To appear on "Annals of Mathematics and Artificial Intelligence" special issue devoted to "Boundaries of Tractability for Artificial Intelligence"

On the Complexity of Entailment in Propositional Multivalued Logics

Marco Cadoli[*] Marco Schaerf[†]

Abstract

Multivalued logics have a long tradition in the philosophy and logic literature that originates from the work by Lukaszewicz in the 20's. More recently, many AI researchers have been interested in this topic for both semantic and computational reasons. Multivalued logics have indeed been frequently used both for their semantic properties and as tools for designing tractable reasoning systems.

We focus here on the computational aspects of multivalued logics. The main result of this paper is a detailed picture of the impact that the semantic definition, the syntactic form and the assumptions on the relative sizes of the inputs have on the complexity of entailment checking. In particular we show new polynomial cases and generalize polynomial cases already known in the literature for various popular multivalued logics. Such polynomial cases are obtained by means of two simple algorithms, sharing a common method.

[*]Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Roma, Italy. Tel. +39-6-49918326 Fax +39-6-85300849, E-mail: `cadoli@dis.uniroma1.it`

[†]Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Roma, Italy, Tel. +39-6-49918332 Fax +39-6-85300849, E-mail: `schaerf@dis.uniroma1.it`

1 Introduction

Multivalued logics (MV logics) have a long tradition in the philosophy and logic literature that originates from the seminal work by Łukaszewicz in the 20's. The main motivation for adopting an MV logic was the necessity to overcome some semantic limitations of classical logic. One of the drawbacks of classical logic that has been more frequently pointed out is that a generic unsatisfiable formula — e.g. $a \wedge \neg a$ — implies any formula b.

The work of Anderson and Belnap [1] in relevance logic has been one of the first attempts at defining a semantically well-founded logical system that does not have this undesired feature. As shown by Belnap in [2] their system can indeed be characterized by a multivalued semantics.

More recently many AI researchers have been interested in MV logics for several reasons. Starting from the work on relevance logics, Frisch introduced in [12, 13] a notion of weak inference for capturing the idea of *local* reasoning.

In the field of epistemic reasoning, the *impossible world* approach defined by Hintikka in [16] — i. e. each world is a multivalued truth assignment — is a common technique for dealing with the *logical omniscience* problem. An example of such an approach is reported by Fagin, Halpern and Vardi in [10].

MV logics have also been used as a uniform framework for representing sophisticated forms of reasoning like *default* reasoning and propositional attitudes. Relevant results along these lines have been obtained by Ginsberg in [15] and by Kifer & Lozinskii in [18].

In the field of Logic Programming, three-valued semantics have been frequently adopted for capturing the notion of *negation as failure*. In particular it is worth mentioning the *well-founded* semantics by Van Gelder, Ross and Schlipf [26], Fitting's *three-valued completion* [11] and Przymusinski's formalization of non-monotonic reasoning [25].

Apart from purely semantic issues, Levesque investigated about the possibility of using MV logics for capturing a computationally tractable form of reasoning. In his works [20, 21, 22] he proved that inference in some MV logics can be performed more easily than in classical logic.

Following these results, MV logics have been used as tools for designing tractable reasoning systems by Patel-Schneider in [23, 24] and by the present authors in [4, 5, 6]. The goal of systems of this kind is to provide a meaningful balance between complexity of inference and completeness of deduction. The gain of efficiency of these systems must therefore be compared to a general loss of inferential power.

In this paper we are interested in a detailed analysis of the computational aspect. The complexity of inference in MV logic depends on several distinct factors. First of all the complexity of reasoning is affected by the underlying semantics: inference in Levesque's 3-valued logic [22] is polynomial for classes of formulae in which inference in classical logic is intractable. Secondly, the syntactic form of the formulae involved also plays a role: reasoning in Levesque's 3-valued system is polynomial if the formulae are in conjunctive normal form, while it is co-NP-complete if no normal form is assumed. Finally, the way the inference task is defined is also important. As a matter of fact, given a multivalued definition of models, let's say $*$-models, the fact that a formula β follows from a Knowledge Base (KB) α can be defined in three ways:

1. all the $*$-models of α are $*$-models of β;

2. $\alpha \to \beta$ is $*$-valid, i. e. true in all $*$-models;

3. $\alpha \wedge \neg\beta$ has no $*$-models.

While the three definitions are equivalent for classical models, they differ for example in Levesque's 3-valued logic [22]. The three definitions lead also to problems of different computational complexity.

The goal of this paper is to understand the tractability threshold of the entailment checking problem for various MV propositional logics. We take into account logical consequence as defined in point 1 above, but the results we obtain are readily usable for characterizing the complexity of the other definitions as well.

We focus on a specific class of MV logics, that is the four-or-less valued logics that admit at least the truth values `true` and `false`. In particular we take into account classical 2-valued logic, Levesque's 3-valued logic, Belnap's 4-valued logic and the 3-valued logic underlying well-founded semantics.

As for the syntactic classes of propositional formulae, we consider three possibilities: formulae in conjunctive normal form, in disjunctive normal form and in arbitrary form.

In our complexity analysis we also take into account the relative sizes of the KB and the query. Our motivation is in that in many practical cases the size of the query is much smaller than the size of the KB (cf. also Vardi in [27]) This assumption leads to the analysis of the *data complexity*, i. e. complexity wrt the size of the KB, considering the size of the query to be a constant. The dual case, when the size of the KB is assumed to be much smaller than that of the query (*expression complexity*) might be interesting when dealing with mathematical theories in which the size of the KB is very small and the query can be very complex. When no specific assumption is made, one deals with the *combined complexity.*

The main result of this paper is a detailed picture of the impact that the semantic definition, the syntactic form and the assumptions on the relative sizes of the inputs have on the complexity of entailment checking. In particular we show new polynomial cases and generalize polynomial cases already known in the literature. One of the distinctive features of our results is in that all polynomial cases (both the new ones and those already known) are obtained by means of two simple algorithms, sharing a common method.

The MV logics we consider in this work have already been used as the basis for more complex systems, such as the four-valued terminological logic [23] and the four-valued (decidable) first-order logic [24] presented by Patel-Schneider. Simple propositional generalizations are also used in the approximation schemata introduced by the present authors in [4, 5, 6]. The results shown in this paper can be used for relaxing syntactic restrictions adopted in those systems in order to ensure tractability.

The structure of the paper is as follows: Section 2 contains some preliminary definitions. Sections 3 and 4 contain the complexity analysis, that is discussed in Section 5. Section 6 contains some conclusions.

2 Definitions

The relevant definitions about propositional multivalued logics are introduced.

Let $L = \{l_1, \ldots, l_n\}$ be a finite set of propositional letters. A literal is a letter l of L or its negation $\neg l$. The symbol L^* denotes the set of all literals associated with the letters of L plus the two special atoms $\top$ (top) and $\bot$ (bottom). Propositional well-formed formulae (wffs) are built up from the set L^* using the logical connectives $\vee, \wedge, \neg, \rightarrow$, in the usual way.

