SAT-Based Decision Procedures for Classical Modal Logics

Enrico Giunchiglia
DIST, V.le Causa 13
16145 Genova, Italy

Fausto Giunchiglia
DISA, Università di Trento.
IRST, 38050 Povo,
Trento, Italy

Armando Tacchella
DIST, V.le Causa 13
16145 Genova, Italy

Abstract. We present a set of SAT-based decision procedures for various classical modal logics. By SAT-based, we mean built on top of a SAT solver. We show how the SAT-based approach allows for a modular implementation for these logics. For some of the logics we deal with, we are not aware of any other implementation. For the others, we define a testing methodology which generalizes the $3\mathrm{CNF}_K$ methodology by Giunchiglia and Sebastiani. The experimental evaluation shows that our decision procedures perform better than or as well as other state-of-the-art decision procedures.

1. Introduction

Propositional reasoning is a fundamental problem in many areas of Computer Science. Many researchers have put, and still put, years of effort in the design and implementation of new and more powerful SAT solvers. Most importantly, the source code of many of these implementations is freely available and can be used as a basis for the development of decision procedures for more expressive logics (see, e.g., (Giunchiglia and Sebastiani, 1996a; Cadoli *et al.*, 1998)).

In this paper, we present a set of SAT-based decision procedures for various classical modal logics (Montague, 1968; Segerberg, 1971). By SAT-based, we mean built on top of a SAT solver. We show how the SAT-based approach allows for a modular implementation for these logics. For some of the logics we deal with, we are not aware of any other implementation. For the others, we define a testing methodology which generalizes the $3\mathrm{CNF}_K$ methodology by Giunchiglia and Sebastiani (1996a). The experimental evaluation shows that our decision procedures perform better than or as well as other state-of-the-art decision procedures.

Our approach, first suggested by Giunchiglia and Sebastiani (1998), consists of two steps:

1. take off the shelf one of the fastest SAT procedures available, and

2. use it as the basis for your modal decider.

Our approach differs from most of the previous works in decision procedures for modal logics, which are either tableau-based, (i.e., based on Smullyan's tableau for propositional logic (Smullyan, 1968): see, e.g., (Fitting, 1983; Massacci, 1994; Baader *et al.*, 1994)), or translation-based (i.e., based on a reduction to first order logic: see, e.g., (van Benthem, 1984; Ohlbach, 1988; Hustadt and Schmidt, 1997a)). Some of the differences and advantages of the SAT-based approach with respect to the tableau based and translation based approaches have been already pointed out in (Giunchiglia and Sebastiani, 1996b; Giunchiglia *et al.*, 1998). Here we show how the SAT-based approach provides for a simple and natural schema for the development of decision procedures for modal logics.

Our approach differs also from the approaches by Horrocks (1998) and Patel-Schneider (1998). These authors developed their own SAT checker and used it as a basis for their modal decider. This approach allows for a better integration among the different modules of the system and for a finer tuning of the reasoning strategies. On the other hand, it does not exploit the great amount of ongoing work on SAT deciders. Each year, new and faster SAT engines are proposed. With some standard modifications of the source code we can inherit in the modal setting all the benefits of state-of-the-art propositional checkers.[1]

Finally, our system, called *SAT, improves also on previous works by Sebastiani and ourselves (1996a; 1996b; 1998). In particular,

- *SAT is built on top of the SAT decider SATO (Zhang, 1997). *SAT inherits many of the SATO configurable options, and allows for some more options (including early pruning, caching, two forms of backjumping, and the choice of various splitting heuristics); and

- *SAT is able to deal with 8 modal logics, namely E, EM (=M), EN, EMN, EC, EMC (=R), ECN, EMCN (=K) (see (Chellas, 1980)). For EN, EMN, EC and ECN, we do not know of any

[1] A similar point is made by Kautz and Selman (1998) as an explanation for the better performances of SATPLAN with respect to specialized engines for planning problems.

other implemented decision procedure, nor of any reduction to a formalism for which a decision procedure is available.

The paper is structured as follows. In Section 2 we review some basic definitions about classical modal logics. Then, in Section 3, we present the idea of the SAT-based approach to modal reasoning, and discuss in detail the algorithms for the 8 logics we consider. Section 4 is devoted to *SAT: it discusses some of the motivations underlying its construction, and some of its features. The experimental analysis comparing *SAT with other state-of-the-art decision procedures is done in Section 5. Our analysis is restricted to K (for which there are several systems available) and E (for which there exists a reduction to bimodal K). We end in Section 6 with the conclusions and the future work.

2. Classical modal logics

Following (Chellas, 1980), a *modal logic* is a set of formulas (called *theorems*) closed under tautological consequence. Most modal logics are closed under the rule

$$\frac{(\varphi_1 \wedge \ldots \wedge \varphi_n) \supset \psi}{(\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) \supset \Box\psi}$$

for certain values of n. If $n = 1$ then the logic is said to be *monotone*. If $n = 2$ then the logic is said to be *regular*. If $n \geq 0$ then the logic is said to be *normal*. The smallest monotone, regular and normal modal logics are called M, R and K respectively (see, e.g., (Chellas, 1980)).

Classical modal logics (Montague, 1968; Segerberg, 1971) are weaker than normal modal logics. In fact, the only requirement is that the set of theorems is closed under the rule

$$\frac{\varphi \equiv \psi}{\Box\varphi \equiv \Box\psi}\ .$$

As a consequence, the schemas

N: $\Box\top$,[2]

M: $\Box(\varphi \wedge \psi) \supset \Box\varphi$,

C: $(\Box\varphi \wedge \Box\psi) \supset \Box(\varphi \wedge \psi)$,

[2] The symbols $\top$ and $\bot$ are 0-ary connectives representing truth and falsity respectively.

which are theorems in K do not necessarily hold in classical modal logics. The three principles N, M, and C enforce closure conditions on the set of provable formulas which are not always desirable, especially if the $\Box$ operator has an epistemic (such as knowledge or belief) reading. If we interpret $\Box\varphi$ as "a certain agent a believes φ", then N enforces that a believes all the logical truths, M that a's beliefs are closed under logical consequence, and C that a's beliefs are closed under conjunction. These three closure properties are different forms of omniscience, and —as such— they are not appropriate for modeling the beliefs of a real agent (see, e.g., (Giunchiglia and Giunchiglia, 1997) and (Fagin *et al.*, 1995) Chapter 9). We can easily imagine situations involving a's beliefs, where only an arbitrary subset of the above properties holds.

There are eight possible ways to add the three schemas M, C, and N to the smallest classical modal logic E. The resulting modal logics are called E, EM (equivalent to the logic M), EN, EMN, EC, EMC (equivalent to R), ECN, EMCN (equivalent to K), where EX denotes the logic obtained adding the schemas in X to E.

