QUIP—A Tool for Computing Nonmonotonic Reasoning Tasks[*]

Uwe Egly, Thomas Eiter, Hans Tompits, and Stefan Woltran
Technische Universität Wien
Abt. Wissensbasierte Systeme 184/3
Favoritenstraße 9–11, A–1040 Wien, Austria
e-mail: [uwe,eiter,tompits,stefan]@kr.tuwien.ac.at

Abstract

In this paper, we outline the prototype of an automated inference tool, called QUIP, which provides a uniform implementation for several nonmonotonic reasoning formalisms. The theoretical basis of QUIP is derived from well-known results about the computational complexity of nonmonotonic logics and exploits a representation of the different reasoning tasks in terms of *quantified boolean formulae* (QBFs).

General Information

Designing theorem provers for nonmonotonic reasoning formalisms is challenged by the increased computational complexity compared to the inherent complexity of classical reasoning. As is well-known (Gottlob 1992; Eiter & Gottlob 1993; Eiter & Gottlob 1995a; Eiter & Gottlob 1995b), the main reasoning tasks for almost all propositional nonmonotonic logics are either Σ_2^p-complete or Π_2^p-complete, i.e., they are located at the second level of the polynomial hierarchy, whereas classical reasoning resides at the first level of that hierarchy.

Important constituents in establishing the Σ_2^p-completeness results for the different nonmonotonic reasoning tasks are so-called *quantified boolean formulae* (QBFs), which generalize ordinary propositional formulae by the admission of quantifiers ranging over propositional variables. To wit, demonstrating Σ_2^p-hardness of a given nonmonotonic decision problem, say NMP, is usually achieved by reducing the problem QSAT$_2$ to NMP, where QSAT$_2$ is the problem of deciding the truth of QBFs being in prenex normal-form whose leading quantifiers are ordered like $\exists P \forall Q$ (P, Q are lists of propositional variables), which is complete for Σ_2^p. (In general, for any class Σ_i^p, $i \geq 1$, there is a corresponding decision problem, QSAT$_i$, complete for Σ_i^p, whose task is to check the truth of QBFs of the form $\exists P_1 \forall P_2 \ldots \exists P_{n-1} \forall P_n \; \Phi$, where $P_1, \ldots, P_n$ are lists of propositional variables and Φ is a propositional formula.) More precisely, Σ_2^p-hardness of NMP is shown by constructing a (polynomial) transformation $\mathcal{S}$ such that for every instance I of QSAT$_2$ it holds that I is a yes-instance of QSAT$_2$ iff $\mathcal{S}(I)$ is a yes-instance of NMP.

Now, given the membership of a nonmonotonic reasoning task NMP in the class Σ_2^p, and from the Σ_2^p-hardness of QSAT$_2$, it follows immediately that every such decision problem NMP can be reduced to QSAT$_2$, i.e., there is a (polynomial) transformation $\mathcal{T}$ such that for every instance I of NMP it holds that I is a yes-instance of NMP iff $\mathcal{T}(I)$ is a yes-instance of QSAT$_2$.

In this paper, we describe the prototype system QUIP (Egly *et al.* 2000a; Egly *et al.* 2000b), an automated reasoning tool utilizing transformations of the latter kind to implement several different reasoning formalisms. The basic idea is to employ *existing sophisticated theorem provers for quantified boolean formulae*, taking care of evaluating the resultant instances of QSAT$_2$.

At present, QUIP handles the following propositional nonmonotonic reasoning approaches:

- abduction;
- autoepistemic logic;
- default logic;
- disjunctive stable model semantics;
- circumscription.

The system has been implemented in C using standard tools like LEX and YACC (comprising a total of 2000 lines of code, excluding the used QBF-solver); it runs currently in a Unix environment (Sun/Solaris and Linux), but is easily portable to other operating systems as well.

The use of QBFs expressing advanced reasoning tasks has been advocated in (Cadoli, Giovanardi, & Schaerf 1998; Rintanen 1999b) and is a natural generalization of a similar method applied for problems in NP. Such problems are often solved by a reduction to SAT, the satisfiability problem of classical propositional logic, which is NP-complete (see, e.g., (Kautz & Selman 1996; Giunchiglia & Sebastiani 1996)). Besides the applications discussed in this paper, a reduction of planning problems to QBFs has been given in (Rintanen 1999a).