We are interested in three normal forms for propositional formulae. The first is conjunctive normal form (CNF), which consists of a single conjunction of clauses, where a clause is a disjunction of literals. The second is disjunctive normal form (DNF), which consists of a single disjunction of conjunctions of literals. Finally formulae in NNF (Negation Normal Form) are considered; they are built up from literals, conjunctions and disjunctions. In other words in NNF formulae negation only occurs in front of letters. Notice that a formula in CNF or DNF is also in NNF.

Let α be a formula in NNF. In the following $[\alpha]$ denotes the set of all the literals (excluding $\top$ and $\bot$) occurring in it. Given a set of literals S, $\overline{S}$ denotes the set of literals which are the complement of literals in S but do not belong to S.

Example 2.1 Let α be the formula $(a \vee \neg a \vee b)$, $[\alpha]$ is the set $\{a, \neg a, b\}$ and $\overline{[\alpha]}$ is the set $\{\neg b\}$.

Let β be the formula $((a \vee \bot) \wedge (\neg a \vee b) \wedge \top)$, $[\beta]$ is the set $\{a, \neg a, b\}$ and $\overline{[\beta]}$ is the set $\{\neg b\}$. $\diamond$

In the following subsections we give definitions for three different aspects of MV logics: semantics, reduction to normal forms and reasoning problems.

2.1 Semantics

A truth assignment is a function mapping the set of literals L^* into the set $\{0, 1\}$, where $\top$ and $\neg\bot$ are always mapped into 1, while $\bot$ and $\neg\top$ into 0.

Four different forms of truth assignment are now introduced. They are characterized by different restrictions on the mapping.

Definition 2.1 (2-, 3-, 3*-, 4-interpretation)

- *A 2-interpretation of L^* is a truth assignment which maps every letter l of L and its negation $\neg l$ into opposite values;*
- *A 3-interpretation of L^* is a truth assignment which does not map both a letter l of L and its negation $\neg l$ into 0;*

- *A 3^*-interpretation of L^* is a truth assignment which does not map both a letter l of L and its negation $\neg l$ into 1;*
- *A 4-interpretation of L^* is a truth assignment.*

The following table summarizes the four different possibilities that may occur for a generic letter l and its negation $\neg l$ in the various forms of interpretation introduced in the above definition.

2	3	3^*	4	l	$\neg l$	
$\surd$	$\surd$	$\surd$	$\surd$	1	0	`true`
$\surd$	$\surd$	$\surd$	$\surd$	0	1	`false`
	$\surd$		$\surd$	1	1	`both`
		$\surd$	$\surd$	0	0	`none`

Table 1: Various forms of interpretation

The table shows that for 2-interpretations only two possibilities exist, since a letter l and its negation $\neg l$ are always mapped into opposite values. Such possibilities are traditionally referred to by saying that l is either `true` or `false`. Both in 3-interpretations and in 3^*-interpretations a third possibility exists. In 3-interpretations both a letter l and its negation $\neg l$ might be mapped into 1. This situation is sometimes referred (cf. [2]) by saying that the truth value of l is `both`. In 3^*-interpretations the dual situation – where l and $\neg l$ are mapped into 0 – is admitted. The truth value of l is sometimes referred to as `none`. Finally, 4-interpretations admit all four possibilities. Notice that every 2-interpretation is also a 3- and a 3^*-interpretation, while these are always 4-interpretations.

Denoting with I a (2-, 3-, 3^*- or 4-) interpretation, satisfaction of formulae is defined as follows (l is a propositional letter and A, B are propositional wffs):

- $I \models \top$;
- $I \not\models \bot$;
- $I \models l$ iff $I(l) = 1$;

- $I \models \neg l$ iff $I(\neg l) = 1$;
- $I \models A \wedge B$ iff $I \models A$ and $I \models B$;
- $I \models A \vee B$ iff $I \models A$ or $I \models B$;
- $I \models \neg(A \wedge B)$ iff $I \models \neg A$ or $I \models \neg B$;
- $I \models \neg(A \vee B)$ iff $I \models \neg A$ and $I \models \neg B$;
- $I \models \neg\neg(A)$ iff $I \models A$;
- $I \models A \rightarrow B$ iff $I \models \neg A \vee B$.

A formula α is x-satisfiable ($x = 2, 3, 3^*$ or 4) iff there exists an x-interpretation I such that $I \models \alpha$. Such an x-interpretation I is called an x-model of α.

Example 2.2 Let α be $a \wedge (a \rightarrow b) \wedge \neg b$. The formula α is 2-unsatisfiable as well as 3^*-unsatisfiable, while it is both 3- and 4-satisfiable. A 3-interpretation (hence also 4-interpretation) satisfying α is $I(a) = I(\neg a) = I(b) = I(\neg b) = 1$. $\diamond$

Notice that x-unsatisfiable formulae do exist for any kind of satisfiability.

Example 2.3 The formula $\perp \wedge a$ is 4-unsatisfiable (as well as 3^*-, 3- and 2-unsatisfiable). $\diamond$

However, NNF formulae containing neither $\perp$ nor $\neg\top$ are always both 3- and 4-satisfiable.

A formula T_1 is equivalent to a formula T_2 iff they have the same 4-models. Clearly equivalent formulae also have the same 2-models, the same 3-models and the same 3^*-models.

Four different forms of logical entailment are now introduced. Each of them corresponds to a definition of interpretation.

Definition 2.2 (2-, 3-, 3^*-, 4-entailment)

- *T 2-entails q ($T \models^2 q$) iff every 2-interpretation satisfying T also satisfies q;*

- *T 3-entails q ($T \models^3 q$) iff every 3-interpretation satisfying T also satisfies q;*
- *T 3*-entails q ($T \models^{3^*} q$) iff every 3*-interpretation satisfying T also satisfies q;*
- *T 4-entails q ($T \models^4 q$) iff every 4-interpretation satisfying T also satisfies q.*

The relation $\models^2$ obviously corresponds to the entailment relation of classical propositional logic, while $\models^4$ has been introduced by Belnap in [2] as a semantic account of tautological entailment as defined in [1]. More recently, Levesque [22] defined $\models^3$ as a refinement of $\models^4$. The remaining relation $\models^{3^*}$ corresponds to the underlying semantics of many 3-valued systems where the third truth value is intended as neither true nor false. This interpretation has been frequently used in many semantics for negation in logic programming, such as the well-founded semantics [26]. The symbols we use are however original and do not correspond to those used by other authors.

These entailment relations do not have all the properties classical entailment has. In fact, while $\models^2$ satisfies contraposition, i. e. $T \models^2 q$ iff $\neg q \models^2 \neg T$, this property does not hold for $\models^3$: as an example $b \models^3 a \vee \neg a$, but $\neg a \wedge a \not\models^3 \neg b$. Similarly, contraposition does not hold for $\models^{3^*}$ while it holds for $\models^4$.