3. SAT-based procedures for classical modal logics

We say that a conjunction μ of propositional literals and formulas of the form $\Box\varphi$ or $\neg\Box\varphi$ is an *assignment* if, for any pair ψ, ψ' of conjuncts in μ, it is not the case that $\psi = \neg\psi'$. An assignment μ *satisfies* a formula φ if μ entails φ by propositional reasoning. A formula φ is *consistent* in a logic L (or L-*consistent*) if $\neg\varphi$ is not a theorem of L, i.e., if $\neg\varphi \notin$ L.

Consider a formula φ. Let S be a set of assignments satisfying φ, and let L be a modal logic. As noticed by Sebastiani and Giunchiglia (1997), the following two facts hold:

- If at least one assignment in S is L-consistent then φ is L-consistent.
- If no assignment in S is L-consistent then φ is not L-consistent as long as the set S is *complete for* φ, i.e., as long as the disjunction of the assignments in S is propositionally equivalent to φ.

From these facts, it follows that the problem of determining whether φ is L-consistent can be decomposed in two steps:

- *generate* an assignment μ satisfying φ, and
- *test* whether μ is L-consistent.

In all the logics we consider, testing the consistency of an assignment μ amounts to determining the consistency of other formulas whose depth (i.e., the maximum number of nested $\Box$ operators) is strictly minor than the depth of μ. This implies that we can check the consistency of these other formulas by recursively applying the above methodology, at the same time ensuring the termination of the overall process. The above methodology can be implemented by two mutually recursive procedures:

- LSAT(φ) for the generation of assignments satisfying φ, and
- LCONSIST(μ) for testing the L-consistency of each generated assignment μ.

In order to simplify the presentation, we first present LCONSIST(μ), and then LSAT(φ).

3.1. LCONSIST(μ)

Whether an assignment is consistent, depends on the particular logic L being considered. Furthermore, depending on the logic L considered, the consistency problem for L (i.e., determining whether a formula is consistent in L) belongs to different complexity classes. In particular, the consistency problem for E, EM, EN, EMN is NP-complete, while for EC, ECN, EMC, EMCN it is PSPACE-complete (see (Vardi, 1989; Fagin *et al.*, 1995)). Here, to save space, we divide these eight logics in two groups. We present the algorithms for checking the L-consistency of an assignment first in the case in which L is one of E, EM, EN, EMN, and then in the case in which L is one of the others.

3.1.1. *Logics E, EM, EN, EMN*

The following Proposition is an easy consequence of the results presented in (Vardi, 1989).

PROPOSITION 1. *Let* $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ *be an assignment in which* γ *is a propositional formula. Let L be one of the logics E, EM, EN, EMN.* μ *is consistent in L if for each conjunct* $\neg\Box\beta_j$ *in* μ *one of the following conditions is satisfied:*

- $(\alpha_i \equiv \neg\beta_j)$ *is L-consistent for each conjunct* $\Box\alpha_i$ *in* μ*, and L=E;*
- $(\alpha_i \wedge \neg\beta_j)$ *is L-consistent for each conjunct* $\Box\alpha_i$ *in* μ*, and L=EM;*
- $\neg\beta_j$ *and* $(\alpha_i \equiv \neg\beta_j)$ *are L-consistent for each conjunct* $\Box\alpha_i$ *in* μ*, and L=EN;*
- $\neg\beta_j$ *and* $(\alpha_i \wedge \neg\beta_j)$ *are L-consistent for each conjunct* $\Box\alpha_i$ *in* μ*, and L=EMN.*

```
function LCONSIST(⋀_i □α_i ∧ ⋀_j ¬□β_j ∧ γ)
  foreach conjunct □β_j do
    foreach conjunct □α_i do
      if M[i,j] = Undef then M[i,j] := LSAT(α_i ∧ ¬β_j);
      if L ∈ {EN,EMN} and M[i,j] = True then M[j,j] := True;
      if L ∈ {E,EN} and M[i,j] = False then
        if M[j,i] = Undef then M[j,i] := LSAT(¬α_i ∧ β_j);
        if L = EN and M[j,i] = True then M[i,i] := True;
        if M[j,i] = False then return False
    end
    if L ∈ {EN,EMN} then
      if M[j,j] = Undef then M[j,j] := LSAT(¬β_j);
      if M[j,j] = False then return False
  end;
  return True.
```

Figure 1. LCONSIST for E, EM, EN, EMN

When implementing the above conditions, care must be taken in order to avoid repetitions of consistency checks. In fact, while an exponential number of assignments satisfying the input formula can be generated by LSAT, at most n^2 checks are possible in L, where n is the number of "$\Box$" in the input formula. Given this upper bound, for each new consistency check, we can cache the result for a future possible re-utilization in a $n \times n$ matrix M. This ensures that at most n^2 consistency checks will be performed. In more detail, given an enumeration $\varphi_1, \varphi_2, \ldots, \varphi_n$ of the boxed subformulas of the input formula, M[i,j], with $i \neq j$, stores the result of the consistency check for $(\varphi_i \wedge \neg\varphi_j)$. M[i,i] stores the result of the consistency check for $\neg\varphi_i$. Initially, each element of the matrix M has value *Undef* (meaning that the corresponding test has not been done yet). The result is the procedure LCONSIST in Figure 1.

Consider Figure 1 and assume that L=E or L=EN. Given a pair of conjuncts $\Box\alpha_i$ and $\neg\Box\beta_j$, we split the consistency test for $(\alpha_i \equiv \neg\beta_j)$ in two simpler sub-tests:

- first, we test whether $(\alpha_i \wedge \neg\beta_j)$ is consistent, and
- only if this test gives *False*, we test whether $(\neg\alpha_i \wedge \beta_j)$ is consistent.

Notice also that, in case L=EN or L=EMN, if we know that, e.g., $(\alpha_i \wedge \neg\beta_j)$ is consistent, then also $\neg\beta_j$ is consistent and we store this result in M[j,j].

PROPOSITION 2. *Let* $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ *be an assignment in which* γ *is a propositional formula. Let L be one of the logics E, EM, EN, EMN. Assume that, for any formula* φ *whose depth is less than the depth of* μ*,* LSAT(φ)

- *returns True if* φ *is L-consistent, and*

- *False otherwise.*

LCONSIST(μ) *returns True if* μ *is L-consistent, and False otherwise.*

Proof: The Proposition is an easy consequence of the hypotheses and Proposition 1. ♠

3.1.2. *Logics EC, ECN, EMC, EMCN*
The following Proposition is an easy consequence of the results presented in (Vardi, 1989).

