P with two clauses $p \leftarrow \neg q, r$ and $q \leftarrow \neg p$ has a stable model $S_1 = \{q\}$, because the reduct of P with respect to S_1 is $q \leftarrow$ and S_1 is its unique minimal model. For instance, $S_2 = \{p, r\}$ is not a stable model of P because the reduct is $p \leftarrow r$ and its unique minimal model is $\{\}$. In fact, S_1 is the unique stable model of P. For a non-ground program P, the stable models are those of the ground instantiation of the program with respect to its Herbrand universe.

The definition of the well-founded semantics is too technical to be presented here. It may suffice to state that every program P is associated with a unique three-valued model, in contrast to the set of stable models that are two-valued. In our example P above, WFS coincides with STABLE. But if we add a clause $s \leftarrow \neg s$ to P, then no stable models exist, whereas the well-founded model is $\{q, \neg p, \neg r\}$, which leaves s undefined.

The relationship between these two semantics have been thoroughly studied by works like [49]. While WFS is *consistent* (model always exists), allows for *goal-directed* computation and has attractive *complexity* (quadratic in the number of atoms), STABLE can become inconsistent (programs may have no stable models), answering queries can not be restricted to the call-graph below that query, and the complexity of STABLE lies one level higher in the polynomial hierarchy (see [61, 54, 62]). Another difference of STABLE and WFS is that for a given program there might be *several* stable models. Each stable model represents a particular consistent view of the world, while the union of any two is inconsistent. This is much like the extensions in default logic [71] and this viewpoint is called *credulous* in general. In contrast, the WFS is much more cautious. WFS can be seen as an approximation to the intersection of all stable models: if an atom is considered true, it is contained in all stable models (dually for the atoms considered to be false). This is called the *sceptical* viewpoint.

Let us note that whether the sceptical or the credulous viewpoint is the more appropriate one depends on the problem at hand. For problems (see Section 5) having multiple solutions each of which has its own right (like colorings of a graph), the credulous viewpoint is more suited. In fact, problems of this kind can be naturally formulated in such a way that *solutions* correspond to *stable models* of the formulation. Whereas for other problems (*are two distinguished nodes always colored the same?*) only the intersection of all solutions are of importance, not the whole set of solutions. This is exactly the sceptical viewpoint and therefore the well-founded semantics (which computes one single model, an approximation of the intersection of all stable models), is more appropriate.

The most sophisticated and general system for computing WFS is the *XSB-system* of David Warren and his colleagues which is based on tabulation-techniques. Its underlying calculus is *SLG-resolution* due to Weidong Chen and David Warren. XSB should not be confused with the *SLG-system* [32, 33], which implements SLG-resolution as a meta-interpreter on top of existing Prolog systems. The SLG-system also implements (a kind of) stable model semantics and is still developed further by W. Chen. In contrast to the SLG-system, XSB implements SLG-resolution on the WAM-level and behaves like an ordinary Prolog system (not like a meta-interpreter) if ordinary Prolog negation and no tabulation is used. In addition, it is of comparable performance (about 4 times slower than the best systems). It is free and is available by anonymous ftp from <URL:ftp://ftp.cs.sunysb.edu/pub/XSB/>. This is the first time that a negation different from negation-as-finite failure (well-founded semantics) has been incorporated into a complete PROLOG-system. We refer the reader to [31, 33, 34, 77] for more detailed information.

SLG-resolution is very procedural in nature because it is based on tabling-techniques. Roughly, given a query and a program it first computes a *residual program* with respect to the query. Under some assumptions (bounded term size property and similar concepts) which imply *non-floundering*[1], this *residual program* is ground and the query can be checked against this program.

Let us emphasize the fact that although XSB handles non-propositional theories, termination is only guaranteed if the computed residual program is finite and ground. Consequently, it is important to consider the propositional case. The approaches we present in this section cover propositional programs.

Many more semantics than considered in this paper have been defined for normal programs. Dix developed

[1]This means that whenever a negative literal is selected, it is grounded, i.e it does not contain free variables.

in [39, 40] methods for classifying these semantics according to their properties. This work was followed by Brass and Dix and we consider it a very nice result that the transformations introduced in [12, 14, 10, 13] for general disjunctive programs can be used to explain SLG-resolution totally declaratively, without hiding effects in low-level data structures. We explain this in Subsection 2.2.

We start with a discussion on an efficient approach to compute STABLE for range-restricted function-free normal programs, developed during the course of our project by I. Niemelä and P. Simons. The method compares very favourably to other proposed methods because of its novel technique for approximating stable models and its over-all low space complexity.

2.1. An Efficient Approach to Compute STABLE Semantics

We have developed a system for computing the stable model and the well-founded model semantics for range-restricted function-free normal programs [70]. It is based on new implementation techniques for general non-monotonic reasoning developed in [63, 64, 69, 65]. The goal has been to devise an implementation of the stable model semantics that can handle realistic size programs (tens of thousands of ground rules) with a potentially large number of stable models. The system can be obtained from <URL:http://saturn.hut.fi/pub/-smodels/>.

Our system includes two modules: an algorithm for implementing the two semantics for ground programs and an algorithm for computing a grounded version of a range-restricted function-free normal program. The latter algorithm does not produce the whole set of ground instances of the program but a subset which is sufficient in the sense that the stable models are preserved. The emphasis has been on developing efficient methodology for computing stable models of ground programs. Our approach is based on bottom-up backtracking search. It works in linear space and employs a powerful method for pruning the search space which is based on approximating stable models. The approximation technique is closely related to the well-founded semantics and provides an efficient algorithm for computing the well-founded model of a ground program. The system has been tested extensively and compared with a state of the art implementation of the stable model semantics, the SLG system of Chen and Warren [32, 33]. In tests involving ground programs our implementation clearly outperforms SLG.

Figure 1 provides an example of the tests performed. The test cases are generated from planar graphs by translating the graphs into ground programs such that the stable models of the programs correspond exactly to the 3-colorings of the graphs. The figure shows the minimum, maximum and average times for ten different runs on a pseudo-randomly shuffled set of rules. All times are in seconds and they represent the time to find one stable model if one exists, or the time to decide that there are no stable models. For more details and further test cases, see [70].

The test results clearly show that our implementation, **smodels**, computes stable models significantly faster and is able to handle substantially larger examples than SLG. The key to our success appears to lie in the new approximation technique for stable models which is closely related to the well-founded semantics. This semantics is also exploited in SLG but in a different way. The SLG system performs goal-directed query-evaluation and the well-founded semantics is employed only in the beginning where the SLG system computes a residual program for a given query using SLG-resolution. The query can then be evaluated with respect to the well-founded semantics against the residual program. The SLG system evaluates the query with respect to the stable model semantics by computing the stable models of the ground residual program with an *assume-and-reduced* algorithm. The better performance of our method for computing stable models of ground programs when compared to the assume-and-reduce algorithm in SLG can be largely contributed to the fact that we use the well-founded type approximation recursively during the whole search for stable models. This keeps the search space substantially smaller in **smodels**.