In general, since the evaluation of *arbitrary* QBFs is PSPACE-complete (in contrast to the Σ_i^p-completeness

[*]This work was partially supported by the Austrian Science Fund Project N Z29-INF.

of the restricted QBFs mentioned above), in principle *any* formalism having a decision problem in PSPACE can be handled by QUIP, provided a proper transformation has been found.

The next section outlines the overall architecture of QUIP and gives some background information on the different reasoning tasks implemented in QUIP. Then, the usability of the approach is described. The last section contains a discussion on benchmark problems and a comparison with other implementations.

Description of the System

System Architecture

The overall architecture of QUIP is depicted in Figure 1. QUIP consists of three parts, namely the `filter` program, the QBF-evaluator `boole`, and the output interpreter `int`.

The input filter translates the given problem description (e.g., a default theory, a disjunctive logic program, an abductive problem, etc.) into a quantified boolean formula, which is then sent to the QBF-evaluator `boole`. The result of `boole`, usually a formula in disjunctive normal form (often called *sum of products*, SOP), is interpreted by `int`. The latter part associates a meaningful interpretation to the formulae occurring in SOP and provides an explanation in terms of the underlying problem instance (e.g., an extension, a stable model, abductive explanations, etc.). The interpretation relies on a mapping of internal variables of the generated QBF into concepts of the problem description which is provided by `filter`.

The QBF-evaluator `boole` is a commonly available propositional theorem prover based on *binary decision diagrams* (BDDs) (Bryant 1986).[1] We chose to use this particular tool because of several reasons. For one, the program, together with its source code, is in the public domain and can be downloaded from the web (see http://www.cs.cmu.edu/∼modelcheck/bdd.html). Second, it represents a sophisticated reasoning engine with several years of development. Third, it does not require of the input formula to be in a specific normal form. The latter point distinguishes `boole` from other QBF-solvers, like those proposed in (Cadoli, Giovanardi, & Schaerf 1998; Rintanen 1999b), which operate on formulae being in *prenex conjunctive normal form.* Although these provers can in principle also be employed in QUIP, their inclusion would require an additional normal form translation, since the "natural" reductions of nonmonotonic reasoning tasks to QBFs do in general *not* result in formulae being in a particular normal form.

In order to incorporate new formalisms into QUIP, one has to extend the `filter` program responsible for the appropriate reductions, the mapping of the variables, and the interpreter `int`. The deductive engine remains unchanged in this process.

[1]For a comparison between different BDD packages, cf. (Long 1998).

Implemented Reasoning Tasks

In this subsection, we briefly discuss the different reasoning tasks currently implemented in QUIP. We will not present the concrete transformations into QBFs expressing these tasks; details are given in (Egly *et al.* 2000b).

All formalisms processed by QUIP are propositional. In what follows, we assume a propositional language $\mathcal{L}$ generated by a finite set of propositional variables V using the standard sentential connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$. A *theory* is a finite set $T \subseteq \mathcal{L}$, which will be identified with the formula $\bigwedge_{\phi \in T} \phi$.

Note that, strictly speaking, the different decision problems presented below return either *yes* or *no*, but usually one is more interested in the corresponding *function problem* returning the actual objects responsible for a *yes* answer (like the extensions of a default theory, the stable models of a logic program, etc.). QUIP permits queries of this form, by engaging the so-called *detail mode* (see discussion below).

Abduction. Classical abduction from a theory T on V may be defined as follows (Selman & Levesque 1990; Poole 1989; Eiter & Gottlob 1995b). Let $H \subseteq V$ be a set of *hypotheses*, and let $p \in V$ be a distinguished atom. A subset $E \subseteq H$ is an *abductive explanation* for p from T and H, if

(i) $T \cup E$ is consistent, and

(ii) $T \cup E \models p$, i.e., $T \cup E$ logically implies p.

An explanation E is *minimal*, if no proper subset $E' \subset E$ is an abductive explanation.

The following tasks are implemented in QUIP:

- Given a theory T, a set H of variables, and an atom p. Is there a (minimal) abductive explanation $E \subseteq H$ for p from T and H?
- The *relevance problem*: Given a theory T, a set H of variables, an atom p, and some hypothesis $h \in H$. Is there a (minimal) abductive explanation $E \subseteq H$ for p from T and H containing h?
- The *necessity problem*: Given a theory T, a set H of variables, an atom p, and some hypothesis $h \in H$. Does h occur in every (minimal) abductive explanation $E \subseteq H$ for p from T and H?