PROPOSITION 3. *Let* $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ *be an assignment in which* γ *is a propositional formula. Let* Δ *be the set of formulas* α_i *such that* $\Box\alpha_i$ *is a conjunct of* μ*. Let L be one of logics EC, ECN, EMC, EMCN.* μ *is consistent in L if for each conjunct* $\neg\Box\beta_j$ *in* μ *one of the following conditions is satisfied:*

- $((\bigwedge_{\alpha_i \in \Delta'} \alpha_i) \equiv \neg\beta_j)$ *is L-consistent for each non empty subset* Δ' *of* Δ*, and L=EC;*

- $((\bigwedge_{\alpha_i \in \Delta'} \alpha_i) \equiv \neg\beta_j)$ *is L-consistent for each subset* Δ' *of* Δ*, and L=ECN;*

- Δ *is empty or* $((\bigwedge_{\alpha_i \in \Delta} \alpha_i) \wedge \neg\beta_j)$ *is L-consistent, and L=EMC;*

- $((\bigwedge_{\alpha_i \in \Delta} \alpha_i) \wedge \neg\beta_j)$ *is L-consistent, and L=EMCN.*

Assume that L=EC or L=ECN. The straightforward implementation of the corresponding condition may lead to an exponential number of checks in the cardinality $|\Delta|$ of Δ. More carefully, for each conjunct $\neg\Box\beta_j$ in μ, we can perform at most $|\Delta| + 1$ checks if

1. for each formula α_i in Δ, we first check whether $(\neg\alpha_i \wedge \beta_j)$ is consistent in L. Let Δ' be the set of formulas for which the above test fails. Then,

function LCONSIST($\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma$)
  $\Delta := \{\alpha_i \mid \Box\alpha_i$ is a conjunct of $\mu\}$;
  foreach conjunct $\Box\beta_j$ **do**
    $\Delta' := \Delta$;
    if L $\in$ {EC, ECN} **then**
      foreach conjunct $\Box\alpha_i$ **do**
        if $M[j,i] = \mathit{Undef}$ **then** $M[j,i] :=$ LSAT($\neg\alpha_i \wedge \beta_j$);
        if $M[j,i] = \mathit{True}$ **then** $\Delta' = \Delta' \setminus \{\alpha_i\}$
      end;
    if L $\in$ {ECN, EMCN} **or** $\Delta' \neq \emptyset$ **then**
      if not LSAT($\bigwedge_{\alpha_i \in \Delta'} \alpha_i \wedge \neg\beta_j$) **then return** *False*
  end;
  return *True*.

Figure 2. LCONSIST for EC, ECN, EMC (R), EMCN (K)

2. in case L=ECN or $\Delta' \neq \emptyset$, we perform the last test, checking whether $((\bigwedge_{\alpha_i \in \Delta'} \alpha_i) \wedge \neg\beta_j)$ is consistent in L.

Furthermore, the result of the consistency checks performed in the first step can be cached in a matrix M analogous to the one used in the previous subsection.

If L=EC or L=ECN, the procedure LCONSIST in Figure 2 implements the above ideas. Otherwise, it is a straightforward implementation of the conditions in Proposition 3.

PROPOSITION 4. *Let $\mu = \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma$ be an assignment in which γ is a propositional formula. Let L be one of logics EC, ECN, EMC, EMCN. Assume that, for any formula φ whose depth is less than the depth of μ,* LSAT(φ)

– *returns True if φ is L-consistent, and*

– *False otherwise.*

LCONSIST(μ) *returns True if μ is L-consistent, and False otherwise.*

Proof: The Proposition is an easy consequence of the hypotheses and Proposition 3. ♠

```
function LSAT(φ)
   return LSAT_DP(cnf(φ), ⊤).

function LSAT_DP(φ, μ)
   if φ = ⊤ then return LCONSIST(μ);                  /* base */
   if φ = ⊥ then return False;                        /* backtrack */
   if { a unit clause (l) occurs in φ }               /* unit */
      then return LSAT_DP(assign(l, φ),μ ∧ l);
   l := choose-literal(φ, μ);
   return LSAT_DP(assign(l, φ),μ ∧ l) or              /* split */
            LSAT_DP(assign(l̄, φ),μ ∧ l̄).
```

Figure 3. LSAT and LSAT_{DP}

3.2. LSAT(φ)

Consider a formula φ. Let L be a modal logic. The generation of assignments satisfying φ is independent of the particular logic L being considered. Furthermore, it can be based on any procedure for SAT:

- if the SAT decider is complete, then we can generate a finite and complete set of assignments for φ as follows:

  - at step 0, ask for an assignment satisfying φ, and
  - at step $i + 1$, ask for an assignment satisfying φ and the negation of the assignments generated in the previous steps.

  By checking the L-consistency of each assignment, we obtain a correct and complete decider for L.

- if the SAT decider is correct but incomplete, then we cannot generate a complete set of assignments for φ, but we can still build a correct but incomplete decider for L by checking each generated assignment. Of course, whether an incomplete effective procedure for SAT can be turned into an effective incomplete procedure for a modal logic L, is still an open point.

The above method for generating a complete set of assignments for φ has the advantage that the SAT decider is used as a blackbox.

The obvious disadvantage is that the size of the input formula checked by the SAT solver may become exponentially bigger than the original one. A better solution is to invoke the test for L-consistency *inside* the SAT procedure whenever an assignment satisfying the input formula is found. In the case of the Davis-Putnam (DP) procedure (Davis and Putnam, 1960), we get the procedure LSAT represented in Figure 3. In the figure,

- $cnf(\varphi)$ is a set of clauses —possibly with newly introduced propositional variables— such that, for any assignment μ in the extended language, the following two properties are satisfied:[3]

  1. if μ satisfies $cnf(\varphi)$ then the restriction of μ to the language of φ satisfies φ, and
  2. if μ satisfies φ then there exists an assignment in the language of $cnf(\varphi)$ which (i) extends μ and (ii) satisfies $cnf(\varphi)$.

  Examples of such conversions are the "classical conversion" (which given a formula in negative normal form recursively distribute conjunctions over disjunctions), and the conversions based on "renaming", such as those described in (Tseitin, 1970; Plaisted and Greenbaum, 1986; de la Tour, 1990).

- $choose\text{-}literal(\varphi, \mu)$ returns a literal occurring in φ and chosen according to some heuristic criterion.

- if l is a literal, $\bar{l}$ stands for A if $l = \neg A$, and for A if $l = \neg A$;

- for any literal l and formula φ, $assign(l, \varphi)$ is the formula obtained from φ by (i) deleting the clauses in which l occurs as a disjunct, and (ii) eliminating $\bar{l}$ from the others.

As can be observed, the procedure LSAT$_{\text{DP}}$ in Figure 3 is the DP-procedure modulo the call to LCONSIST(μ) when it finds an assignment μ satisfying the input formula. Notice that in this procedure the pure literal rule, used by some DP implementations, is not implemented. In fact, our main interest is in correct and complete modal deciders. With the pure literal rule, the set of assignments checked by LCONSIST —assuming that each of such call returns *False*— is not ensured to be complete.[4]

[3] Let μ be an assignment in a language L. Let $L' \subseteq L$ be a language. The *restriction of* μ *to* L' is the assignment obtained from μ by deleting the conjuncts not in L'. Let μ' an assignment. μ' *extends* μ if each conjunct of μ is also a conjunct of μ'.