2.2 Normal forms

Let $NNF()$, $CNF()$ and $DNF()$ be three functions that take a wff α and return an equivalent wff α' in NNF, CNF or DNF, respectively. The function $NNF()$ can be implemented in linear time by repeatedly applying the following well-known rewriting rules:

$$\begin{aligned}
\neg(A \wedge B) &\mapsto \neg A \vee \neg B \\
\neg(A \vee B) &\mapsto \neg A \wedge \neg B \\
\neg\neg(A) &\mapsto A \\
(A \rightarrow B) &\mapsto (\neg A \vee B)
\end{aligned}$$

By the definition of satisfaction given previously it immediately follows that the above set of rewriting rules preserves the semantics. More precisely, for a generic formula α the set of x-models of α and the set of x-models of $NNF(\alpha)$ are the same, for $x = 2$, 3, 3^* and 4.

As for the CNF and the DNF of a formula α, they can be obtained in the following way:

1. compute $NNF(\alpha)$;

2. (CNF) repeatedly apply the rewriting rule:

$$A \vee (B \wedge C) \mapsto (A \vee B) \wedge (A \vee C).$$

2. (DNF) repeatedly apply the rewriting rule:

$$A \wedge (B \vee C) \mapsto (A \wedge B) \vee (A \wedge C).$$

The above method preserves the semantics, since it rewrites a formula into an equivalent one. As far as its computational cost is concerned, it might take $O(2^{|\alpha|})$ time. Moreover, the formula we end up with might have $O(2^{|\alpha|})$ length. According to studies in the theory of circuit complexity (see [3] for a comprehensive survey), this exponential increase does not seem to be avoidable. In the following, $CNF(\alpha)$ and $DNF(\alpha)$ denote the formulae obtained using the above method.

2.3 Decision problems

Our goal in this paper is to identify the tractability threshold of the problem of testing whether $T \models^x q$, where T and q are propositional formulae and $\models^x$ is one of the four entailment relations introduced in Definition 2.2. We now show — by means of examples — that the complexity of such a problem is affected by three distinct factors:

entailment relation

- $T \models^2 q$ is co-NP-complete if T, q are in CNF;
- $T \models^4 q$ is polynomial if T, q are in CNF [20];

syntactic form

- $T \models^3 q$ is polynomial if T, q are in CNF [22];
- $T \models^3 q$ is co-NP-complete if T, q are general propositional formulae;

what is the input of the problem

- $T \models^2 q$ is co-NP-complete if $\langle T, q \rangle$ is the input;
- $T \models^2 q$ is polynomial if q is the input (in CNF) and the size of T is fixed.

In the following we focus on the analysis of the computational complexity of all forms of entailment under various assumptions, thus capturing the different combinations of the above factors. In particular we analyze the entailment problems with respect to three different notions of complexity defined by Vardi in [27]:

DATA complexity

T	is the input;
q	has fixed size;

EXPRESSION complexity

q	is the input;
T	has fixed size;

COMBINED complexity

T, q	are the input.

In fact, Vardi defines data complexity for fixed queries (not fixed size) and dually defines expression complexity for fixed KBs. Anyway our results hold for the more general definition.

As noticed previously, the complexity of computing $CNF(\alpha)$ is exponential in the size of α if α is not already in CNF. Nevertheless, while this operation is exponential in the size of the formula, it may not be so in the size of the input. As an example, if one is dealing with data complexity, computing $CNF(q)$ takes constant time. A similar argument holds for DNF formulae.

The complexity analysis of the problem of testing whether $T \models^x q$ is carried out for T and q in CNF, DNF or with no syntactic restrictions. Summing up, we are considering four different logics, three different syntactic forms, both for the KB T and the query q, and three complexity definitions. In total we have one hundred and eight decision problems.

All of these decision problems belong to the class co-NP (cf. [14]). As a matter of fact the complementary problem of testing whether $T \not\models^x q$ holds belongs to the class NP, as the following polynomial non-deterministic procedure shows: guess non-deterministically an x-interpretation I and return **false** if $I \models T$ and $I \not\models q$. Obviously model checking can be done in deterministic linear time. The non-deterministic procedure is shown in Figure 1.

```
Algorithm Entails
(* Input two formulae $T, q$ *)
(* Output true, if $T \models^x q$, false otherwise *)
begin
  guess an $x$-interpretation $I$ of $T$;
  if $I \models T$ and $I \not\models q$
    then return false
  end;
  return true;
end.
```

Figure 1: The algorithm *Entails*

3 Complexity analysis: known results

This section contains a survey of some computational aspects of the MV logics defined in the previous section.

3.1 Polynomial reductions

The entailment problems we defined are not independent on each other. In fact several reductions among the four entailment relations can be easily obtained from the definitions (T and q are generic wffs; remind that the alphabet is $L = \{l_1, \ldots, l_n\}$):

Red-1 $T \models^2 q$ iff $(l_1 \vee \neg l_1) \wedge \cdots \wedge (l_n \vee \neg l_n) \wedge T \models^4 q \vee (l_1 \wedge \neg l_1) \vee \cdots \vee (l_n \wedge \neg l_n)$.

Red-2 $T \models^2 q$ iff $T \models^3 q \vee (l_1 \wedge \neg l_1) \vee \cdots \vee (l_n \wedge \neg l_n)$.

Red-3 $T \models^3 q$ iff $(l_1 \vee \neg l_1) \wedge \cdots \wedge (l_n \vee \neg l_n) \wedge T \models^4 q$.

Red-4 $T \models^2 q$ iff $(l_1 \vee \neg l_1) \wedge \cdots \wedge (l_n \vee \neg l_n) \wedge T \models^{3^*} q$.

Red-5 $T \models^{3^*} q$ iff $T \models^4 q \vee (l_1 \wedge \neg l_1) \vee \cdots \vee (l_n \wedge \neg l_n)$.

Examples illustrating how the above reductions can be used for obtaining complexity results will be shown in the following.

It is well known that the complexity of the entailment relation strongly depends on the complexity of two other operations, namely satisfiability and validity. The following reductions map an unsatisfiability or validity checking task into an entailment one.

Red-6 T is x-unsatisfiable iff $T \models^x \bot$;

Red-7 q is x-valid iff $\top \models^x q$.

3.2 Complexity of satisfiability and validity testing

Let us review the complexity of checking x-satisfiability ($x = 2, 3, 3^*, 4$) of formulae. First of all, it is well known [7] that:

Sat-1 2-satisfiability of a formula α can be checked in polynomial time if $DNF(\alpha)$ can be computed in polynomial time. Checking 2-satisfiability of a formula in CNF is NP-complete.

As far as the other kinds of interpretations are concerned, it is useful to introduce an ordering between truth assignments.

Definition 3.1 (ordering in the set of truth assignments) *Let I_1 and I_2 be two truth assignments. I_1 is less than or equal to I_2 (written $I_1 \leq I_2$) iff for each $l \in L^*$, $I_1(l) \leq I_2(l)$ holds. The relation $\geq$ is defined as its inverse, i. e. $I_2 \geq I_1$ iff $I_1 \leq I_2$.*

The following lemma states that this ordering has one important property, namely monotonicity.