Autoepistemic logic. The language of Moore's autoepistemic logic (Moore 1985) contains the modal operator L, where $L\phi$ intuitively means that ϕ is believed. By $\mathcal{L}_L$ we denote the language $\mathcal{L}$ extended by L. Formulae $L\phi$ are viewed as propositional variables, which are called *modal atoms*.

A *stable expansion* of an autoepistemic theory $T \subseteq \mathcal{L}_L$ is a set $E \subseteq \mathcal{L}_L$ such that

$$E = Th(T \cup \{L\phi \mid \phi \in E) \cup \{\neg L\phi \mid \phi \notin E\}),$$

where $Th(\cdot)$ is the classical consequence operator with respect to the extended language $\mathcal{L}_L$.

QUIP handles the following tasks:

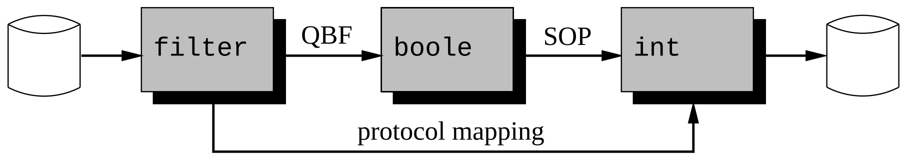

Figure 1: QUIP's system architecture.

- Given an autoepistemic theory $T \subseteq \mathcal{L}_L$, is there a stable expansion E of T?
- *Brave reasoning*: Given an autoepistemic theory $T \subseteq \mathcal{L}_L$ and some formula Φ, is there a stable expansion E of T containing Φ?
- *Skeptical reasoning*: Given an autoepistemic theory $T \subseteq \mathcal{L}_L$ and some formula Φ, is Φ contained in every stable expansion E of T?

For brave and skeptical reasoning, if the detail mode is engaged, QUIP returns *witnesses* corresponding to theses tasks: for brave reasoning, all stable expansions containing Φ are returned, whereas for skeptical reasoning, all stable expansions *not* containing Φ are returned.

Default logic. A *default theory* is a pair $\Delta = (T, D)$, where $T \subseteq \mathcal{L}$ is a set of formulae and D is a set of *defaults* of the form $\frac{\alpha\,:\,\beta}{\gamma}$.[2] Intuitively, the default is applied (γ is concluded) if α is provable and the *justification* β can be consistently assumed.

The semantics of $\Delta = (T, D)$ is defined in terms of *extensions* (Reiter 1980). Following (Marek & Truszczyński 1993), extensions can be characterized thus. For any $S \subseteq \mathcal{L}$, let $D(S)$ be the monotonic rules $\{\frac{\alpha}{\gamma} \mid \frac{\alpha\,:\,\beta}{\gamma} \in D, \neg\beta \notin S\}$. Then, $E \subseteq \mathcal{L}$ is an extension of Δ iff $E = Th^{D(E)}(T)$, where $Th^{D(E)}(T)$ is the set of all formulae derivable from T using classical logic together with the rules from $D(E)$.

QUIP expresses the following reasoning tasks:

- Given a default theory Δ, is there an extension E of Δ?
- *Brave reasoning*: Given a default theory Δ and some formula Φ, is there an extension E of Δ containing Φ?
- *Skeptical reasoning*: Given a default theory Δ and some formula Φ, is Φ contained in every extension E of Δ?

As for stable expansions, QUIP returns witnesses if the detail mode is engaged. Moreover, each of these reasoning tasks has been implemented in terms of two independent transformations: The first category of reductions is based on the characterization of extensions discussed above; the second category is based on a characterization of extensions using the notion of a *full set* (Niemelä 1995). Interestingly, although the transformations based on the latter method are more succinct than the corresponding reductions of the first kind, they have often an inferior performance compared to the former transformations.

[2]For simplicity, we omit multiple justifications here.