[4] According to some authors (see, e.g., Freeman (1995)) the pure literal rule only helps for a few classes of SAT problems. Furthermore, when this rule does help, it does not result in a substantial reduction in search tree size.

In the following, for any formula φ and for any assignment μ in the (possibly extended) language of *cnf*(φ), μ_φ is the restriction of μ to the language of φ. Analogously, for any set of assignments Γ, $\Gamma_\varphi = \{\mu_\varphi \mid \mu \in \Gamma\}$.

PROPOSITION 5. *Let φ be any modal formula. Assume that, for any assignment μ,* LCONSIST(μ) *prints μ and returns False. Let Γ be the set of assignments printed as the result of invoking* LSAT(φ). *The disjunction of the assignments in Γ_φ is propositionally equivalent to φ.*

Proof: In the hypotheses of the Proposition, what we need to prove is that

$$\varphi \equiv \bigvee_{\mu \in \Gamma_\varphi} \mu.$$

First, it is clear that *cnf*(φ) $\equiv \bigvee_{\mu \in \Gamma} \mu$. For the thesis, the right-to-left implication is an easy consequence of the first of the two properties of *cnf*(φ). Assume that the left-to-right implication is false. This means that there exists an assignment μ in the language of φ such that

(1) for any propositional atom A, either A or $\neg A$ is a conjunct of μ,

(2) for any formula $\Box\psi$ in the language of φ, either $\Box\psi$ or $\neg\Box\psi$ is a conjunct of μ,

(3) μ satisfies φ, and

(4) μ does not satisfy $\bigvee_{\mu \in \Gamma_\varphi} \mu$.

Given (3) and the second of the properties of *cnf*(φ), there must exist an assignment μ' in the language of *cnf*(φ) which

(5) extends μ,

(6) satisfies *cnf*(φ), and thus also $\bigvee_{\mu \in \Gamma} \mu$.

(6) means that μ' entails $\bigvee_{\mu \in \Gamma} \mu$ by propositional reasoning. Thus, there must exist an assignment μ'' in Γ such that —given (1), (2), (5)— each conjunct of μ''_φ is also a conjunct of μ'_φ and thus of μ. This implies that μ entails μ''_φ. Since $\mu''_\varphi \in \Gamma_\varphi$, μ entails $\bigvee_{\mu \in \Gamma_\varphi} \mu$, contradicting (4). ♠

We can now state and prove soundness and completeness results for our procedures.

Theorem [Soundness and Completeness] *Let L be one of the logics E, EM, EN, EMN, EC, ECN, EMC, EMCN.* LSAT *is sound and* *complete for L, i.e., for any modal formula φ,* LSAT(φ) *returns True if φ is L-consistent, and False otherwise.*

Proof: First observe that LSAT(φ) returns

- *True* if there exists a call to LCONSIST(μ) which returns *True*, and
- *False* if each call to LCONSIST(μ) returns *False*.

The proof is by induction on the depth d of φ. If $d = 0$ the Theorem is trivial. Assume that $d = m + 1$. By induction hypothesis, for any formula ψ whose depth is less than d, LSAT(ψ) returns *True* if ψ is L-consistent, and *False* otherwise. In the following, we use the following two facts:

(1) For any two assignments μ and μ' differing only for propositional conjuncts, μ is L-consistent iff μ' is L-consistent.
This is a trivial consequence of Proposition 1 and Proposition 3.

(2) Let μ be an assignment such that LCONSIST(μ) is invoked during the execution of LSAT(φ). LCONSIST(μ) returns *True* if μ is L-consistent and *False* otherwise.
Given that the depth of μ is less than or equal to d, this is a consequence of the the induction hypothesis, Proposition 2 and Proposition 4.

There are two cases:

1. φ is L-consistent. From Proposition 5, it follows that there exists a call to LCONSIST(μ) in which the assignment μ (*i*) satisfies *cnf*(φ), and (*ii*) is such that μ_φ is L-consistent. Since μ possibly extends μ_φ in that it may assign some newly introduced propositional variables, from (1) it follows that μ is L-consistent, and from (2) it follows that LCONSIST(μ) —and thus also LSAT(φ)— returns *True*.

2. φ is not L-consistent. From Proposition 5, it follows that for each call LCONSIST(μ), the assignment μ_φ is not L-consistent. As above, from (1) it follows that μ is not L-consistent, and from (2) it follows that LCONSIST(μ) returns *False*. ♠

4. *SAT

*SAT is built on top of SATO ver. 3.2 (Zhang, 1997). The choice of adopting SATO as the basis for our system has been driven by the following motivations:

- SATO is fast. We have not performed and are not aware of any up-to-date extensive comparison among the different SAT solvers publicly available.[5] However, according to some experiments we have done and to the results presented in (Zhang, 1997), SATO seems to behave better than most of the currently available SAT solvers.

- SATO has many options, including various splitting heuristics and backjumping. We have inherited some of these options, and they are available for experimentation.

- SATO has been written using some Software Engineering conventions which have made and will make much easier to tune it for our goals.

Besides the options inherited from SATO, our system allows for other possibilities that we have developed while implementing the system. It is out of the goals of this paper to describe *SAT structure, optimizations and configurable options. For a more detailed presentation, see (Tacchella, 1999) and the manual distributed with *SAT. For our goals, it suffices to say that the core of *SAT is a C implementation of the procedures LSAT and LCONSIST in Figures 1, 2, 3. In particular, with reference to Figure 3, in *SAT

- *cnf*(φ) is the set of clauses obtained from φ by applying a conversion based on renaming, such as those described in (Tseitin, 1970; Plaisted and Greenbaum, 1986).

- *choose-literal*(φ, μ) returns a literal according to a MOMS heuristic (Maximum Occurrences in clauses of Minimum Size) (Freeman, 1995).

- *assign*(l, φ) is a highly optimized procedure which takes time linear in the number of occurrences of l in φ.

*SAT also implements two important optimizations which have been used in the tests presented in the next Section:

Early-pruning. Before each splitting step in LSAT, the L-consistency of the assignment generated so far is checked by a call to LCONSIST. As in KSATC, care is taken to avoid the repetition of L-consistency checks on the same branch of the propositional search tree.[6] Early

[5] See `http://aida.intellektik.informatik.th-darmstadt.de/~hoos/SATLIB/` for a list of publicly available SAT solvers and more.

[6] This is obtained through a pointer in the assignment stack which keeps track of the portion of the current assignment which has been already verified to be L-consistent.

pruning has proved to be very effective at least on some of the tests presented in the next Section (see (Giunchiglia *et al.*, 1998)).