Lemma 3.1 (monotonicity) *Let I be a truth assignment and α a formula. If $I \models \alpha$ holds then $\forall I'.I \leq I'$ implies $I' \models \alpha$. If $I \not\models \alpha$ holds then $\forall I'.I \geq I'$ implies $I' \not\models \alpha$.*

PROOF: Assume that $I_1 \leq I_2$ and there exists a formula α s. t. $I_1 \models \alpha$ and $I_2 \not\models \alpha$. Since satisfaction of a formula is always reduced to satisfaction of literals (see the rules for satisfaction in Section 2.1), then there must exist a literal l s. t. $I_1(l) = 1$ and $I_2(l) = 0$. But this contradicts the hypotheses. $\diamond$

By Lemma 3.1 it follows that a formula is x-satisfiable iff there exists a maximal (w. r. t. $\geq$) x-interpretation satisfying it. Since 3- and 4-interpretations have a unique maximal interpretation — namely the interpretation I_{max} mapping all literals into 1 — a formula α is 3- and 4-satisfiable iff $I_{max} \models \alpha$, and this can be checked in $O(|\alpha|)$ time. On the other hand, since the maximal 3^*-models of α are exactly its 2-models, deciding 3^*-satisfiability is equivalent to deciding 2-satisfiability. These observations are summarized in the following list:

Sat-2 3-satisfiability of a formula α can always be checked in $O(|\alpha|)$ time.

Sat-3 3^*-satisfiability of a formula α can be checked in polynomial time iff 2-satisfiability of α can be checked in polynomial time.

Sat-4 4-satisfiability of a formula α can always be checked in $O(|\alpha|)$ time.

Let us turn our attention to the complexity of checking x-validity ($x = 2, 3, 3^*, 4$) of formulae. First of all, it is well known that:

Val-1 2-validity of a formula α can be checked in polynomial time if $CNF(\alpha)$ can be computed in polynomial time. Checking 2-validity of a formula in DNF is co-NP-complete.

By Lemma 3.1 it follows that a formula is x-valid iff all the minimal (w. r. t. $\leq$) x-interpretations satisfy it. Since 3^*- and 4-interpretations have a unique minimal interpretation — namely the interpretation I_{min} mapping all literals into 0 — a formula α is 3^*- and 4-valid iff $I_{min} \models \alpha$, and this can be checked in $O(|\alpha|)$ time. On the other hand, since the minimal 3-models of α are exactly its 2-models, deciding 3-validity is equivalent to deciding 2-validity. These observations are summarized in the following list:

Val-2 3-validity of a formula α can be checked in polynomial time iff 2-validity of α can be checked in polynomial time.

Val-3 3^*-validity of a formula α can always be checked in $O(|\alpha|)$ time.

Val-4 4-validity of a formula α can always be checked in $O(|\alpha|)$ time.

3.3 Known computational results

Returning to our original goal of analyzing the complexity of checking $T \models^x q$, we show how to use the results of the previous subsections in order to prove intractability results.

The fact **Sat-1** can be used along with the reductions we listed to obtain intractability results, as shown by the following example.

Example 3.1 (proving intractability results) From **Red-6** and **Red-2** it follows that $T \not\models^3 (l_1 \wedge \neg l_1) \vee \cdots \vee (l_n \wedge \neg l_n)$ iff T is 2-satisfiable. As a consequence of **Sat-1**, the combined complexity of testing $T \models^3 q$ is co-NP-hard if T is in CNF and q is in DNF.

In a similar way, using **Red-6** and **Red-4** we prove that $(l_1 \vee \neg l_1) \wedge \cdots \wedge (l_n \vee \neg l_n) \wedge T \not\models^{3^*} \bot$ iff T is 2-satisfiable. As a consequence of **Sat-1**, the data complexity of testing $T \models^{3^*} q$ is co-NP-hard if T is in CNF. $\diamond$

In the same style used in the above example, **Val-1** can be used along with reductions **Red-1**, ..., **Red-7** to obtain intractability results.

Let us turn our attention to the polynomial cases. As for $T \models^2 q$, the following well-known fact can be used as a basis for an algorithm.

Observation 3.2 (testing $T \models^2 q$) *$T \models^2 q$ iff for each clause c of $CNF(q)$ and for each conjunction d of $DNF(T)$ it holds that $d \models^2 c$,* i.e. *at least one of the following conditions holds:*

1. *c is 2-valid;*
2. *d is 2-unsatisfiable;*
3. *$([d] \cap [c]) \neq \emptyset$.*

An algorithm based on Observation 3.2 decides $T \models^2 q$ in polynomial time iff both $CNF(q)$ and $DNF(T)$ can be computed in polynomial time.

As for the relations $\models^3$ and $\models^4$, a polynomial case has been shown by Levesque in [20, 22] using the following observation:

Observation 3.3 (testing $T \models^4 q$ and $T \models^3 q$) *$T \models^4 q$ iff for each clause c_1 of $CNF(q)$ there exists a clause c_2 of $CNF(T)$ such that $[c_1] \supseteq [c_2]$.*

$T \models^3 q$ iff for each clause c_1 of $CNF(q)$ at least one of the following conditions holds:

1. *c_1 contains a pair of complementary literals;*
2. *there exists a clause c_2 of $CNF(T)$ such that $[c_1] \supseteq [c_2]$.*

This guarantees that both $T \models^3 q$ and $T \models^4 q$ can be decided in $O(|T| \cdot |q|)$ time if both $CNF(q)$ and $CNF(T)$ can be computed in linear time.

Using all the above observations and reductions, Table 2 can be filled. The table shows the (combined, data and expression) complexity of deciding $T \models^x q$. The symbol **Any** means that no assumption is made on the syntactic form of the formula. The symbol **P** means that the relation can be decided in polynomial time, while **coNP** means that the problem is co-NP-complete. The symbol **?** means that the complexity of the corresponding problem cannot be determined by using the observations and reductions shown in this section.

$T \models^x q$		COMBINED				DATA				EXPRESSION			
T	q	2	3	3*	4	2	3	3*	4	2	3	3*	4
Any	Any	coNP	coNP	coNP	coNP	coNP	?	coNP	?	coNP	coNP	?	?
Any	CNF	coNP	?	coNP	?	coNP	?	coNP	?	P	P	?	P
Any	DNF	coNP	coNP	coNP	coNP	coNP	?	coNP	?	coNP	coNP	?	?
CNF	Any	coNP	coNP	coNP	coNP	coNP	P	coNP	P	coNP	coNP	?	?
CNF	CNF	coNP	P	coNP	P	coNP	P	coNP	P	P	P	?	P
CNF	DNF	coNP	coNP	coNP	coNP	coNP	P	coNP	P	coNP	coNP	?	?
DNF	Any	coNP	coNP	?	?	P	?	?	?	coNP	coNP	?	?
DNF	CNF	P	?	?	?	P	?	?	?	P	P	?	P
DNF	DNF	coNP	coNP	?	?	P	?	?	?	coNP	coNP	?	?