Disjunctive Logic Programming. A *disjunctive logic program*, Π, is a set of rules

$$r: \quad H(r) \leftarrow P(r), N(r)$$

where $H(r)$ is a disjunction of variables, $P(r)$ is a conjunction of variables, and $N(r)$ is a conjunction of negated variables. A Herbrand interpretation I of V is a *stable model* of Π (Gelfond & Lifschitz 1988; Przymusinski 1991), if it is a minimal model (with respect to set-inclusion) of the program Π^I resulting from Π as follows: remove each rule r such that $I \models a$ for some $\neg a$ in $N(r)$, and remove $N(r)$ from all remaining clauses.

Similar to autoepistemic logic and default logic, QUIP handles the problem whether a given logic program has a stable model, as well as brave and skeptical reasoning.

Circumscription. In contrast to the formalisms described above, propositional circumscription is already a quantified boolean formula, hence it does not require a separate reduction. So, QUIP can handle circumscription in a straightforward way.

In the propositional case, the parallel circumscription of a set of atoms $P = \{p_1, \ldots, p_n\}$ in a theory T, where the atoms Q are fixed and the remaining atoms $Z = \{z_1, \ldots, z_m\} = V \setminus (P \cup Q)$ may vary, is given by the following QBF $CIRC(T; P, Z)$, cf. (Lifschitz 1985):

$$T \wedge \forall P' \forall Z' \Big((T[P/P', Z/Z'] \wedge (P' \leq P)) \rightarrow (P \leq P') \Big).$$

Here, $P' = \{p'_1, \ldots, p'_n\}$ and $Z' = \{z'_1, \ldots, z'_m\}$ are sets of new propositional variables corresponding to P and Z, respectively, and $T[P/P', Z/Z']$ results from T by substitution of the variables in $P' \cup Z'$ for those in $P \cup Z$. QUIP implements circumscriptive inference of a formula ϕ from T, which is expressed by the QBF

$$\forall V (CIRC(T; P, Z) \rightarrow \phi).$$

Applying the System

Methodology

One of the basic motivations for the development of QUIP was to make a rapid prototyping tool available, aimed for experimenting with different knowledge-representation formalisms. Accordingly, QUIP is designed in such a way that the important reasoning tasks corresponding to the different formalisms under consideration can be directly encoded as queries. So, the methodology of specifying queries suitable to be processed by QUIP is the same as the methodology of formalizing a particular problem with one of the formalisms implemented in QUIP (subject to the restriction, of course, that QUIP currently accepts only queries specified over a propositional language; but in a future version it is planned to extend the language to allow function-free formulae with variables as well.)

Specifics

Let us illustrate how the queries of QUIP are structured. Generally, QUIP takes an input file as argument and writes the output to standard-out:

```
quip input_file
```

The input file comprises nonmonotonic theories and specifications of reasoning tasks. The format of the input uses two major concepts: *definitions* and *commands*. In the definitions, one can specify abductive theories, autoepistemic theories, default theories, logic programs, and propositional theories in general. The commands specify which reasoning tasks (with respect to the chosen formalism) have to be executed. Theories and formulas can be nested by using suitable names, so definitions can be edited rather conveniently. Further, commands can refer to specified theories without the need to describe them more than once. An input file can contain several definitions and commands, even referring to different formalisms.

In the following we describe some of these features using a simple example from default logic. Consider the following default theory $\Delta = (T, D)$, representing the well-known Nixon-diamond:

$$T = \{\mathit{Republican}, \mathit{Quaker}\};$$
$$D = \left\{ \frac{\mathit{Republican} : \neg \mathit{Pacifist}}{\neg \mathit{Pacifist}}, \frac{\mathit{Quaker} : \mathit{Pacifist}}{\mathit{Pacifist}} \right\}.$$

This default theory has two extensions:

$Th(\{\mathit{Republican}, \mathit{Quaker}\} \cup \{\neg \mathit{Pacifist}\})$;
$Th(\{\mathit{Republican}, \mathit{Quaker}\} \cup \{\mathit{Pacifist}\})$.

The extensions of this example can be computed by QUIP using the following input file, named `nixon`:

```
@SET DETAIL
   T :=   Republican & Quaker
@D D := { Republican : !Pacifist | !Pacifist,
          Quaker : Pacifist | Pacifist
        }

@DL ( T ; @D D )
```

Executing the command

```
quip nixon
```

results in the following output:

```
Th( {(Republican)&(Quaker)} u {(Pacifist)} )
Th( {(Republican)&(Quaker)} u {(!Pacifist)} )
```

The meaning of the different commands and definitions in the input file `nixon` can be explained as follows. First of all, the command `@SET DETAIL` invokes the detail mode, i.e., all extensions will be displayed. (Recall that QUIP admits the processing of both *decision problems* and *function problems.*) Choosing the command `@UNSET DETAIL` would have resulted in a simple yes/no answer.