Caching for K. In the case of the logic K, *SAT uses an additional data structure which allows to associate to any formula φ the result of LSAT(φ). Before invoking LSAT on a formula ψ, LCONSIST checks whether the K-consistency of ψ has already been determined. As in DLP, caching introduces some additional costs, but it may produce dramatic speedups (see (Horrocks and Patel-Schneider, 1999b)).

We have not yet conducted an exhaustive experimental analysis to see, for each class of formulas, which combination of *SAT options leads to the best results (see (Horrocks and Patel-Schneider, 1999b) for a similar study of DLP options). In all the tests in the next Section, we used *SAT options which seemed more reasonable to us. In particular, we have set *SAT as to use

- early pruning on all tests, and
- caching (for K) when the depth of the input formula is greater than 1.

5. A comparative analysis

The availability of decision procedures for the logics we consider varies significantly. For EMCN, that we recall is equivalent to K, there are many implemented decision procedures available, see, e.g., (Franconi *et al.*, 1998; de Swart, 1998). For E, EM and EMC, Gasquet and Herzig (1996) provide a reduction to normal modal logics: by implementing this reduction we indirectly obtain decision procedures for these logics. Fitting (1983) calls U the logic EM, and defines a tableau system for it. More recently, Governatori and Luppi (1999) define a tableau-like proof system for classical, monotonic and regular modal logics. We are not aware of any implementation of these tableau systems. For EN, EC, EMN and ECN we are not aware of any other implemented decision procedure, nor of any reduction into a formalism for which a decision procedure is available.

Our comparative analysis is restricted to K and E. In fact, both our decision procedures for E and EM, and Gasquet and Herzig's reductions for E and EM, are similar. We expect that the experimental analysis for EM would lead to results similar to the ones we have for E. For

EMC, Gasquet and Herzig's reduction is to a normal modal logic for which we do not have a system available.[7]

5.1. Modal K

As we said, there are several systems able to solve the consistency problem for K. In our comparative experimental analysis, we consider the four systems *SAT, KSATC (Giunchiglia *et al.*, 1998), DLP (Patel-Schneider, 1998) and TA (Hustadt and Schmidt, 1997a), i.e., some among the fastest solvers for K. We remember that TA, given a modal formula φ, first determines a corresponding first order formula φ^* and then it performs conventional first-order theorem proving. In our tests, as in (Hustadt and Schmidt, 1997a), TA uses FLOTTER to convert φ^* in a set of clauses $Cl(\varphi^*)$, and then the theorem prover SPASS to solve $Cl(\varphi^*)$. For a brief description of FLOTTER and SPASS, see (Weidenbach *et al.*, 1996).[8]

We test these systems on three problem sets of randomly generated 3CNF_K formulas. A 3CNF_K formula is a conjunction of 3CNF_K clauses, each with three disjuncts. Each disjunct in a 3CNF_K clause is either a propositional literal or a formula having the form $\Box C$ or $\neg\Box C$, where C is a 3CNF_K clause. See (Giunchiglia and Sebastiani, 1996a) for a more detailed presentation. We only remark that for any formula φ there exist a 3CNF_K formula which is K-consistent iff φ is K-consistent.

Sets of 3CNF_K formulas can be randomly generated according to the following parameters:

(i) the modal depth d;

(ii) the number L of clauses at depth $d = 0$;

[7] The reduction maps the consistency problem for a formula φ in EMC, into the consistency problem for a formula φ' in the smallest normal modal logic with two modal operators $\Box_1$, $\Box_2$ and augmented with the schema

$$\psi \supset \Box_1 \psi.$$

See (Gasquet and Herzig, 1996) for more details.

[8] The experimental results have been obtained with DLP ver. 3.1, TA ver. 1.4 (SPASS/FLOTTER ver. 0.55), KSATC ver. 1.0, and *SAT ver. 1.2. To compile the systems we have used `sml-nj 110.0.3`, `sicstus prolog 3`, `ACL 5.0`, and `gcc 2.7.2.3`. The tests have been run on several Intel PCs, whose configuration varies from `P200MHz` with `64MbRAM`, up to a `PII350MHz` with `256 MbRAM`. All platforms are running `Linux 5.x RedHat`. DLP is available at `http://www-db.research.bell-labs.com/user/pfps`. TA is available at `http://www.doc.mmu.ac.uk/STAFF/U.Hustadt/mdp`. KSATC is available at `ftp://ftp.mrg.dist.unige.it/` in `pub/mrg-systems`. *SAT is available at the WEB page `http://www.mrg.dist.unige.it/~tac/StarSAT.html`.

(iii) the number N of propositional variables;

(iv) the probability p with which a disjunct occurring in a clause at depth $< d$ is purely propositional.

Care is taken in order to avoid multiple occurrences of a formula in a clause, at the same time ensuring that the modal vs. the propositional structure of each generated formula only depends on p. In more detail, a clause is generated by randomly generating its disjuncts. When generating a disjunct, we first decide whether it has to be a propositional literal or not. Then a disjunct of the proper type is repeatedly generated as long as it does not occur in the clause generated so far.

In the tests we consider, a problem set is characterized by N and p: d is fixed to 1, while L is varied in such a way to empirically cover the "100% satisfiable – 100% unsatisfiable" transition. For each pair N, p in a problem set, 100 3CNF$_K$ formulas are randomly generated, and the resulting formulas are given in input to the procedure under test. Then, for each run, we consider the time the systems take for the main processing of the formula, thus excluding the negligible time the systems take to read and somehow normalize the input formula. (In particular, for TA this means that we take into account only the time needed by SPASS to solve the formula generated at the end of the translation process.) For practical reasons, a timeout mechanism stops the execution of the system on a formula after 1000 seconds of CPU time. Even more, for any pair N, p, the execution of *SAT, KSATC and TA is stopped after the system exceeds the timeout on 51 of the 100 samples corresponding to the fixed N, p. DLP instead, stops its execution on the 100 samples corresponding to a pair N, p if a supermajority of the first n tests timeout.[9] When this happens, it is assumed that DLP exceeds the timeout for more than 50% of the tests with that N, p.

The first three problem sets we consider have $N = 4, 5, 6$ while p is fixed to 0%: according to Hustadt and Schmidt (1997b), fixing $p = 0\%$ corresponds to particularly difficult tests. We call these problem sets PKN4p0, PKN5p0, and PKN6p0 respectively. PKN4p0 and PKN6p0 are called PS12 and PS13 respectively in (Hustadt and Schmidt, 1997b). In order to better highlight the behavior of *SAT and KSATC, we also run these systems on a problem set (called PKN7p0) having $N = 7$ and $p = 0\%$. In Figure 4, satisfiability percentages and the median of the CPU times for the four systems are plotted against the number

[9] In more detail, DLP stops its execution if there are at least 5 tests so far and more than 90% of them timeout, or if there are at least 10 tests so far and more than 75% of them timeout, or if there are at least 20 tests so far and more than 55% of them timeout (Patel-Schneider, 1999).