Table 2: Known results on the complexity of checking $T \models^x q$

4 Complexity analysis: new results

We now develop a new method for testing entailment in propositional multivalued logics, thus providing a polynomial-time algorithm for all the cases whose complexity has not been settled according to Table 2. The method is based on a two-step procedure:

1. simplify syntactically the KB and the query, thus obtaining a new formula W;
2. test validity or satisfiability of W.

First of all we specify the first step of the above procedure. To this end the function *Simplify* — described in Figure 2 — is introduced. *Simplify* performs a partial evaluation of a formula, deleting all the literals occurring in two sets.

```
Function $Simplify(\alpha, S_p, S_n)$ : formula;
(* Input a formula $\alpha$ and two disjoint sets of literals $S_p$ and $S_n$;
        $\top$ and $\bot$ occur neither in $S_p$ nor in $S_n$. *)
(* Output a formula in NNF where all the literals in $S_p$
          are replaced by $\top$ and the literals in $S_n$ by $\bot$. *)
```

```
begin
  $Temp := NNF(\alpha)$;
  for each $l_1 \in S_p$ do
        replace every occurrence of $l_1$ in $Temp$ by $\top$;
  for each $l_2 \in S_n$ do
        replace every occurrence of $l_2$ in $Temp$ by $\bot$;
  $Simplify := Temp$;
end.
```

Figure 2: The function *Simplify*

Notice that the function *Simplify* can be computed in $O(|\alpha| \cdot |S_p \cup S_n|)$ time. Moreover if α is in CNF (DNF), then also its simplified form is in CNF (DNF).

The function *Simplify* is now illustrated by means of two examples.

Example 4.1

$$\begin{aligned} T &= (a \wedge \neg(\neg b \wedge d)) \vee \neg(b \vee (c \wedge d)) \\ S_p &= \{\neg a, b\} \\ S_n &= \{a, \neg b\}. \end{aligned}$$

The negation normal form of T is:

$$NNF(T) = (a \wedge (b \vee \neg d)) \vee (\neg b \wedge (\neg c \vee \neg d)).$$

The simplification yields:

$$Simplify(T, S_p, S_n) = (\bot \wedge (\top \vee \neg d)) \vee (\bot \wedge (\neg c \vee \neg d))$$

$\diamond$

Example 4.2

$$\begin{aligned} q &= (a \vee (b \wedge \neg c)) \rightarrow \neg(a \vee c) \\ S_p &= \{\neg a, b\} \\ S_n &= \{a, \neg b\} \\ NNF(q) &= (\neg a \wedge (\neg b \vee c)) \vee (\neg a \wedge \neg c) \\ Simplify(q, S_p, S_n) &= (\top \wedge (\bot \vee c)) \vee (\top \wedge \neg c) \end{aligned}$$

$\diamond$

The function *Simplify* yields two possible methods for deciding the entailment relations. The first method is based on simplifying the KB T and testing the satisfiability of the resulting formula, as specified by the following theorem.

Theorem 4.1 (testing $T \models^x q$ #1) *$T \models^x q$ iff for each clause c of $CNF(q)$ one of the following conditions holds:*

1. *c is x-valid*
2. *$Simplify(T, \overline{[c]}, [c])$ is x-unsatisfiable.*

PROOF: It is an immediate consequence of the semantics that if γ is in CNF then $T \models^x \gamma$ holds iff for each clause c of γ it is the case that $T \models^x c$. In order to show that $T \models^x c$ holds iff c is x-valid or $Simplify(T, \overline{[c]}, [c])$ is x-unsatisfiable we use the following lemma:

Lemma 4.2 *Let T be a formula, c a clause which is not x-valid, C the set of letters occurring in c and I_c the x-interpretation of the alphabet C that maps all the literals of $[c]$ into 0 and the literals of $\overline{[c]}$ into 1. An x-interpretation $I_{L \setminus C}$ of the alphabet $L \setminus C$ satisfies $Simplify(T, \overline{[c]}, [c])$ iff $I = (I_{L \setminus C} \cup I_c)$ satisfies T.*

PROOF: First of all, we remind that the two sets $[c]$ and $\overline{[c]}$ are disjoint by definition and that they do not contain $\top$, $\bot$. All the occurrences of literals of $[c]$ in T have been replaced by $\bot$ in $Simplify(T, \overline{[c]}, [c])$ and all the literals of $\overline{[c]}$ have been replaced by $\top$, which are always mapped into 0 and 1, respectively. But this is exactly the same value they are mapped into by I_c, hence the x-interpretation $I_{L \setminus C}$ satisfies $Simplify(T, \overline{[c]}, [c])$ iff I satisfies T. $\diamond$

(only if) Assume that $T \models^x c$ holds, c is not x-valid and $Simplify(T, \overline{[c]}, [c])$ is x-satisfiable. We show that a contradiction follows. Let C be the set of letters occurring in c and let I_1 be an x-interpretation of the alphabet $L \setminus C$ satisfying $Simplify(T, \overline{[c]}, [c])$. It is always possible to find such an I_1 since $Simplify(T, \overline{[c]}, [c])$ contains no letters of C. Let I_2 be the interpretation of the alphabet C that maps all the literals of $[c]$ into 0 and all the literals of $\overline{[c]}$ into 1. It is clearly the case that $I_2 \not\models c$, we now show that I_2 is an x-interpretation by a simple case analysis. If $x = 2$ or 3, since c is not x-valid it cannot contain complementary literals, hence if $l \in [c]$ then its complementary literal belongs to $\overline{[c]}$ therefore I_2 is both a 2- and a 3-interpretation of the alphabet C because it maps complementary literals into opposite values. If $x = 3^*$ or 4, since for each letter $l \in C$ either $l \in [c]$ or $\neg l \in [c]$ we have that I_2 maps at least one literal of each pair of complementary literals into 0, hence it is both a 3*- and a 4-interpretation of the alphabet C. Now, let I be an interpretation of the alphabet L which for all literals of $L \setminus C$ it coincides with I_1 and for all literals of C it coincides with I_2. Obviously, I is an x-interpretation, $I \not\models c$ and $I \models \mathit{Simplify}(T, \overline{[c]}, [c])$. But by Lemma 4.2, I also satisfies T which implies $T \not\models^x c$, hence contradiction.

(if) If c is x-valid it is obviously the case that $T \models^x c$, hence we assume that c is not x-valid and $\mathit{Simplify}(T, \overline{[c]}, [c])$ is x-unsatisfiable but $T \not\models^x c$. Let I be an x-interpretation of the alphabet L which satisfies T but not c. We show that a contradiction follows. Since I does not satisfy c it must map all of its literals into 0. Let I' be another x-interpretation which agrees with I for all literals not in $[c] \cup \overline{[c]}$, maps all the literals of $[c]$ into 0 and all the literals of $\overline{[c]}$ into 1. It is clearly the case that $I' \geq I$, hence by Lemma 3.1 we have that $I' \models T$. But by Lemma 4.2, I' also satisfies $\mathit{Simplify}(T, \overline{[c]}, [c])$, hence contradiction. $\diamond$

This theorem tells us that whenever we can put q in CNF in polynomial time and we can decide x-satisfiability in polynomial time then we have a polynomial algorithm for deciding $T \models^x q$ (the cost of testing x-validity of a clause is negligible w. r. t. the other costs). More precisely, let us denote with Γ the number of clauses occurring in $CNF(q)$, with Ξ the time complexity of simplifying T with each such clause, with Θ_x the time occurring to test x-satisfiability of each simplified formula. The above theorem states that $T \models^x q$ can be tested in $O((\Xi + \Theta_x) \cdot \Gamma)$ time. As an example, if q is already in CNF and we are dealing with the combined complexity of 3-entailment, the test can be done in $O(|T| \cdot |q|)$ time. The same upper bound holds for all the polynomial cases.