The next tokens of the input file specify the constituents of the default theory Δ: `T` represents the background knowledge of Δ, and `D` contains the defaults. To avoid ambiguities, references to defaults always have to start with a special string "`@D`". The logical operators are represented in the obvious way ("`&`" denotes conjunction, "`!`" negation, and "`|`" separates a default's consequent from its justification).

The command `@DL` tells QUIP to compute the extensions of the specified default theory. To perform brave or skeptical reasoning, the input file `nixon` would be changed as follows:

```
@SET DETAIL
   T :=   Republican & Quaker
@D D := { Republican : !Pacifist | !Pacifist,
          Quaker : Pacifist | Pacifist
        }

@SET BRAVE
@DL ( W ; @D D ) |= Pacifist

@SET SKEPTICAL
@DL ( W ; @D D ) |= Pacifist
```

The `@SET`-commands switches between the two reasoning modes; "`|=`" is the symbol standing for the respective default consequence relations. The first command checks whether *Pacifist* is a brave consequence of Δ, the second commands checks skeptical consequence of *Pacifist* from Δ. Observe that it is permitted that an input file contains a sequence of reasoning tasks.

The output corresponding to the first command will be

```
Th( {(Republican)&(Quaker)} u {(Pacifist)} )
```

while the second command displays the extensions from which *Pacifist* does *not* follow:

```
Th( {(Republican)&(Quaker)} u {(!Pacifist)} )
```

Users and Usability

Obviously, proper usage of QUIP depends on a potential user's ability to express a given problem in terms of the implemented formalisms. However, a particular advantage of QUIP is that it incorporates several different knowledge representation formalisms. Hence, users have a *choice* selecting among different (yet closely related) approaches, singling out that particular formalism best suited for a specific purpose, or choosing a method based on a user's personal preference (e.g., because he/she understands that particular approach best). As well, the problem can be represented with different methods *simultaneously*, specifying the instances

k	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dlv	0.0	0.0	0.0	0.1	0.2	0.4	0.7	1.1	2.0	3.3	5.2	8.3	12.5	19.3
QUIP (DLP)	0.4	0.4	0.4	0.4	0.6	0.4	0.5	0.6	0.9	1.8	3.8	9.0	23.7	54.1
DeReS	0.0	0.2	1.7	21.2	67.9	–	–	–	–	–	–	–	–	–
QUIP (DL)	0.0	0.0	0.1	0.1	0.1	0.2	0.3	0.5	1.0	2.1	4.8	12.0	33.5	80.4
QUIP (ABD)	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.5	1.5	4.2	13.6	48.7	–

Table 1: Performance Results

of the resultant formalizations in a single input file, which can then be processed by QUIP requiring only one initial execution command.

Evaluating the System

Benchmarks

There are different approaches how systems handling knowledge representation formalisms can be evaluated. One is to perform comparisons taking into account the representational power of the implemented formalisms. That is to say, under such a comparison, one chooses some "natural" problems, encodes it with respect to the specific methodologies associated with the implemented formalisms, and uses the resultant instances as queries of the respective systems. So, basically, different systems are compared on the basis of (possibly) different representations *of the same problem*. However, to achieve a fair comparison, it is necessary that the experimenter is able to encode the given problem "in the best possible way" with respect to the particular formalisms. Also, incompatibilities of the underlying formalisms render comparisons between different systems often difficult.

Another possibility is to compare systems on problem classes *common* to each of the considered systems. Such a comparison is appropriate if different implementations of *the same formalism* are evaluated.

Since QUIP incorporates a wide array of different knowledge representation methods, comparisons with other implementations can in general be achieved employing the latter procedure.

Accepted benchmarks for nonmonotonic theorem provers have been realized by the well-known TheoryBase system (Cholewinski *et al.* 1995). This testbed provides encodings of various graph problems in terms of default theories or equivalent logic programs. However, the generated problems are at most NP-hard (or co-NP-hard, depending on the reasoning task), and thus do not take full advantage of the expressibility supported by the nonmonotonic formalisms. The only practically applied benchmark test utilizing a Σ_2^p-hard problem is the strategic companies example carried out for testing the system `dlv` (Eiter *et al.* 1998).