These results lead to an interesting idea of combining SLG-resolution and our method for computing stable models of ground programs. An implementation of SLG-resolution, like XSB [77], could be used for implementing the well-founded semantics and for computing the residual program for a query. When continuing towards the stable model semantics, our system could be employed starting from the ground residual program. However, there is a difference when using our grounding algorithm or SLG-resolution for computing a grounded version of a program with variables. The ground program computed by our algorithm is typically larger than the residual program given by SLG-resolution. On the other hand, it has exactly the same stable models as the original program whereas the use of the residual program can led to unsound results: the stable models of the residual program are not necessarily stable models of the original program [33].

Planar graphs, 3-colouring							
		SLG			**smodels**		
Vertices	Rules	min	max	ave	min	max	ave
10	73	7.57	34.45	23.81	0.03	0.11	0.04
12	87	2.68	140.01	82.53	0.03	0.04	0.03
14	101	35.09	657.48	390.56	0.03	0.06	0.04
16	115	3.75	2444.90	1421.19	0.04	0.06	0.05
100	703				0.19	0.23	0.21
300	2103				0.63	0.86	0.71
500	3503				1.03	1.35	1.12

Fig. 1. SLG vs. **smodels**

2.2. An Approach to Compute WFS based on Computing Normal Form

There are various ways to compute the well-founded semantics of a program P. Most methods are based one way or the other on van Gelder's alternating fixpoint procedure and therefore directly compute the three-valued well-founded model. Our approach is quite different: it applies certain program-transformations to P and eventually reaches a normal form, referred to as the *residual program* res(P). This normal form res(P) is such that answering queries with respect to it is extremely simple. The transformations we apply are very natural. For example, if a rule $A \leftarrow$ is in P, then we replace every occurrence of A in the body of clauses in P by true. Dually, if an atom A does not occur in the head of any rule in P, we replace every occurrence of $\neg A$ in P by true and remove all remaining rules containing A (because we know that A is false by the closed world assumption). We also remove non-minimal rules and allow unfolding of clauses.

We do not give here a complete description of our transformations, because this whole approach is investigated in more depth in Section 4. The reason is that it applies to arbitrary *disjunctive programs* and restricting these transformations to non-disjunctive programs is immediate. Also the main results carry over ([11, 13]):

- The calculus of program transformations is *confluent*, so any program is associated with a unique normal form, called the *residuum*. The order in which the transformations are applied does not matter: we always arrive at the same normal form.
- Answering a query with respect to the residuum is very easy: A is true iff the residuum contains a rule $A \leftarrow$, and $\neg A$ is true iff no rule in the residuum contains A in its head.

One main drawback of this approach is that, due to the unfolding of clauses, the residuum could be *exponential* in the size of the original program. Therefore one might argue that our approach is not very efficient for computing WFS. But recently, Brass, Freitag, and Zukowski have shown ([28]) that the unfolding rule can be restricted so that unfolding of certain atoms are delayed. This results in a quadratic procedure to compute WFS which is provably better than, for example, the alternating fixpoint procedure on all inputs. The resulting normal form is different from the original residuum. It is called the *program remainder*. Both of these approaches have been implemented as Prolog-programs.

Other works related to well-founded semantics based on program transformation include [6, 4].

3. Positive Disjunctive Programs

In this section, we restrict ourselves to positive programs (where no negative literals appear in the body of a clause), and discuss how answer computation and answering negative queries can be carried out. Though answering positive queries for positive programs is a monotonic problem, it turns out that most classical theorem provers can only compute trivial answers and are unable to find *definite* (i.e. non-disjunctive) answers. In subsection 3.1, we briefly review a theorem proving calculus that can be successfully employed for computing answers.

Answering negative queries is generally carried out through Generalized Closed World Assumption (GCWA) which is equivalent to minimal model reasoning [58]. GCWA allows one to assume an atom to be false if it does not appear in any minimal model of the program. A weak form of GCWA, referred to as WGCWA, has also been proposed [73, 74]. Unlike

GCWA, WGCWA is defined using the notion of derived clauses, and an atom A is assumed to be false if there is no positive clause K s.t. $A \vee K$ is derivable from the program.

With the aim of merging theorem proving (computing answers for positive queries) and logic programming paradigms [3, 8] two approaches to handling query evaluation for positive disjunctive programs with respect to minimal model semantics have been studied. Subsection 3.2 discusses the first method which is based on abduction and restart model-elimination ([7]). The second method is explained in Subsection 3.3: it is based on extending the hyper tableau calculus of [25] (see [67, 66]).

3.1. Computing Answers using Restart ME

In [26], it has been shown that theorem provers using a special calculus, referred to as *restart model elimination calculus*, can be used as answer complete interpreters for positive disjunctive logic programs. On the declarative side, given a disjunctive logic program D and a query $\leftarrow Q$, a disjunction $Q\theta_1 \vee \cdots \vee Q\theta_n$ of instances of Q is a correct answer iff $P \models \forall (Q\theta_1 \vee \cdots \vee Q\theta_n)$. On the procedural side, restart model elimination calculus is used to build a tableau and the disjunction of all the instances of the query in the tableau is given as a computed answer. This process is illustrated by a very simple example below. More details can be obtained from [26].

Example 1 *Consider a disjunctive logic program with the following clauses: $on(a, b)$, $on(b, c)$, $co(a, gr)$, $co(c, bl)$, and $co(b, gr) \vee co(b, bl)$. This program describes a block world in which there are three blocks a, b, and c. It is known that a is on b and b is on c. The colour of a is green, c is blue, and that of b is either blue or green. Now consider a query $\leftarrow on(X, Y), co(X, gr), co(Y, bl)$. The answer computation process based on restart model elimination is shown in Figure 2. As shown a non-definite answer for the given goal is computed.* $\square$

The main result here is that for every correct answer there exists a computed answer that subsumes it and thus restart model elimination is an answer complete calculus for positive programs with positive queries. With respect to definite answers, the completeness result can be stated as follows: if $Q\theta$ is a correct definite answer, then this calculus can compute an answer $Q\sigma$ s.t. $Q\sigma\delta = Q\theta$ (for some substitution δ). To focus only on definite answers or to compute answers with less number of disjuncts, a variant of restart model elimination referred to as ancestry restart model elimination has been introduced and the details are in [26].

3.2. Minimal Model Reasoning based on Negation by Failure to Explain

In [7], we have developed an abductive framework for positive disjunctive logic programs that captures minimal model reasoning wrt both GCWA [58] and WGCWA [73, 74]. Given a program D and a goal G, an abductive explanation Δ for G consists only of negative literals s.t. $D \cup \Delta \models G$ and $D \cup \Delta$ is consistent. We have introduced an inference rule, referred to as *negation by failure to explain*, that allows us to infer negation of an atom A if there is no abductive explanation for A wrt D. It is also shown that negation by failure to explain is equivalent to negation by GCWA.

To generate abductive explanations of a given goal G wrt D, we have modified the *restart model elimination calculus* of [21, 26]. The modified calculus is used to generate all necessary (possibly empty) *potential candidates* Δ s.t. $D \cup \Delta \models G$, and consistency checks are then carried out to verify if $D \cup \Delta$ is consistent or not. If $\emptyset$ is found to be an explanation of G, then obviously G is a logical consequence of D and hence declared to be true. In case if no abductive explanation for G is found, then G is declared to be false under GCWA. If G has an abductive explanation and $\emptyset$ is not an explanation, then it is declared to be indefinite under GCWA.