Let us give a list of all the polynomial cases we can obtain using the above result along with **Sat-1**, **Sat-2**, **Sat-3** and **Sat-4**:

- $T \models^2 q$ can be decided in polynomial time if both:
  1. $CNF(q)$ can be computed in polynomial time **and**
  2. $DNF(T)$ can be computed in polynomial time.
- $T \models^3 q$ can be decided in polynomial time if $CNF(q)$ can be computed in polynomial time.
- $T \models^{3^*} q$ can be decided in polynomial time if both:
  1. $CNF(q)$ can be computed in polynomial time **and**
  2. $DNF(T)$ can be computed in polynomial time.
- $T \models^4 q$ can be decided in polynomial time if $CNF(q)$ can be computed in polynomial time.

The following example — which continues Example 4.1 — shows how Theorem 4.1 applies.

Example 4.3 (query in CNF) As in Example 4.1, T is $(a \wedge \neg(\neg b \wedge d)) \vee \neg(b \vee (c \wedge d))$. Let q be $a \vee \neg b$. The simplification of the KB T with respect to q works as follows:

$$\begin{array}{rcl} [q] & = & \{a, \neg b\} \\ \overline{[q]} & = & \{\neg a, b\} \\ Simplify(T, \overline{[q]}, [q]) & = & (\bot \wedge (\top \vee \neg d)) \vee (\bot \wedge (\neg c \vee \neg d)) \end{array}$$

Notice that $Simplify(T, \overline{[q]}, [q])$ is x-unsatisfiable for $x = 2, 3, 3^*, 4$, hence $T \models^x q$ holds for $x = 2, 3, 3^*, 4$. $\diamond$

The second method for deciding the entailment relations is based on simplifying the query and deciding the validity of the resulting formula, as specified by the following theorem.

Theorem 4.3 (testing $T \models^x q$ #2) *$T \models^x q$ iff for each conjunction d of $DNF(T)$ one of the following conditions holds:*

1. d is x-unsatisfiable

2. $Simplify(q, [d], \overline{[d]})$ is x-valid.

PROOF: It is an immediate consequence of the semantics that if Σ is in DNF then $\Sigma \models^x q$ holds iff for each conjunction d of Σ it is the case that $d \models^x q$. In order to show that $d \models^x q$ holds iff d is x-unsatisfiable or $Simplify(q, [d], \overline{[d]})$ is x-valid we will use the following lemma:

Lemma 4.4 *Let q be a formula, d an x-satisfiable conjunction of literals, D the set of letters occurring in d and I_d the x-interpretation of the alphabet D that maps all the literals of $[d]$ into 1 and the literals of $\overline{[d]}$ into 0. An x-interpretation $I_{L \setminus D}$ of the alphabet $L \setminus D$ satisfies $Simplify(q, [d], \overline{[d]})$ iff $I = I_{L \setminus D} \cup I_d$ satisfies q.*

PROOF: All the occurrences of literals of $[d]$ in q have been replaced by $\top$ in $Simplify(q, [d], \overline{[d]})$ and all the literals of $\overline{[d]}$ have been replaced by $\perp$, which are always mapped into 1 and 0, respectively. But this is exactly the same value they are mapped into by I_d, hence the x-interpretation $I_{L \setminus D}$ satisfies $Simplify(q, [d], \overline{[d]})$ iff I satisfies q. $\diamond$

(only if) Assume that $d \models^x q$ holds, d is x-satisfiable and $Simplify(q, [d], \overline{[d]})$ is not x-valid. We show that a contradiction follows. Let D be the set of letters occurring in d and let I_1 be an x-interpretation of the alphabet $L \setminus D$ not satisfying $Simplify(q, [d], \overline{[d]})$. Let I_2 be the interpretation of the alphabet D that maps all the literals of $[d]$ into 1 and the literals of $\overline{[d]}$ into 0. It is clearly the case that $I_2 \models d$. We now show that I_2 is an x-interpretation by a simple case analysis. If $x = 2$ or 3^*, since d is x-satisfiable it cannot contain complementary literals, hence if $l \in [d]$ then its complementary literal belongs to $\overline{[d]}$ therefore I_2 is both a 2- and a 3^*-interpretation of the alphabet D because it maps complementary literals into opposite values. If $x = 3$ or 4, since for each letter $l \in D$ either $l \in [d]$ or $\neg l \in [d]$ we have that I_2 maps at least one of the complementary literals into 1, hence it is both a 3- and a 4-interpretation of the alphabet D. Now, let I be an interpretation of the alphabet L which for all literals of $L \setminus D$ it coincides with I_1 and for all literals of D it coincides with I_2. Obviously, I is an x-interpretation, $I \models d$ and $I \not\models Simplify(q, [d], \overline{[d]})$. But by Lemma 4.4, I also does not satisfy q which implies $d \not\models^x q$, hence contradiction.

(if) If d is x-unsatisfiable it is obviously the case that $d \models^x q$ holds, hence we assume that d is x-satisfiable and $Simplify(q, [d], \overline{[d]})$ is x-valid but $d \not\models^x q$. Let I be an x-interpretation of the alphabet L which satisfies d but not q. We show that a contradiction follows. Since it satisfies d it must map all of its literals into 1. Let I' be another x-interpretation which agrees with I for all literals not in $[d] \cup \overline{[d]}$, maps all the literals of $[d]$ into 1 and all the literals of $\overline{[d]}$ into 0. It is clearly the case that $I' \leq I$, hence by Lemma 3.1 we have that $I' \not\models q$. But by Lemma 4.4, I' does not satisfy $Simplify(q, [d], \overline{[d]})$, hence contradiction. $\diamond$

This theorem tells us that whenever we can put T in DNF in polynomial time and we can decide x-validity in polynomial time then we have a polynomial algorithm for deciding $T \models^x q$ (the cost of testing x-satisfiability of a conjunction of literals is negligible w. r. t. the other costs). More precisely, let us denote with Γ the number of conjunctions occurring in $DNF(T)$, with Ξ the time complexity of simplifying q with each such conjunction, with Θ_x the time occurring to test x-validity of each simplified formula. The above theorem states that $T \models^x q$ can be tested in $O(\Gamma \cdot (\Xi + \Theta_x))$ time. As an example, if T is already in DNF and we are dealing with the combined complexity of 3^*-entailment, the test can be done in $O(|T| \cdot |q|)$ time. The same upper bound holds for all the polynomial cases.