Here, we propose a straightforward method how Σ_2^p-hard benchmark problems for propositional nonmonotonic reasoning formalisms can be generated. The idea is to use the class of problems establishing the Σ_2^p-hardness of a formalism. Recall from our previous discussion that Σ_2^p-hardness of a nonmonotonic formalism is usually demonstrated by constructing a (polynomial) transformation $\mathcal{S}$ mapping instances I of $\textsc{qsat}_2$ (the evaluation problem of QBFs having quantifier order $\exists\forall$) into instances $\mathcal{S}(I)$ of the considered nonmonotonic reasoning task, $\textsc{nmp}$, such that I is a yes-instance of $\textsc{qsat}_2$ iff $\mathcal{S}(I)$ is a yes-instance of $\textsc{nmp}$. Thus, in some sense, the class of problems $\mathcal{S}(I)$ represents "worst-case" examples for the problem $\textsc{nmp}$, and therefore is particularly useful estimating the performance of a theorem prover solving the task $\textsc{nmp}$. Moreover, these problems are easily scalable by parameterizing different instances of $\textsc{qsat}_2$. An added feature of QUIP is that these examples provide at the same time a simple method for *testing* whether the implementation works correct, because QUIP turns instances $\mathcal{S}(I)$ of $\textsc{nmp}$ back to instances $\mathcal{T}(\mathcal{S}(I))$ of $\textsc{qsat}_2$, satisfying the condition that I is a yes-instance of $\textsc{qsat}_2$ iff $\mathcal{T}(\mathcal{S}(I))$ is a yes-instance of $\textsc{qsat}_2$. The next subsection describes comparisons between QUIP and some state-of-the-art provers on the basis of these benchmark problems.

Comparison

We compare the default-logic module of QUIP with DeReS (Cholewinski, Marek, & Truszczynski 1996) and the logic-programming module of QUIP with `dlv` (Eiter *et al.* 1998), using the class of examples discussed above. Space limits preclude a discussion on the structure of these problems; details can be found in the relevant literature (e.g., (Gottlob 1992; Eiter & Gottlob 1993; Eiter & Gottlob 1995a; Eiter & Gottlob 1995b)). We do not include a comparison with `smodels` (Niemelä & Simons 1996) here, because that system is currently not designed to handle Σ_2^p-problems (a comparison between QUIP, DeReS, `dlv`, `smodels`, and Theorist (Poole 1989), using examples from TheoryBase and some abduction problems, is given in (Egly *et al.* 2000b)).

Results of the comparison are given Table 1. All tests have been performed on a SUN ULTRA 60 with 256MB RAM; the run-time is measured in seconds with an upper limit of 90sec (i.e., instances requiring a longer period are not displayed). The first group of entries gives the results for the disjunctive logic programming test; the second group gives the results for the default logic test; and the final row contains some measurements using instances of the corresponding abductive problem class. The respective input-QBFs have been randomly generated and are parameterized by the number k of existential quantifiers (the number of variables was held fixed and was set to 20). Although the given results represent only a small sample, they do indicate that our *ad hoc* implementation performs sufficiently well.

Problem Size

It is rather obvious that QUIP cannot compete with state-of-the-art implementations like `dlv` or `smodels` in terms of problem size. These tools are highly optimized systems developed with a particular semantics in mind, whereas the purpose of QUIP is to provide a *uniform* method dealing with several knowledge representation tasks at the same time. Under this perspective, and taking into account that QUIP utilizes at present no optimizations whatsoever, our results demonstrate that implementing nonmonotonic reasoning formalisms using reductions to quantified boolean formulae is a feasible approach. Moreover, the modular architecture of QUIP allows an easy scalability and parallelization, by using, e.g., several QBF-provers simultaneously, each of which with its own optimization method.

References

[Bryant 1986] Bryant, R. E. 1986. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers* C-35(8):677–691.

[Cadoli, Giovanardi, & Schaerf 1998] Cadoli, M.; Giovanardi, A.; and Schaerf, M. 1998. An Algorithm to Evaluate Quantified Boolean Formulae. In *Proceedings AAAI-98*, 262–267. Menlo Park: AAAI Press.