Reasoning wrt WGCWA is relatively easier and requires no consistency check. G is declared to be false under WGCWA if there exists no potential candidate (possibly empty) to explain it. If G is not true (or alternatively $\emptyset$ is not a potential candidate) and has at least one potential candidate, then it is declared to be indefinite under WGCWA. Note that no consistency checks are performed to verify if the found potential candidates are indeed abductive explanations.

This approach to minimal model reasoning has been incorporated into the theorem prover PROTEIN[2] [22]. The following example illustrates the above ideas and how to use the system.

Example 2 *The left half of the Figure 3 lists a PROTEIN input file, a disjunctive logic program, and a query. PROTEIN uses Prolog like syntax with the sym-*

[2] Available on the web at <URL:http://www.uni-koblenz.de/ag-ki/Protein/>.

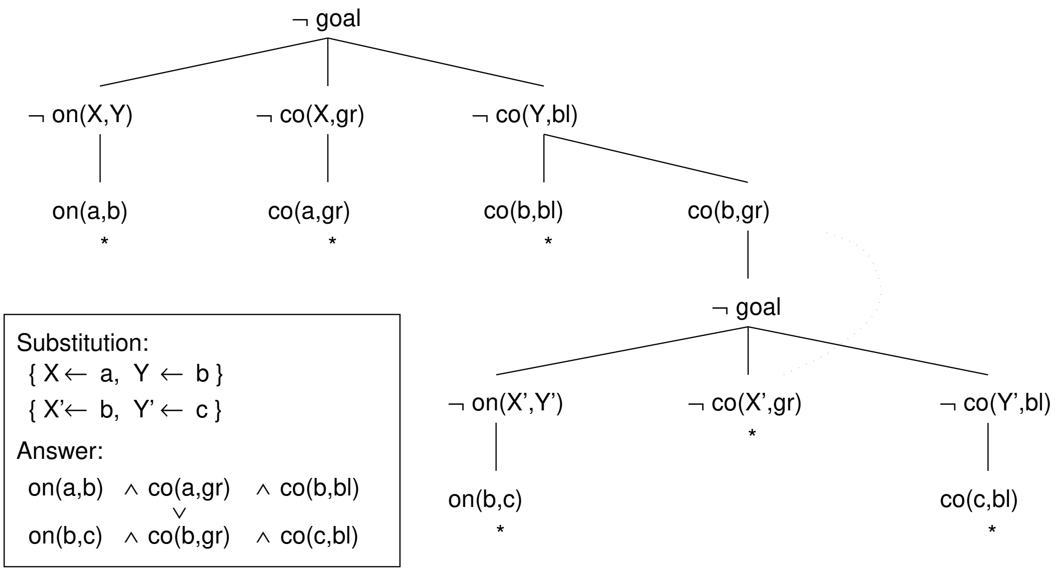

Fig. 2. Answer computing with Restart ME calculus

```
% protein_flag(mmr,wgcwa).
protein_flag(mmr,gcwa).

number(0).
number(s(X)) :- number(X).
number(X); bird(X).

?- bird(s(s(s(0)))).
% ?- bird(tweety).
```

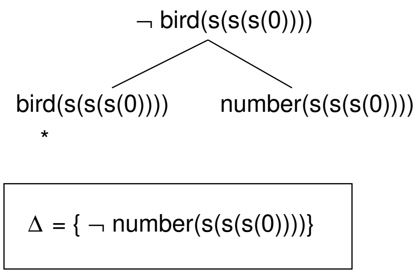

Fig. 3. Minimal model reasoning based on abduction

bol "`;`*" in the head of a clause that denotes disjunction. There are two flags "*`gcwa`*" and "*`wgcwa`*" for performing minimal model reasoning wrt GCWA and WGCWA respectively. Any line starting with the character "*`%`*" is discarded as comment.*

In this example, given the query `bird(s(s(s(0))))` *and the flag* `gcwa`*, PROTEIN first constructs a potential candidate for explaining this. As shown in the right half of the Figure 3, PROTEIN finds that* `s(s(s(0)))` *could be a* `bird` *if it is not a* `number`*. However, after a consistency check this is discarded, and since no more potential candidates can be generated, PROTEIN declares the given goal to be false. If the same query is run with* `wgcwa` *flag, PROTEIN declares the goal to be indefinite since it has found a potential explanation. No consistency check is carried out in this case.* □

3.3. Minimal Model Generation and Reasoning

We have developed a method for handling a very general form of minimal model reasoning (parallel circumscription with both fixed and varying predicates) [67, 66]. The method is based on a new technique for extending a tableau calculus to generate minimal models. As the underlying tableau method we have used a novel hyper tableau calculus [25] which merges features from tableau techniques and hyper resolution. This calculus is extended to minimal model generation by employing the new technique which exploits a groundedness property of minimal models and enables a one branch at a time approach to constructing tableaux for minimal model reasoning. The calculus is sound for general first-order circumscription and complete when no function symbols are allowed. In contrast to other methods for minimal model reasoning, the new technique leads to low space complexity and, e.g., in the ground case polynomial space complexity is obtained. We are not aware of any other implementation of minimal model reasoning with this property. The method can be used for both query-answering and minimal model generation and it can handle examples with hundreds of thousands of minimal models. The method has been implemented in Eclipse Prolog and is available at <URL:http://www.uni-koblenz.de/ag-ki/DLP/>.

Example 3 *The* `mm` *module of DisLoP is illustrated by the following two cases* `demo1` *and* `demo2`. `demo1` *consists of the following statements and instructs the* `mm` *module to compute all the minimal models of the given clauses:*

```
minimized(rest).
query_mode(all).

a1;b1;c1;d1;e1<-true.      a2 <- a3.
a2;b2;c2;d2;e2<-true.      a3 <- a4.
a3;b3;c3;d3;e3<-true.      a4 <- a5.
a4;b4;c4;d4;e4<-true.      a5 <- a1.
a5;b5;c5;d5;e5<-true.
```

`demo2` *consists of all the above statements plus the following additional query statement*

```
?- b1;c1;d1;e1;(a1,a2,a3,a4,a5).
```

which instructs the `mm` *module to verify whether the query is true in every minimal model. Now, a session with* `mm` *generates all 1365 minimal models for* `demo1` *and answers affirmatively to the query of* `demo2`*:*

```
[eclipse 2]: mm(demo1).
1. [e1, e2, e3, e4, e5]
2. [e1, e2, e3, e4, d5]
3. [e1, e2, e3, e4, c5]
.....
1363. [b1, a2, a3, a4, b5]
1364. [b1, a2, a3, a4, a5]
1365. [a1, a5, a4, a3, a2]

1365 model(s) found.
6300 ms. of CPU time used.

[eclipse 3]: mm(demo2).
Query succeeded.
0 countermodel(s) found.
0 ms. of CPU time used.

[eclipse 4]:
```

□

4. General Disjunctive Programs

For general disjunctive programs we have focused on STATIC [19] and the D-WFS semantics [9, 15, 10] and on generalizations of the abductive approach to incorporate negation. We have studied their mutual relationships, implementation methods and relationship to minimal model reasoning.