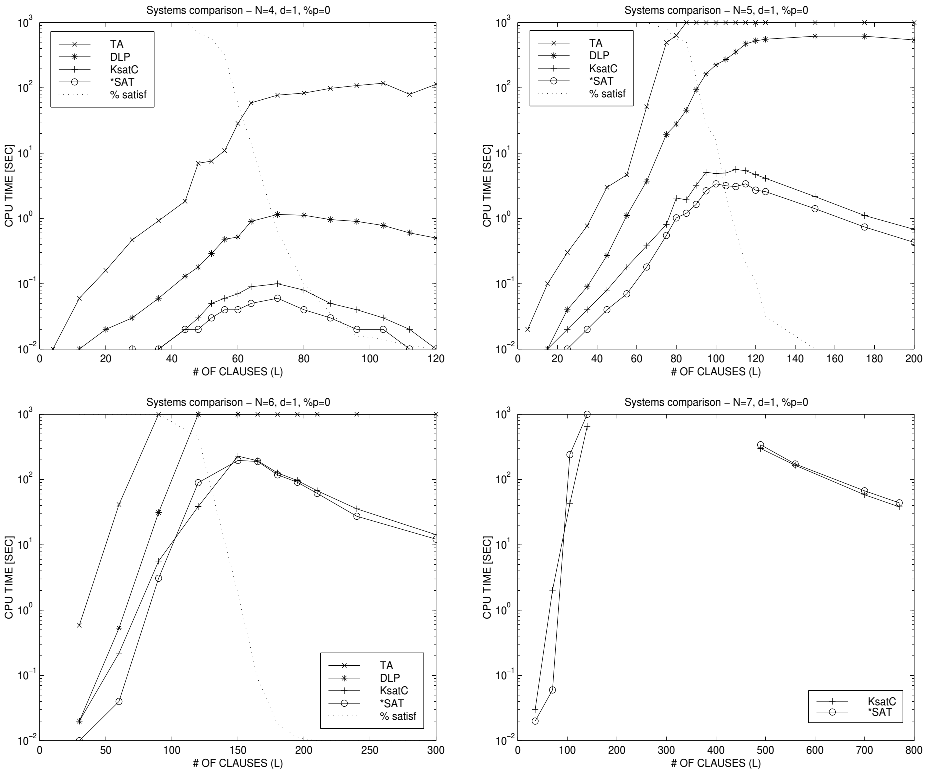

Figure 4. Logic K. *SAT, KSATC, DLP, and TA median CPU time. $N = 4, 5, 6, 7$. $p = 0\%$. 100 samples/point. Background: satisfiability percentage.

of clauses L. Notice the logarithmic scale on the vertical axis, which causes that values equal to 0.00 do not get plotted.

Consider Figure 4. The first observation is that *SAT and KSATC are the fastest. The two systems perform roughly in the same way, with *SAT performing better when $L = 4, 5$. For $L = 6, 7$, the two systems have similar performances, one system performing better than the other for some values of L, but worse for other values of L. This comes at no surprise: the two systems have the same underlying structure, both use early pruning and a MOMS heuristic to select the splitting literal. The two systems do not have an identical behavior because they use different data-structures and implement slightly different MOMS strategies.

Considering the other systems, for PKN4p0 the gap between *SAT/KSATC and DLP [resp. TA] is of more than one order of magnitude at the cross-over point of 50% of satisfiable formulas, and goes up to almost 2 [resp. 4] orders of magnitude at the right end side of the horizontal axis. For PKN5p0 and PKN6p0, TA median values exceed the timeout for $L = 85$ and $L = 90$ respectively, while the corresponding values of

*SAT [resp. KSATC] are 1.22 [resp. 1.83] and 3.32 [resp. 5.64] seconds. TA keeps exceeding the timeout for all the successive values. The gap between *SAT/KSATC and DLP on PKN5p0 is of more than one order of magnitude at the cross-over points of 50% satisfiable formula, and goes up to almost 3 orders at the very right of the plot. When $N = 6$, DLP median values exceed the timeout for $L = 120$ and $L = 150$. For $L > 150$, DLP does not terminate gracefully.[10] Comparing *SAT and DLP on PKN4p0 and PKN5p0, we see that

- the gap between *SAT and DLP seems to increase with L. Both for PKN4p0 and PKN5p0, the difference in logarithmic scale between *SAT and DLP is (almost always) monotonically increasing.
- At the crossover point of 50% satisfiable formulas, the gap between *SAT and DLP [resp. TA] is roughly 1s [resp. 60s] for PKN4p0; and $\geq$220s [resp. $\geq$1000s] for PKN5p0.

Such good performances by *SAT and KSATC are due to early pruning, which has revealed to be very effective on these problem sets. For example, if we disable early pruning in *SAT and rerun it on PKN5p0, the system keeps exceeding the time limit for $65 \leq L \leq 200$.

When $p = 0\%$, the better behavior of *SAT and KSATC than DLP and TA, is confirmed by the $Q\%$-percentile graphs in Figure 5, corresponding to $N = 4$. Formally, the *$Q\%$-percentile* of a set S of values is the value V such that $Q\%$ of the values in S are smaller or equal to V. The median value of a set thus corresponds to the 50% percentile of the set. Figure 5 reports the 50%, 60%, 70%, 80%, 90% and 100% percentile values of the CPU-times when *SAT (top left), KSATC (top right), DLP (bottom left) and TA (bottom right) are run on PKN4p0. The percentile plots for PKN5p0 and PKN6p0 look similar to the plots in Figure 5. This means that all the systems perform in roughly the same way on the 50 most difficult samples of the 100 tests corresponding to a fixed N and L.

We also run the four systems on problems with $N = 4, 5, 6, 7$ while p is fixed to 50%. We call these problem sets PKN4p50, PKN5p50, PKN6p50, and PKN7p50. In Figure 6 the satisfiability percentages and the median of the CPU times for the four systems are plotted against the number of clauses L. As for $p = 0\%$, *SAT and KSATC perform roughly in the same way and are the fastest. Differently from the tests in which $p = 0\%$, at the transition point of 50% of satisfiable formulas, the gap between *SAT/KSATC and DLP seems to diminish when the number of variables increases. Horrocks and Patel-Schneider (1999a)

[10] DLP ver. 3.2 is able to successfully handle formulas with $N = 6$, $p = 0\%$ and $L > 150$.