Let us give a list of all the polynomial cases we can obtain using the above result along with **Val-1**, **Val-2**, **Val-3** and **Val-4**:

- $T \models^2 q$ can be decided in polynomial time if both:
  1. $DNF(T)$ can be computed in polynomial time **and**
  2. $CNF(q)$ can be computed in polynomial time.
- $T \models^3 q$ can be decided in polynomial time if both:
  1. $DNF(T)$ can be computed in polynomial time **and**
  2. $CNF(q)$ can be computed in polynomial time.
- $T \models^{3^*} q$ can be decided in polynomial time if $DNF(T)$ can be computed in polynomial time.

- $T \models^4 q$ can be decided in polynomial time if $DNF(T)$ can be computed in polynomial time.

The following example — which continues Example 4.2 — shows how Theorem 4.3 applies.

Example 4.4 (knowledge base in DNF) As in Example 4.2, q is $(a \vee (b \wedge \neg c)) \rightarrow \neg(a \vee c)$. Let T be $\neg a \wedge b$. The simplification of the query q with respect to T works as follows:

$$\begin{array}{rcl} [T] & = & \{\neg a, b\} \\ \overline{[T]} & = & \{a, \neg b\} \\ Simplify(q, [T], \overline{[T]}) & = & (\top \wedge (\bot \vee c)) \vee (\top \wedge \neg c) \end{array}$$

Notice that $Simplify(q, [T], \overline{[T]})$ is both 2- and 3-valid. If both c and $\neg c$ are mapped into 0 the formula is not satisfied, hence it is neither 3^*- nor 4-valid. As a consequence, $T \models^2 q$, $T \models^3 q$, $T \not\models^{3^*} q$ and $T \not\models^4 q$ hold. $\diamond$

Using Theorems 4.1 and 4.3 we are able to complete Table 2. In particular all the missing entries correspond to polynomial-time problems. Table 3 summarizes all the complexity results on the entailment of propositional multivalued logics.

$T \models^x q$		COMBINED				DATA				EXPRESSION			
T	q	2	3	3^*	4	2	3	3^*	4	2	3	3^*	4
Any	Any	coNP	coNP	coNP	coNP	coNP	P	coNP	P	coNP	coNP	P	P
Any	CNF	coNP	P	coNP	P	coNP	P	coNP	P	P	P	P	P
Any	DNF	coNP	coNP	coNP	coNP	coNP	P	coNP	P	coNP	coNP	P	P
CNF	Any	coNP	coNP	coNP	coNP	coNP	P	coNP	P	coNP	coNP	P	P
CNF	CNF	coNP	P	coNP	P	coNP	P	coNP	P	P	P	P	P
CNF	DNF	coNP	coNP	coNP	coNP	coNP	P	coNP	P	coNP	coNP	P	P
DNF	Any	coNP	coNP	P	P	P	P	P	P	coNP	coNP	P	P
DNF	CNF	P	P	P	P	P	P	P	P	P	P	P	P
DNF	DNF	coNP	coNP	P	P	P	P	P	P	coNP	coNP	P	P

Table 3: The complexity of checking $T \models^x q$

It is important to point out that the two methods delivered by Theorems 4.1 and 4.3 provide us with polynomial algorithms both for the previously known polynomial cases (shown in Section 3, cf. Observations 3.2 and 3.3) and for the new ones. With two simple methods we are able to characterize a large number of tractable cases.

5 Discussion

It is useful to discuss the relevance and applicability of the complexity analysis presented in the last section.

Table 3 shows that the syntactic form of the KB and of the query are indeed sources of complexity. The worst case is obviously when no assumption on the syntactic form is made. It is interesting to notice that when the KB is in CNF and the query in DNF the complexity is the same as in the worst case. Furthermore, the first, second and third rows of Table 3 are identical to the fourth, the fifth and the sixth ones respectively. In other words it makes no difference whether the KB is in CNF or not. The generally assumed hypothesis that the KB is in CNF (see for example the work by Frisch and Levesque [13, 20, 22]) can therefore be withdrawn.

As is already known from the literature, weakening inference by adopting a multivalued semantics is a technique for decreasing the complexity of reasoning. Table 3 makes a clear picture of the effectiveness of this technique, showing exactly when multivalued logics are indeed useful for this goal.

As noticed in Section 4, the combined complexity of our algorithms is $O(|T| \cdot |q|)$ for all the polynomial cases. This means $O(|T|)$ for the data complexity and $O(|q|)$ for the expression complexity. This is exactly the same complexity of Levesque's algorithm (cf. Observation 3.3 and [22]) that only applies to a specific case.

An important fact to be noticed about the results of Table 3 is the following: columns corresponding to data and expression complexity analysis show a significant number of polynomial cases with respect to the combined complexity ones. As an example, the data complexity of 3-entailment is polynomial, while the combined complexity is co-NP-complete in the worst case. Analogously, both the data and the expression complexity of 4-entailment is polynomial, regardless of the syntactic form of the formulae, while the combined complexity is co-NP-complete in the worst case. This shows that the combined complexity cannot be obtained by taking the maximum (in the obvious sense) of the data and expression complexity.

In this paper we showed that the task of proving entailment in propositional multivalued logics can be reduced to proving satisfiability or validity of a formula. As a consequence, the methods described can be efficiently used along with algorithms for proving satisfiability or validity, even for more restricted classes of formulae.

As an example, it is well-known that Horn CNF formulae admit linear-time algorithms for testing 2-satisfiability (cf. [9]). Using Theorem 4.1 and the fact that the function $Simplify$ applied to a Horn CNF formula outputs a Horn CNF formula, we can say that testing whether $T \models^{3^*} q$ hold can be done in polynomial time whenever T is a Horn formula and $CNF(q)$ can be computed in polynomial time. Similar arguments can be performed for other efficient algorithms for testing satisfiability or validity.

The assumptions used in the data and expression complexity analyses are very strong. In data complexity it is assumed that the size of the query is completely irrelevant and can be ignored. Dually, the expression complexity assumption allows us to ignore the size of the KB. In fact it is possible to relax these assumptions without changing the edge of tractability. For example, we can relax the data complexity assumption and say that the size of the query is logarithmic in the size of the KB, i.e. $|q| = O(\ln |T|)$. What happens is that we may have to take into account the cost of changing the syntactic form of the query, e.g. computing its CNF. The time complexity of this transformation is $O(2^{|q|}) = O(|T|)$ and the output is a formula with size $O(|T|)$. As a consequence, complexity of inference may increase up to $O(|T|^2)$. Dually, if we relax the expression complexity assumption and say that the size of the KB is logarithmic in the size of the query, i.e. $|T| = O(\ln |q|)$, complexity of inference may increase up to $O(|q|^2)$. Anyway polynomial cases continue to remain polynomial while intractable cases remain intractable.

It is interesting to investigate the optimality of all the algorithms we presented in this work. In particular we want to briefly analyze the importance of adopting adequate data structures for representing logical formulae.