There are of course more semantics for disjunctive programs around. The first book on this topic was [56] which introduced many different versions most of which have been implemented by D. Seipel[3]. Also the stable semantics generalizes obviously to disjunctive programs ([53]) and there are also frameworks based on autoepistemic logic ([19]) as well as approaches to reduce the disjunctive case to normal programs [38]. In addition, there are works where different sorts of disjunction, ranging from *exclusive* to *inclusive* are allowed ([75, 76, 29]).

The reason why we focused on D-WFS is that (1) some of the semantics mentioned above are not *well-behaved* (see [39–41]) and (2) there is strong evidence that this semantics is the natural disjunctive counterpart of WFS. Also we can show that any semantics satisfying some natural conditions is an extension of D-WFS: any query decided under D-WFS gets the same truth value for stronger semantics. This means that in order to compute any of the stronger semantics, we can safely start first with D-WFS. Also the STATIC semantics is strongly related to D-WFS.

In Subsection 4.1 we present our transformation approach for ground programs. There is also a corresponding bottom-up computation which works for range-restricted Datalog programs and is closely related to our transformations. Subsection 4.2 describes an implementation of D-WFS using hyper tableaux. Unlike most other implementations of disjunctive semantics ours only requires polynomial space. Finally, we mention the approach [47] of Dix and Stolzenburg which generalizes the transformations from the ground to the non-ground case. This is ongoing work and subject to further research.

4.1. D-WFS Based on Transformations

Brass and Dix define in [11, 13] the semantics D-WFS as a confluent calculus of program transformation

[3] visit <URL:http://www-info1.informatik.uni-wuerzburg.de/-database/DisLog/introduction.html> for more information.

as well as a bottom-up computation using certain fixpoints. Both methods have been implemented with various settings (which transformations should be given higher priority and so on) and are available on the web[4] as Prolog programs.

All the transformations are defined for ground programs (see Subsection 4.3 for an extension to non-ground rules). The *Tautology*-transformation allows us to delete rules that contain the same atom in their head and in the body. The *Non-Minimal*-transformation deletes non-minimal rules: if both the head and the body of a rule r are contained in another rule r', then r' can be deleted.

Positive Reduction allows us to simply skip all occurrences of negative literals $\neg A$ if A does not occur in the head of any rule. *Negative Reduction* allows us to delete a rule with an occurrence of $\neg A$ in its body, if there is a rule $A \leftarrow$ in the program.

The *Unfolding*-Transformation is the most costly operation. It allows us to select any positive occurrence of an atom A in the body of a clause, and to replace this clause by resolving it with all *definitions* of A (these are all rules having A in their heads). Obviously, this transformation should only be applied when all others are no more applicable.

The main result then is that our calculus of transformations is confluent and terminating. This implies that any program has a normal form, called the *residuum*. This residuum contains sufficient information to answer a query immediately:

- $A_1 \vee A_2 \ldots \vee A_n$ is true in P iff the residuum contains a rule $\mathcal{A} \leftarrow$, with $\mathcal{A} \subseteq \{A_1, A_2, \ldots, A_n\}$
- $\neg A_1 \vee \neg A_2 \ldots \vee \neg A_n$ is true in P iff $\exists i_0 : 1 \leq i_0 \leq n$ such that A_{i_0} does not occur in any head of the residuum

In addition, sub-calculi suitable for positive disjunctive programs (GCWA) and normal programs (WFS) (described in [11, 13]) have been implemented. In particular, the Brass, Freitag and Zukowski method for WFS, which works in quadratic time and computes a slightly different residuum, is available.

Example 4 *Consider the following disjunctive logic program:*

```
a;b <- c, not c, not d.
c;d <- not e.
a;c <- b.
b <- not c, not d, not e.
```

Our implementation of the transformation approach described above computes the residuum of this program as follows:

```
Positive reduction:
  replacing c; d <- not e
  by c; d <-
Positive reduction:
  replacing b <- not c,not d,not e
  by b <- not c, not d
Negative reduction:
  removing a;b <- c, not c, not d
Negative reduction:
  removing b <- not c, not d
GPPE:
  replacing a;c <- b
  by nothing

residual program:
c ; d <-
```

□

In [10, 15] we have also developed a bottom-up approach to compute the residuum. In this approach, the fixpoint of conditional facts is computed first and later the transformations are applied to the computed fixpoint. This approach works also for range-restricted function-free disjunctive programs. In [9] we showed how to compute the stable semantics given the residuum. We are currently comparing these methods with the others described above.

Example 5 *Consider the same program as in Example 4. The bottom-up fixpoint approach to compute the residuum is carried out as follows:*

```
Fixpoint computation:
   new conditional facts
   c; d <- not e
   b <- not c, not d, not e
Fixpoint computation:
   new conditional facts
   a; b; d <- not c, not d, not e
   a; c <- not c, not d, not e
Fixpoint computation:
   new conditional facts
   b; a <- not c, not d, not e
   c; a; d <- not c, not d, not e

Nonminimal rule:
   removing b;a<-not c,not d,not e
    (b <- not c,not d,not e)
```

[4] <URL:http://www.uni-koblenz.de/ag-ki/DLP/>

```
Nonminimal rule:
   removing c;a;d<-not c,not d,not e
    (a;c <- not c,not d,not e)
Nonminimal rule:
   removing a;b;d<-not c,not d,not e
    (b <- not c,not d,not e)
Positive reduction:
   replacing a;c<-not c,not d,not e
   by a;c <- not c,not d
Positive reduction:
   replacing c; d <- not e
   by c; d <-
Positive reduction:
   replacing b<-not c,not d,not e
   by b <- not c, not d
Negative reduction:
   removing a; c <- not c,not d
Negative reduction:
   removing b <- not c,not d
```

□

4.2. *Implementing D-WFS with Polynomial Space*

Although D-WFS is defined using a calculus of program transformation without any reference to models, a connection between D-WFS and minimal models has been established [11, 13]. It has been further shown that if the notion of a model of a disjunctive program is strengthened, then D-WFS can be defined iteratively using standard parallel circumscription [18]. This enables us to develop a novel implementation method for D-WFS which, in contrary to most other implementations of disjunctive semantics, works in *polynomial space* in the ground case[5]. For related work towards polynomial space implementation of disjunctive semantics we refer to [79, 57] where the stable semantics is addressed.

The idea is to implement D-WFS as an iterative reduction on disjunctive logic programs. The reduction employs parallel circumscription and classical reasoning. It starts with the original program and leads to a reduced program with the property that every query can be answered from this program with one call to a theorem prover for parallel circumscription. This can be seen as a compilation or a partial evaluation of the program leading to a smaller program from which all queries can be answered. The polynomial space complexity is obtained by combining the reduction approach with the new technique for minimal model reasoning described in Section 3.3. The method is also applicable to a restricted form of the STATIC semantics (referred to as flat STATIC), which follows from our result that flat STATIC coincides with D-WFS [18].