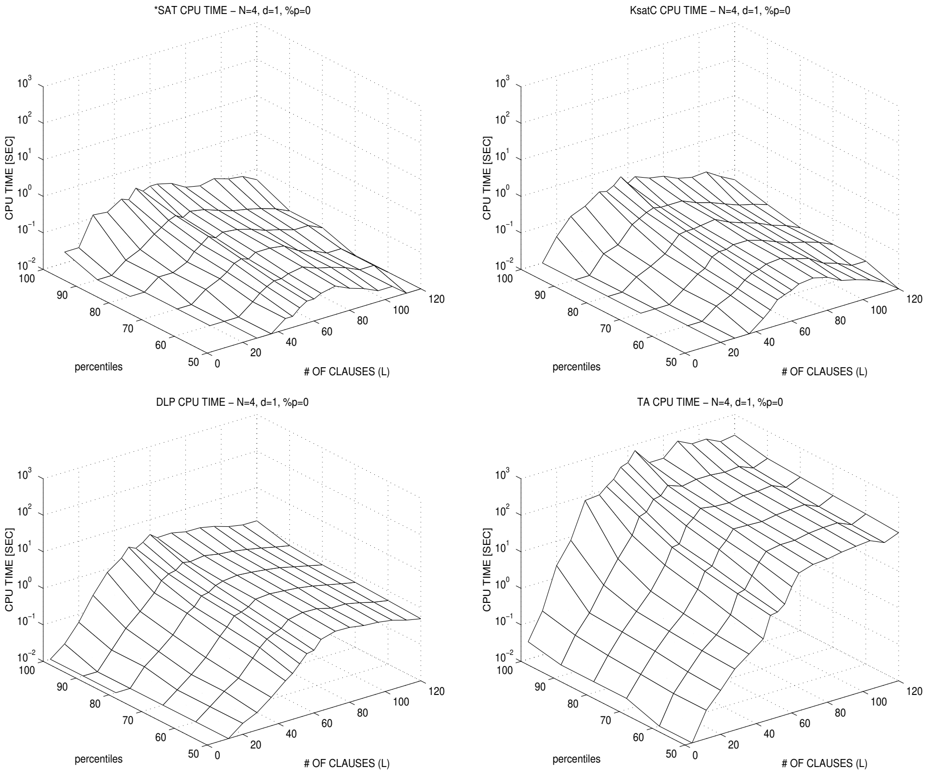

Figure 5. Logic K. 50%–100% percentile CPU times of *SAT, KSATC, DLP, and TA. $N = 4$. $p = 0\%$. 100 samples/point.

show that for values of L bigger than 7, DLP performances are superior to those of KSATC when $d = 1$ and $p = 50\%$. They also conclude that DLP performs better than KSATC when p is high and worse when p is low. We have not yet done such a broad comparison using *SAT instead of KSATC. However, we believe that Horrocks and Patel-Schneider's conclusions should extend also to *SAT, if *SAT is used with the parameter settings we have currently used, i.e., those which make *SAT most similar to KSATC. Of course, a big role can be played by *SAT already available configurable options, and this will be the issue of future research. In any case, both *SAT and KSATC perform better than the other systems for large values of L, when the formulas are trivially unsatisfiable. This is due to the fact that for large values of L, formulas become propositionally unsatisfiable, and thus both *SAT and KSATC mostly take advantage of their SAT-based nature, e.g., of their optimized data structures for handling large formulas.

As for $p = 0\%$, the percentile plots of the timings of the systems on PKN7p50, do not show big differences with respect to the plots of

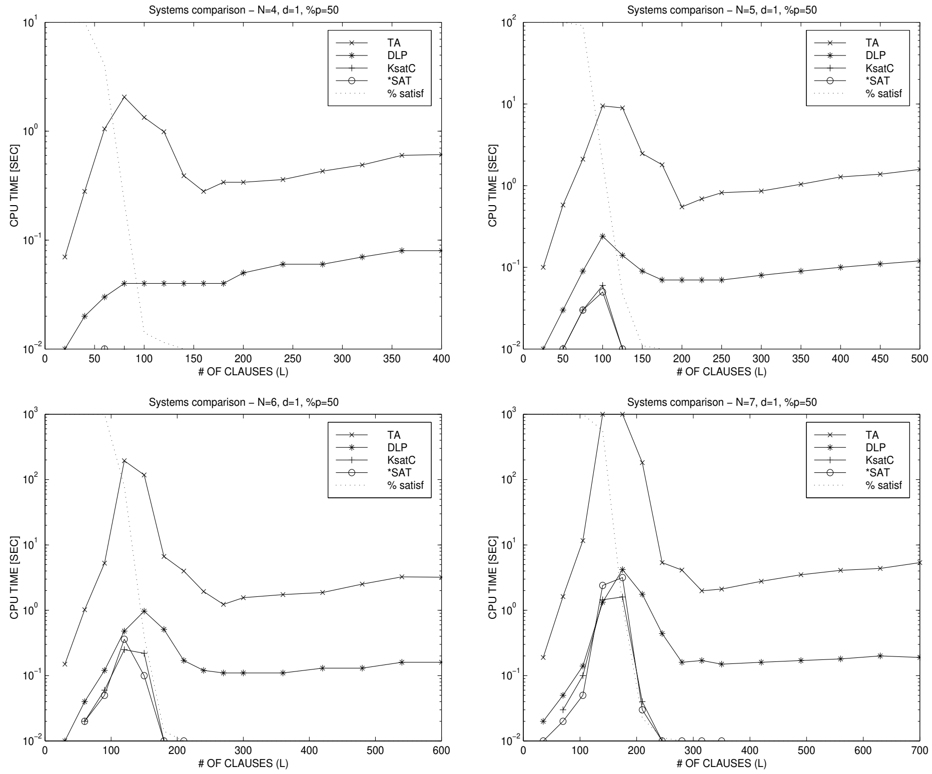

Figure 6. Logic K. *SAT, KSATC, DLP, and TA median CPU time. $N = 4, 5, 6, 7$. $p = 50\%$. 100 samples/point. Background: satisfiability percentage.

the medians. Considering DLP and *SAT 100% percentile plots, it is interesting to observe that DLP has a lower maximum than *SAT; on the other hand, *SAT values decrease more rapidly than those of DLP.

Finally, we see that for most of the problem sets we consider, all the systems seem to have an easy-hard-easy pattern, whose peak roughly correspond to the 50% of satisfiable formulas. When $p = 0\%$, this phenomenon is best evident for *SAT and KSATC.