[Cholewinski *et al.* 1995] Cholewinski, P.; Marek, V. W.; Mikitiuk, A.; and Truszcynski, M. 1995. Experimenting with Nonmonotonic Reasoning. In Sterling, L., ed., *Proceedings ICLP-95*, 267–282. Cambridge: MIT Press.

[Cholewinski, Marek, & Truszcynski 1996] Cholewinski, P.; Marek, V. W.; and Truszcynski, M. 1996. Default reasoning system DeReS. In *Proceedings KR-96*. 518–528.

[Egly *et al.* 2000a] Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000a. Implementing Default Reasoning Using Quantified Boolean Formulae. In *Proceedings 14. Workshop Logische Programmierung, Würzburg, Germany*, 223–225. GMD Report 90.

[Egly *et al.* 2000b] Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000b. Solving Advanced Resoning Tasks Using Quantified Boolean Formulae. Submitted.

[Eiter & Gottlob 1993] Eiter, T., and Gottlob, G. 1993. Propositional Circumscription and Extended Closed World Reasoning are Π_2^P-complete. *Journal of Theoretical Computer Science* 114(2):231–245, Addendum in vol. 118, p. 315, 1993.

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

[Eiter & Gottlob 1995b] Eiter, T., and Gottlob, G. 1995b. The Complexity of Logic-Based Abduction. *Journal of the Association of Computing Machinery* 42(1):3–42.

[Eiter *et al.* 1998] Eiter, T.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1998. The KR System dlv: Progress Report, Comparisons, and Benchmarks. In *Proceedings KR-98*, 406–417.

[Gelfond & Lifschitz 1988] Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, 1070–1080. Cambridge, Mass.: MIT Press.

[Giunchiglia & Sebastiani 1996] Giunchiglia, F., and Sebastiani, R. 1996. A SAT-based Decision Procedure for ALC. In *Proceedings KR-96*. San Francisco, California: Morgan Kaufmann. 304–314.

[Gottlob 1992] Gottlob, G. 1992. Complexity Results for Nonmonotonic Logics. *Journal of Logic and Computation* 2(3):397–425.

[Kautz & Selman 1996] Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. In *Proceedings AAAI-96*, 1194–1201. Menlo Park: AAAI Press / MIT Press.

[Lifschitz 1985] Lifschitz, V. 1985. Computing Circumscription. In *Proceedings IJCAI-85*, 121–127.

[Long 1998] Long, D. E. 1998. The Design of a Cache-Friendly BDD Library. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD-98)*, 639–645.

[Marek & Truszczyński 1993] Marek, W., and Truszczyński, M. 1993. *Nonmonotonic Logics – Context-Dependent Reasoning*. Springer.

[Moore 1985] Moore, R. 1985. Semantical Considerations on Nonmonotonic Logics. *Artificial Intelligence* 25:75–94.

[Niemelä & Simons 1996] Niemelä, I., and Simons, P. 1996. Efficient Implementation of the Well-founded and Stable Model Semantics. In *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, 289–303. Cambridge: MIT Press.

[Niemelä 1995] Niemelä, I. 1995. Towards Efficient Default Reasoning. In *Proceedings IJCAI-95*, 312–318.

[Poole 1989] Poole, D. 1989. Explanation and Prediction: An Architecture for Default and Abductive Reasoning. *Computational Intelligence* 5(1):97–110.

[Przymusinski 1991] Przymusinski, T. 1991. Stable Semantics for Disjunctive Programs. *New Generation Computing* 9:401–424.

[Reiter 1980] Reiter, R. 1980. A Logic for Default Reasoning. *Artificial Intelligence* 13:81–132.

[Rintanen 1999a] Rintanen, J. 1999a. Constructing Conditional Plans by a Theorem-Prover. *Journal of Artificial Intelligence Research* 10:323–352.

[Rintanen 1999b] Rintanen, J. 1999b. Improvements to the Evaluation of Quantified Boolean Formulae. In Dean, T., ed., *Proceedings IJCAI-99*, 1192–1197. Stockholm, Sweden: Morgan Kaufmann Publishers.

[Selman & Levesque 1990] Selman, B., and Levesque, H. J. 1990. Abductive and Default Reasoning: A Computational Core. In *Proceedings AAAI-90*, 343–348.