Example 6 *Consider the same program as in Example 4. The* `dwfs_mm` *module of DisLoP first computes the following partial evaluation of the program:*

```
c ; d <- true
a ; c <- b
b <- not c , not d
a ; b <- c , not c , not d
```

Now a query can be answered through a theorem prover call (with parallel circumscription) with this compiled program. For example, the query `?- c ; d.` *succeeds with this program.* □

4.3. *D-WFS for arbitrary first-order programs*

One of the most important advantages of the logic programming paradigm and therefore the success of Prolog is its ability to compute answer-substitutions for a given query. Although semantics for logic programs with negation are undecidable if function symbols and variables are allowed, we are convinced that query answering mechanisms for the non-ground-case have significant advantages over the propositional case. Of course, such procedures can only be sound and not complete. But completeness can hold for certain restricted classes of programs as well as for certain queries.

Recently ([47], an extended version to appear as [48]), Dix and Stolzenburg have extended the transformation calculus presented in the previous subsection to non-ground programs with function symbols. The problem here is the Unfolding-Transformation, which is not sound for rules with variables because of the occurrence of unifiable atoms in the heads of rules.

However, we can make Unfolding sound (which allows us to use the results of [11, 13]) by introducing *inequality constraints*. This immediately leads us to introduce *constraint disjunctive logic programs* and consequently to extend our transformations to this class of programs. Surprisingly, this extended framework retains the same nice properties as our original calculus. In fact we can lift the results of [11, 13] to the non-ground case: (1) the new calculus is confluent for arbitrary programs, (2) for finite ground programs it is equivalent to the terminating calculus introduced in

[5]For an implementation, see <URL:http://www.uni-koblenz.de/-ag-ki/DLP/>.

Subsection 4.1, and (3) it approximates a generalization of D-WFS for arbitrary programs.

In principle, any constraint theory known from the field of constraint logic programming can be exploited in the context of non-monotonic reasoning, not only equational constraints over the Herbrand domain. However, the respective constraint solver must be able to treat negative constraints of the considered constraint domain.

In this respect, this work yields the basis for a general combination of two paradigms: constraint logic programming and non-monotonic reasoning ([48]).

4.4. *Comparison with the STATIC Semantics*

The STATIC semantics ([19]) for disjunctive logic programs is stronger than D-WFS: there are queries that are answered yes or no by the STATIC semantics while they are undetermined under D-WFS.

Example 7 *Consider the following disjunctive logic program:*

$$\begin{aligned} P : a \vee c &\leftarrow \neg d \\ c &\leftarrow \neg c \\ d &\leftarrow c \end{aligned}$$

Now STATIC concludes $\neg a$ but D-WFS does not. □

The reason for this behaviour is that in STATIC, a program is translated into a belief theory. Negative literals $\neg A$ in such programs are translated into $\mathcal{B}_{el}(\neg A)$. But such theories are treated in a modal logic, where we usually have the well-known (K)–axiom $\mathcal{B}_{el}(F \to G) \to (\mathcal{B}_{el}F \to \mathcal{B}_{el}G)$. This axiom implies in our example $\neg c \leftarrow \neg d$ in the corresponding belief theory.

However, STATIC is still weaker than STABLE, even for stratified programs. Nevertheless, STATIC is strongly related to D-WFS as shown in [18] (D-WFS coincides with flat STATIC).

We are currently investigating how the method from Subsection 4.2 can be used to implement full STATIC. Another implementation for STATIC (done by Stefan Brass) is also available. It is based on the model-theoretical characterization [19, Theorem 4.1].

5. Test Case Generation

We have developed a system for generating test cases for disjunctive logic programming systems and for theorem provers. Currently, the test cases are positive ground programs (sets of ground clauses) and they are generated using the following idea: a graph and an NP-complete problem are selected and then from the graph a set of clauses is produced so that the clause set has a model if and only if there is a solution for the given problem for this particular graph. The advantages of the approach include that families of test cases with similar structure but increasing size and complexity can be easily generated, a large variety of different types of test cases are available, and test cases can be grounded to realistic data by choosing interesting graphs representing real-life information.

These kinds of test cases are particularly useful for developing an automated reasoning system. They provide a systematic way of analyzing experimentally effects of different design decisions on the performance. Families of similar test cases with increasing complexity are also important for evaluating the scalability of the implementation and for determining the limits of its performance.

Our system can be seen as an extension of the TheoryBase system [30] which generates test default theories and normal logic programs using the same idea of creating test cases from graph problems. For generating graphs, TheoryBase uses Knuth's Stanford GraphBase [55] which is a portable collection of programs and data that serves as a platform for combinatorial algorithms and that is capable of generating and manipulating very many different kinds of graphs. TheoryBase provides a nice interface to producing graphs using GraphBase. We employ this interface for creating graphs and extend the scope of TheoryBase to disjunctive programs by providing new translations from graph problems to disjunctive programs.

We have developed a web interface for our system through which test cases can be generated by filling out a form describing the parameters of the desired test case. To visit the interface, see the project home page at <URL:http://www.uni-koblenz.de/ag-ki/DLP/>.

6. Other Activities

During the course of the DisLoP project, we have organized several workshops that are related to the aims of the project:

– IJCAI '95 Workshop on *Applications and Implementations of Non-monotonic Reasoning Systems*, Montreal, Canada, August 21, 1995 [20].

- Dagstuhl Seminar 9627: *Disjunctive Logic Programming and Databases: Non-monotonic Aspects*, Schloß Dagstuhl, Germany, July 1–5, 1996 [42][6].
- ECAI '96 Workshop on *Integrating Non-monotonicity into Automated Reasoning Systems*, Budapest, Hungary, August 12, 1996 [68].
- JICSLP '96 Postconference Workshop on *Non-monotonic Extensions of Logic Programming*, Bad Honnef, Sep 5–6, 1996 [44] (see also [43]).

An important outcome of the Dagstuhl Seminar 9627 is that we volunteered to construct a web page to collect and disseminate information on various logic programming systems that concentrate on non-monotonic aspects (different kinds of negation, disjunction, abduction etc.). This web page is actively maintained at <URL:http://www.uni-koblenz.de/ag-ki/LP/>.

We are also organizing two major international conferences:

- LPNMR '97 (<URL:http://www.uni-koblenz.de/-~lpnmr97/> and [36]),
- JELIA '98 (<URL:http://www.uni-koblenz.de/-~jelia98/> and [35]).

as well as workshops and tutorials related to our work:

- TAB '97 Tutorial on *Clause Normal Form Tableaux*, Pont-a-Mousson, France, May 13, 1997.
  (<URL:http://www.loria.fr/~galmiche/>)
- ESSLLII '97 Tutorial on *Knowledge Representation with Extended Logic Programs*, Aix-en-Provence, France, August 11–16, 1997.
  (<URL:http://www.lpl.univ-aix.fr/~esslli97/> and [16])
- KI '97 Workshop on *Inference-systems from a logical and a cognitive viewpoint*, Freiburg, Germany, September 9, 1997.
  (<URL:http://www.coling.uni-freiburg.de/workshop97/WS-10.html>)
- ILPS '97 Workshop on *Logic Programming and Knowledge Representation*, Port Jefferson, NY, USA, October 16, 1997.
  (see <URL:http://www.cs.ucr.edu/~teodor/cfp-97.html> and [46]).

[6] <URL:http://www.uni-koblenz.de/ag-ki/dag9627/>

7. Applications

We live in a constantly changing and uncertain world and this reflects in the knowledge/information/data that we have at hand. More often the knowledge is indefinite and in a constant state of flux. Disjunctive logic programming is a natural way of representing and dealing with such knowledge. One real-world example is the set of rules used by a bank to calculate banking fees. Because of the indefiniteness of the rules, it is not obvious whether a banking fee can be calculated for any given situation and whether only one fee exists for a given situation. We successfully used constraint disjunctive logic programs to deal with such questions [78].