Here we focus on Levesque's algorithm for testing 4-entailment when the KB is a CNF formula $T = T_1 \wedge \cdots \wedge T_n$ and the query q is a clause. The algorithm is described in Observation 3.3 and basically works as follows: check if there exists a clause T_i of T such that $T_i \models^4 q$, i.e. $[T_i] \subseteq [q]$. If q is represented as a list and T is a list of lists, the complexity of testing $[T_i] \subseteq [q]$ is $O(|T_i| \cdot |q|)$ and therefore the complexity of the whole algorithm is $O(\sum_{i=1}^{n}(|T_i| \cdot |q|)) = O(|T| \cdot |q|)$. If an ordering in the set of letters is assumed and the clause q is represented as an AVL tree (cf. for example [17]), the test $[T_i] \subseteq [q]$ can be performed in time $O(|T_i| \cdot log(|q|))$. Representing q in this way costs $O(|q| \cdot log(|q|))$ and the complexity of testing whether $T \models^4 q$ holds is $O(\sum_{i=1}^{n}(|T_i| \cdot log(|q|))) = O(|T| \cdot log(|q|))$. This technique can be obviously generalized to the case in which q is a CNF formula $q_1 \wedge \cdots \wedge q_m$. In such case the answer is yes iff for each j $(1 \leq j \leq m)$ there exists an i $(1 \leq i \leq n)$ such that $[T_i] \subseteq [q_j]$. This can be done by building m AVL trees in time $O(\sum_{j=1}^{m} |q_j| \cdot log(|q_j|))$ and the complexity of testing whether $T \models^4 q$ holds is $O(|T| \cdot \sum_{j=1}^{m} log(|q_j|))$.

This shows that the $O(|T| \cdot |q|)$ time complexity is not optimal in general and can be improved by carefully choosing an appropriate data structure. This kind of analysis deserves further investigation.

Multivalued semantics have been frequently adopted for obtaining tractable formalisms for knowledge representation. As an example, Patel-Schneider describes in [23] a tractable four-valued semantics for terminological reasoning, and generalizes it to a full first-order language in [24]. Levesque [20] and Lakemeyer [19] propose tractable languages for epistemic reasoning. The present authors described in [4, 5, 6] systems based on multivalued logics for describing tractable deduction in terminological, epistemic and non-monotonic reasoning. All of those systems adopt strong syntactic restrictions, that might be relaxed using techniques presented in this paper.

The results shown in this paper could also be used for analyzing the computational complexity of more sophisticated multivalued logics, like those described by Kifer and Lozinskii in [18] and by Ginsberg in [15].

We conclude the section by noticing that the use of partial evaluation for speeding up logical inference — as the function *Simplify* does — is not completely new. For example Dalal shows in [8] an algorithm, called *propositional fact propagation*, that performs a form of unit resolution on an NNF propositional formula ϕ. The output of the algorithm is a simpler formula ϕ' that is satisfiable iff ϕ is so.

6 Conclusions

In this paper we have investigated about the complexity of entailment in four popular propositional multivalued logics. The major results of this analysis are:

- a detailed picture of the threshold between polynomiality and intractability of the problem has been provided. In particular we showed several new polynomial cases;
- two polynomial-time algorithms capturing several tractable cases have been developed. The algorithms are based on a single technique and work both for the polynomial cases already known from the literature and for the new cases;
- the restriction to CNF formulae, adopted in all the previous studies, is frequently not needed to ensure polynomiality;
- the data and expression complexity analyses — performed in this work for the first time — provided several interesting results. For example, while the combined complexity of 4-entailment is co-NP-complete, we proved that both its data and expression complexity is polynomial, regardless of the syntactic form of the formulae involved.

Acknowledgments

We are grateful to Hector Levesque, Alan Frisch and Fabrizio d'Amore for helpful discussions. This work has been supported by the ESPRIT Basic Research Action N.6810-COMPULOG II and by the *Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo* of the CNR (Italian Research Council). The first author is supported by a fellowship from CNR.

References

[1] A. Anderson and N. Belnap. *Entailment: The Logic of Relevance and Necessity.* Princeton University Press, Princeton NJ, 1975.

[2] N. Belnap. A useful four-valued logic. In G. Eppstein and J. Dunn, editors, *Modern uses of many-valued logics*, pages 8–37. Reidel, Dordrecht, Netherlands, 1977.

[3] R. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 14. Elsevier Science Publishers B. V. (North Holland), 1990.

[4] M. Cadoli and M. Schaerf. Approximate inference in default reasoning and circumscription. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 319–323, August 1992. Full version appears in *Fundamenta Informaticae*, 21:103-113, 1994.

[5] M. Cadoli and M. Schaerf. Approximate reasoning and non-omniscient agents. In *Proceedings of the Fourth Conference on Theoretical Aspects of Reasoning about Knowledge (TARK-92)*, pages 169–183, 1992.

[6] M. Cadoli and M. Schaerf. Approximation in concept description languages. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning (KR-92)*, pages 330–341, October 1992.

[7] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory Of Computing (STOC-71)*, pages 151–158, 1971.

[8] M. Dalal. Efficient propositional constraint propagation. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 409–414, 1992.

[9] W. P. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.

[10] R. Fagin, J. Y. Halpern, and M. Y. Vardi. A nonstandard approach to the logical omniscience problem. In *Proceedings of the Third Conference on Theoretical Aspects of Reasoning about Knowledge (TARK-90)*, pages 41–55, 1990.

[11] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2:295–312, 1985.

[12] A. M. Frisch. Using model theory to specify AI programs. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 148–154, 1985.

[13] A. M. Frisch. Inference without chaining. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 515–519, 1987.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, Ca, 1979.

[15] M. L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[16] J. Hintikka. Impossible possible worlds vindicated. *Journal of Philosophical Logic*, 4:475–484, 1975.

[17] E. Horowitz and S. Sahni. *Fundamentals of data structures.* Pitman, London, 1976.

[18] M. Kifer and E. L. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 9(2):179–215, November 1992.

[19] G. Lakemeyer. Tractable meta-reasoning in propositional logics of belief. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 402–408, 1987.

[20] H. J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the Fourth National Conference on Artificial Intelligence (AAAI-84)*, pages 198–202, 1984.

[21] H. J. Levesque. Logic and the complexity of reasoning. *Journal of Philosophical Logic*, 17:355–389, 1988.

[22] H. J. Levesque. A knowledge-level account of abduction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 1061–1067, 1989.

[23] P. F. Patel-Schneider. A four-valued semantics for terminological logic. *Artificial Intelligence Journal*, 38:319–351, 1989.

[24] P. F. Patel-Schneider. A decidable first-order logic for knowledge representation. *Journal of Automated Reasoning*, 6:361–388, 1990.

[25] T. Przymusinski. Three-valued formalizations of non-monotonic reasoning and logic programming. In *Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning (KR-89)*, pages 341–348, 1989.

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

[27] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposyum on Theory Of Computing (STOC-82)*, pages 137–146, 1982.