The disjunctive techniques that we have developed during the course of this project (such as minimal model reasoning, abduction, semantics of negation etc.) are useful in various applications such as diagnosis and database updates. For example, [24, 23] explores how hyper tableau calculus [25] for disjunctive programs along with the minimal model reasoning technique of [67, 66] can be exploited for diagnosis. In a diagnosis setup [72], we have a logical description of the system which clearly describes the behaviour of the system when all components are functioning normally. For a given input vector an observation of the output is made. If this observation contradicts the expected behaviour, the purpose of the diagnosis process is to identify those components that could be faulty. To carry out this task, the key idea of [24, 23] is to transform the given system description, input vector, and the observation into a disjunctive logic program in such a way that minimal model reasoning (minimal wrt the abnormality of the components) with the transformed program provides diagnosis for the original system. Note the use of disjunctive techniques even when the system description does not have any disjunctions!

The diagnosis task is not very different from carrying out view updates in deductive and relational databases (see for example [5]). Especially, it is very closely related to the problem of deleting a view atom. So, disjunctive techniques can also be used for updating databases and this aspect has been explored in detail in [2]. The algorithm presented there employs a transformation approach whereby the given database together with the update request is transformed into a disjunctive logic program in such a way that the (minimal) models of the transformed program correspond to the update possibilities. We have studied two variants of our algorithm, where both the variants run in polynomial space

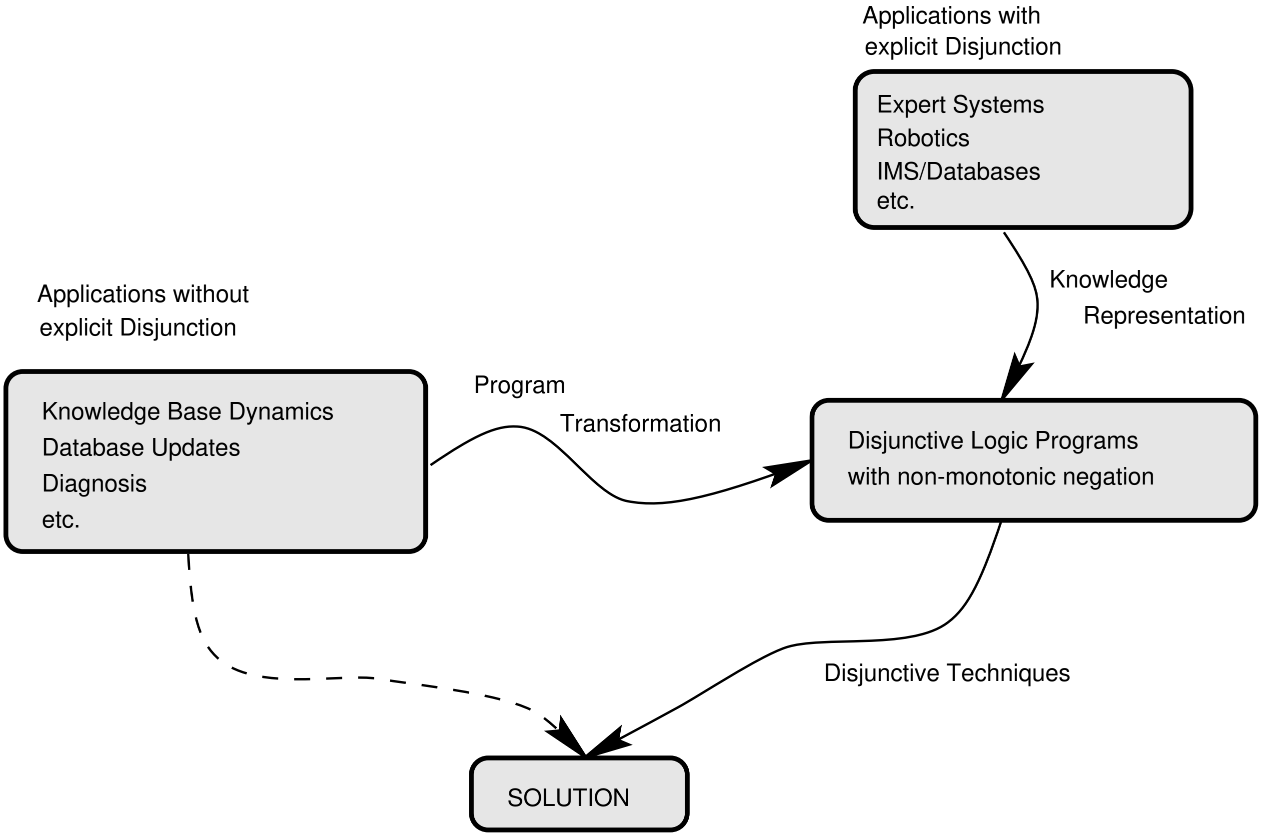

Fig. 4. Applications of Disjunctive Logic Programming

and one variant (which needs off-line pre-processing) runs in polynomial time too. An important aspect of both these variants is that they are rational with respect to certain postulates that are justified from the philosophical works on belief dynamics (see for example [5]).

Another important application of disjunctive techniques in the database area is to glue together different heterogeneous databases to provide a single unified view to the user. With the ever expanding world wide web technology, millions and millions of data in thousands of different formats are thrown at a user who clearly needs some tools to put together information of interest. We have initiated a separate project to address such issues and are beginning to use ideas and concepts developed for the DisLoP project. For example, the system that integrates different heterogeneous databases needs to know what kinds of external database calls are necessary to answer a particular query from a user. Abduction naturally solves such problems and the system looks for abductive explanations for the query in terms of external calls. There are many other aspects where disjunctive techniques could be employed and we are currently studying them in detail.

Other areas for applications of disjunctive techniques include robotics, planning, hypothetical reasoning etc. We plan to explore such areas in the near future.

8. Acknowledgements

The authors would like to thank all the members of the Artificial Intelligence Research Group at the University of Koblenz, Germany, for fruitful discussions and their meticulous reading of the drafts of this paper. Thanks are also due to the anonymous referees for their constructive comments. The research project DisLoP is funded by DFG under grant `Fu 263/3-1`.

References

[1] Krzysztof R. Apt and Roland N. Bol. Logic Programming and Negation: A Survey. *Journal of Logic Programming*, 19-20:9–71, 1994.

[2] Chandrabose Aravindan and Peter Baumgartner. A rational and efficient algorithm for view deletion in databases. In Jan Maluszynski, editor, *Proceedings of International Logic Programming Symposium*. The MIT Press, 1997. Also available as Technical Report RR–10–97 from the University of Koblenz, Germany.

[3] C. Aravindan, P. Baumgartner, J. Dix, U. Furbach, G. Neugebauer, I. Niemelä, D. Schäfer, and F. Stolzenburg. On merging theorem proving and logic programming paradigms. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, page 546, Bonn, Germany, September 1996. The MIT Press.

[4] Chandrabose Aravindan and Phan Minh Dung. Partial deduction of logic programs wrt well-founded semantics. *New Generation Computing*, 13:45–74, 1994. A related report is available on the web from <http://www.uni-koblenz.de/∼arvind/papers/>.

[5] Chandrabose Aravindan and Phan Minh Dung. Knowledge base dynamics, abduction, and database updates. *Journal of Applied Non-Classical Logics*, 5(1):51–76, 1995. A related report is available on the web from <http://www.uni-koblenz.de/∼arvind/papers/>.

[6] Chandrabose Aravindan and Phan Minh Dung. On the correctness of unfold/fold transformations of normal and extended logic programs. *The Journal of Logic Programming*, 24(3):201–218, September 1995. A related report is available on the web from <http://www.uni-koblenz.de/∼arvind/papers/>.

[7] Chandrabose Aravindan. An abductive framework for negation in disjunctive logic programming. In J. J. Alferes, L. M. Pereira, and E. Orlowska, editors, *Proceedings of Joint European workshop on Logics in AI*, number 1126 in Lecture Notes in Artificial Intelligence, pages 252–267. Springer-Verlag, 1996. A related report is available on the web from <http://www.uni-koblenz.de/∼arvind/papers/>.

[8] Chandrabose Aravindan. DisLoP: A disjunctive logic programming system based on PROTEIN theorem prover. In G. Görz and S. Hölldobler, editors, *Proceedings of KI'96*, number 1137 in Lecture Notes in Artificial Intelligence, pages 19–23. Springer-Verlag, 1996.

[9] Stefan Brass and Jürgen Dix. A General Approach to Bottom-Up Computation of Disjunctive Semantics. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 927, pages 127–155. Springer, Berlin, 1995.

[10] Stefan Brass and Jürgen Dix. Disjunctive Semantics based upon Partial and Bottom-Up Evaluation. In Leon Sterling, editor, *Proceedings of the 12th Int. Conf. on Logic Programming, Tokyo*, pages 199–213. MIT Press, June 1995.

[11] Stefan Brass and Jürgen Dix. Characterizing D-WFS: Confluence and Iterated GCWA. In L.M. Pereira J.J. Alferes and E. Orlowska, editors, *Logics in Artificial Intelligence* (*JELIA '96*), LNCS 1126, pages 268–283. Springer, 1996.

[12] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Stable Semantics by Partial Evaluation. *Journal of Logic Programming*, 32(3):207–228, 1997. (Extended abstract appeared in: Characterizations of the Stable Semantics by Partial Evaluation *LPNMR, Proceedings of the Third International Conference, Kentucky*, pages 85–98, 1995. LNCS 928, Springer.).

[13] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA. *Journal of Automated Reasoning*, to appear, 1998. (Extended abstract appeared in: Characterizing D-WFS: Confluence and Iterated GCWA. *Logics in Artificial Intelligence, JELIA '96*, pages 268–283, 1996. Springer, LNCS 1126.).

[14] Stefan Brass and Jürgen Dix. Semantics of Disjunctive Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, accepted for publication, 1998. (Extended abstract appeared in: Disjunctive Semantics Based upon Partial and Bottom-Up Evaluation, *Proceedings of the 12-th International Logic Programming Conference, Tokyo*, pages 199–213, 1995. MIT Press.).

[15] Stefan Brass and Jürgen Dix. Semantics of Disjunctive Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, accepted for publication, 1998.

[16] Gerhard Brewka and Jürgen Dix. Knowledge representation with logic programs. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition, Volume 6, Methodologies*, chapter 6. Reidel Publ., 1998.

[17] Gerd Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning: An Overview*. CSLI Lecture Notes 73. CSLI Publications, Stanford, CA, 1997.

[18] Stefan Brass, Jürgen Dix, Ilkka Niemelä, and Teodor. C. Przymusinski. Comparison and Efficient Computation of the Static and the Disjunctive WFS. In Gerd Brewka, Emil Weydert, and Cees Witteveen, editors, *Proceedings of the third Dutch-German Workshop on Nonmonotonic Reasoning and its Applications*, pages 37–42. February 1997. appeared also as TR 2/96.

[19] Stefan Brass, Jürgen Dix, and Teodor. C. Przymusinski. Super Logic Programs. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference* (*KR '96*), pages 529–541. San Francisco, CA, Morgan Kaufmann, 1996.

[20] Rachel Ben-Eliyahu and Ilkka Niemelä, editors. *Working Notes of the IJCAI'95 Workshop on Applications and Implementations of Nonmonotonic Reasoning Systems*, Montreal, Canada, August 1995. AAAI Press.

[21] Peter Baumgartner and Ulrich Furbach. Model Elimination without Contrapositives and its Application to PTTP. *Journal of Automated Reasoning*, 13:339–359, 1994. Short version in: Proceedings of CADE-12, Springer LNAI 814, 1994, pp 87–101.

[22] Peter Baumgartner and Ulrich Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *Lecture Notes in Aritificial Intelligence*, pages 769–773. Springer, 1994. Available at the URL: http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/.

[23] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. In *15th International Joint Conference on Artificial Intelligence* (*IJ-*

CAI 97), Nagoya, 1997. International Joint Conference on Artificial Intelligence. To appear.

[24] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Tableaux for Diagnosis Applications. In Didier Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, number 1227 in Lecture Notes in Aritificial Intelligence, pages 76–90. Springer, 1997.

[25] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in Lecture Notes in Aritificial Intelligence. European Workshop on Logic in AI, Springer, 1996. (Long version in: *Fachberichte Informatik*, 8–96, Universität Koblenz-Landau).

[26] Peter Baumgartner, Ulrich Furbach, and Frieder Stolzenburg. Computing Answers with Model Elimination. *Artificial Intelligence*, 90(1–2):135–176, 1997.

[27] Chitta Baral and Michael Gelfond. Logic Programming and Knowlege Representation. *Journal of Logic Programming*, 19-20, 1994.

[28] Stefan Brass, Ulrich Zukowski, and Burkhardt Freitag. Transformation Based Bottom-Up Computation of the Well-Founded Model. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 171–201. Springer, Berlin, 1997.

[29] Edward P.F. Chan. A Possible World Semantics for Disjunctive Databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(2):282–292, 1993.

[30] P. Cholewiński, V.W. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Proceedings of the 12th International Conference on Logic Programming*, pages 267–281, Tokyo, June 1995.

[31] Weidong Chen, Terrance Swift, and David S. Warren. Efficient Top-Down Computation of Queries under the Well-Founded Semantics. *Journal of Logic Programming*, 24(3):219–245, 1995.

[32] W. Chen and D.S. Warren. The SLG system, 1993. Available by ftp from seas.smu.edu:pub or sbcs.sunysv.edu:pub/XSB.

[33] Weidong Chen and David S. Warren. Computing of Stable Models and its Integration with Logical Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 17:279–300, 1995.

[34] Weidong Chen and David S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, January 1996.

[35] J. Dix, L. Farinas del Cerro, and U. Furbach, editors. *Logics in Artificial Intelligence*, to appear, Berlin, 1998. Springer.

[36] J. Dix, U. Furbach, and A. Nerode, editors. *Logic Programming and Nonmonotonic Reasoning*, LNAI 1265, Berlin, 1997. Springer.

[37] Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In Andrei Voronkov and Alan Robinson, editors, *Handbook of Automated Reasoning*. Elsevier-Science-Press, to appear 1998.

[38] Jürgen Dix, Georg Gottlob, and Viktor Marek. Reducing disjunctive to non-disjunctive semantics by shift-operations. *Fundamenta Informaticae*, XXVIII(1/2):87–100, 1996.

[39] Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.

[40] Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.

[41] Jürgen Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In Andre Fuhrmann and Hans Rott, editors, *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, pages 241–327. DeGruyter, 1995.

[42] Jürgen Dix, Donald Loveland, Jack Minker, and David. S. Warren. Disjunctive Logic Programming and databases: Nonmonotonic Aspects. Technical Report Dagstuhl Seminar Report 150, IBFI GmbH, Schloß Dagstuhl, 1996.

[43] J. Dix, L. Pereira, and T. Przymusinski, editors. *Non-Monotonic Extensions of Logic Programming*, LNAI 927, Berlin, 1995. Springer.

[44] J. Dix, L. Pereira, and T. Przymusinski, editors. *Non-Monotonic Extensions of Logic Programming*, LNAI 1216, Berlin, 1997. Springer.

[45] Jürgen Dix, Luis Pereira, and Teodor Przymusinski. Prolegomena to Logic Programming for Non-Monotonic Reasoning. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 1–36. Springer, Berlin, 1997.

[46] Jürgen Dix, Luis Moniz Pereira, and Teodor Przymusinski. Logic Programming and Knowledge Representation. Technical Report TR 19/97, University of Koblenz, Department of Computer Science, Rheinau 1, September 1997.

[47] Jürgen Dix and Frieder Stolzenburg. Computation of Non-Ground Disjunctive Well-Founded Semantics with Constraint Logic Programming (preliminary report). In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 202–226. Springer, Berlin, 1997.

[48] Jürgen Dix and Frieder Stolzenburg. A Framework to incorporate Nonmonotonic Reasoning into Constraint Logic Programming. *Journal of Logic Programming*, 1998. Special Issue on *Constraint Logic Programming*, Guest Editors: Kim Marriott and Peter Stuckey, to appear in 1998. Also appeared as Technical Report 19/97 at Dept. of CS, University of Koblenz-Landau, May 1997.

[49] Phan Minh Dung. On the relations between stable and well-founded semantics of logic programs. *Theoretical Computer Science*, 105:7–25, 1992.

[50] Phan Minh Dung. An argumentation theoretic foundation for logic programming. *The Journal of Logic Programming*, 22(2):151–177, 1995.

[51] Phan Minh Dung. On the acceptability of argument and its fundamental role in non-monotonic reasoning and logic programming and n-person game. *Artificial Intelligence*, 76, 1995.

[52] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[53] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–387, 1991. (Extended abstract appeared in: Logic Programs with Classical Negation. *Proceedings of the 7-th International Logic Programming Conference, Jerusalem*, pages 579-597, 1990. MIT Press.).

[54] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, 1992.

[55] D.E. Knuth. *Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley, 1993.

[56] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT-Press, 1992.

[57] N. Leone, L. Rullo, and F. Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.

[58] Jack Minker. On indefinite databases and the closed world assumption. In *Lecture Notes in Computer Science 138*, pages 292–308. Springer-Verlag, 1982.

[59] Jack Minker. An Overview of Nonmonotonic Reasoning and Logic Programming. *Journal of Logic Programming, Special Issue*, 17, 1993.

[60] Jack Minker. Logic and databases: A 20 year retrospective. In Dino Pedreschi and Carlo Zaniolo, editors, *Proceedings of the International Workshop on Logic in Databases* (*LID*), LNCS 1154, pages 3–58. Springer, Berlin, 1996.

[61] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991.

[62] Wiktor Marek and Mirek Truszczyński. *Nonmonotonic Logics; Context-Dependent Reasoning*. Springer, Berlin, 1st edition, 1993.

[63] I. Niemelä. A decision method for nonmonotonic reasoning based on autoepistemic reasoning. *Journal of Automated Reasoning*, 14:3–42, 1995.

[64] I. Niemelä. Towards efficient default reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 312–318, Montreal, Canada, August 1995. Morgan Kaufmann Publishers.

[65] I. Niemelä. Autoepistemic logic as a basis for automating nonmonotonic reasoning. In Patrick Doherty, editor, *Partiality, Modality and Non-monotonicity*, pages 251–289. CSLI Publications, Stanford, CA, 1996.

[66] I. Niemelä. Implementing circumscription using a tableau method. In *Proceedings of the European Conference on Artificial Intelligence*, pages 80–84, Budapest, Hungary, August 1996. John Wiley.

[67] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 278–294, Terrasini, Italy, May 1996. Springer-Verlag.

[68] Ilkka Niemelä, editor. *Proceedings of the ECAI'96 Workshop on Integrating Nonmonotonicity into Automated Reasoning Systems*, Budapest, Hungary, August 1996. Universität Koblenz-Landau, Fachbericht Informatik 18–96. Available at http://www.uni-koblenz.de/universitaet/fb4/publications/GelbeReihe/.

[69] I. Niemelä and P. Simons. Evaluating an algorithm for default reasoning. In *Working Notes of the IJCAI'95 Workshop on Applications and Implementations of Nonmonotonic Reasoning Systems, Montreal, Canada*, pages 66–72, Montreal, Canada, August 1995.

[70] Ilkka Niemelä and Patrik Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

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

[72] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[73] Arcot Rajasekar, Jorge Lobo, and Jack Minker. Weak generalized closed world assumption. *Journal of Automated Reasoning*, 5:293–307, 1989.

[74] K. A. Ross and R. W. Topor. Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, 4(2):397–424, 1988.

[75] Chiaki Sakama. Possible Model Semantics for Disjunctive Databases. In Won Kim, Jean-Marie Nicolas, and Shojiro Nishio, editors, *Deductive and Object-Oriented Databases, Proceedings of the First International Conference* (*DOOD89*), pages 369–383, Kyoto, Japan, 1989. North-Holland Publ.Co.

[76] Ch. Sakama and K. Inoue. An Alternative Approach to the Semantics of Disjunctive Logic Programs and Deductive Databases. *Journal of Automated Reasoning*, 13:145–172, 1994.

[77] K. Sagonas, T. Swift, and D. S. Warren. An abstract machine for computing the Well-Founded semantics. In Michael Maher, editor, *Proceedings of Joint International Conference and Symposium on Logic Programming*, pages 274–288. The MIT Press, 1996.

[78] Frieder Stolzenburg and Bernd Thomas. Analysing rule sets for the calculation of banking fees by a theorem prover with constraints. In *Proceedings of the 2nd International Conference on Practical Application of Constraint Technology*, pages 269–282, London, 1996. Practical Application Company.

[79] J. Stuber. Computing stable models by program transformation. In Pascal Van Hentenryck, editor, *Proceedings of International Conference on Logic Programming*, pages 58–73. The MIT Press, 1994.

[80] Allen van Gelder, Kenneth A. Ross, and J. S. Schlipf. Unfounded Sets and well-founded Semantics for general logic Programs. In *Proceedings 7th Symposion on Principles of Database Systems*, pages 221–230, 1988.

[81] Allen van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.