We also consider the benchmarks formulas for K used at the Comparison of Theorem Provers for Modal Logics at Tableaux'98 (see (de Swart, 1998)). These consist of nine provable parameterized formulas (ending with "_p") and nine unprovable parameterized formulas (ending with "_n"). For each parameterized formula $A(n)$, the test consists in determining the greatest natural number $n \leq 21$ satisfying the following two conditions:

1. the prover returns the correct result for the formulas $A(1), A(2), \ldots, A(n)$ in less than 100 seconds, and

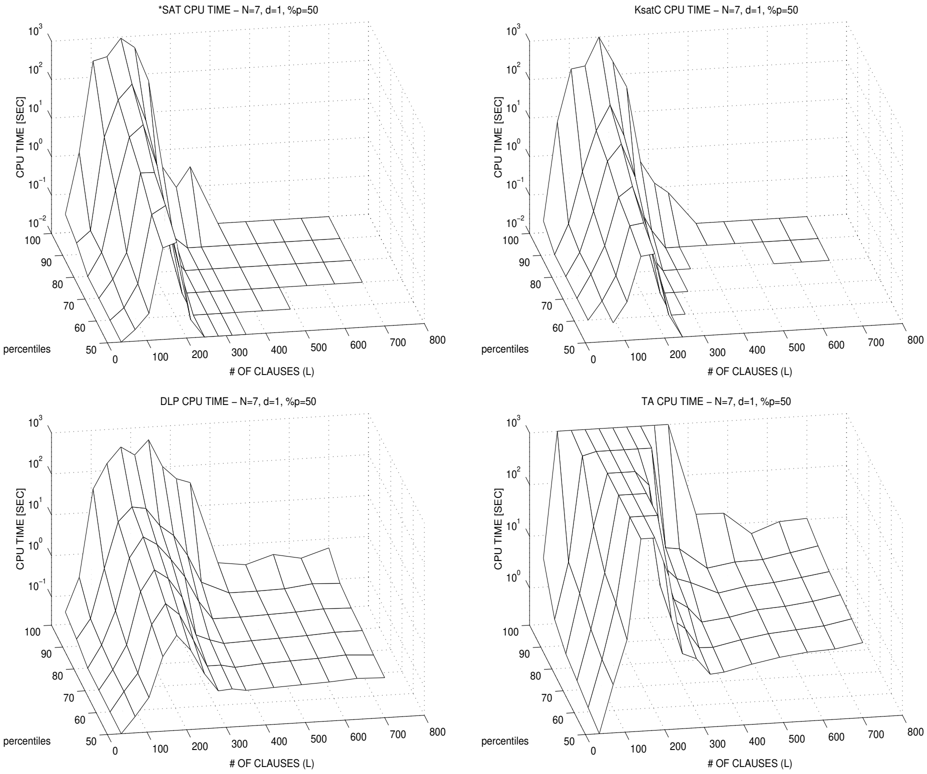

Figure 7. Logic K. 50%–100% percentile CPU times of *SAT, KSATC, DLP, and TA. $N = 7$. $p = 50\%$. 100 samples/point.

2. the prover cannot do the formula $A(n+1)$ in less than 100 seconds or $n = 21$.

Even though it has been proved that most of these tests can be easily solved by current solvers, these are still interesting because

- they are not 3CNF$_K$ formulas, and
- some of these tests have not been solved yet.

The results for *SAT, TA and DLP are reported in Table I. KSATC has not been tested since KSATC is able to deal with 3CNF$_K$ formulas only. We also show the CPU time requested by the system to solve the last instance $A(n)$. Notice that *SAT has been run with caching enabled, since the depth of all the formulas in the Table is greater than 1. For TA, we do not take into account the time needed to compute the first order formula $A^*(n)$ corresponding to $A(n)$ (which is negligible), but we do take into account the time requested by FLOTTER to convert $A^*(n)$

Table I. Logic K. *SAT, DLP and TA performances on Tableaux'98 benchmarks

	*SAT		DLP		TA		
Test	Size	Time	Size	Time	Size	SPASS	FLOTTER
k_branch_n	12	94.49	12	52.29	6	$\underline{84.21}$	12.23
k_branch_p	21	0.21	18	44.56	6	$\underline{51.95}$	13.81
k_d4_n	21	2.87	21	3.43	$\underline{14}$	1.14	42.92
k_d4_p	21	0.06	21	0.17	15	0.64	$\underline{70.47}$
k_dum_n	21	0.12	21	0.13	16	1.75	$\underline{64.07}$
k_dum_p	21	0.04	21	0.07	17	3.32	$\underline{61.67}$
k_grz_n	21	0.01	21	0.20	21	0.16	0.17
k_grz_p	21	0.04	21	0.11	21	0.35	0.16
k_lin_n	21	47.80	21	0.70	21	16.07	63.94
k_lin_p	21	0.01	21	0.08	21	1.03	8.21
k_path_n	21	0.96	21	1.29	4	$\underline{58.70}$	2.14
k_path_p	21	0.72	21	1.19	5	$\underline{22.85}$	2.18
k_ph_n	12	0.60	9	40.16	9	$\underline{45.21}$	9.92
k_ph_p	8	48.54	6	11.34	6	$\underline{42.19}$	0.97
k_poly_n	21	2.25	21	0.65	4	1.23	$\underline{7.86}$
k_poly_p	21	1.73	21	0.34	5	2.48	$\underline{51.00}$
k_t4p_n	21	1.28	21	0.45	9	3.37	$\underline{84.35}$
k_t4p_p	21	0.29	21	0.21	16	3.91	$\underline{84.75}$

into a set $Cl(A^*(n))$ of clauses (reported in the FLOTTER column), and the time requested by SPASS to determine the consistency or inconsistency of $Cl(A^*(n))$ (reported in the SPASS column). Furthermore, we stopped TA on $A(n)$ with $n < 21$, either because FLOTTER does not terminate gracefully when computing $Cl(A^*(n+1))$, or because SPASS or FLOTTER exceed the 100 seconds time limit. In the table, these three cases correspond to the rows in which the value for n/SPASS/FLOTTER respectively is underlined.

As can be observed from Table I, the three systems are able to solve all the instances of a formula in four cases. *SAT and DLP are able to solve all the instances except for k_branch_n, k_branch_p, k_ph_n, k_ph_p. Except for the first of these four parameterized formulas, *SAT is able to solve more instances than DLP. For k_branch_n, both *SAT and DLP are able to solve the 12th instance, with DLP taking less time than *SAT to solve it.

5.2. Modal E

Gasquet and Herzig (1996) provide a translation which maps any formula φ into a formula φ_{GH} in K_2, i.e., the smallest normal modal logic with two modal operators $\Box_1$ and $\Box_2$. The translation is such that φ is satisfiable in E iff φ_{GH} is satisfiable in K_2. This translation is defined in the following way:

- $\varphi_{GH} = \varphi$, if φ is a propositional variable,
- $\varphi_{GH} = \neg\Box_1\neg(\Box_2\psi_{GH} \wedge \Box_1\neg\psi_{GH})$, if $\varphi = \Box\psi$,

and homomorphic for the cases of the propositional connectives.

Consider a formula φ. We compare *SAT performances on φ with respect to *SAT, DLP and TA performances on φ_{GH}. We could not run KSATC on φ_{GH}, since KSATC accepts only 3CNF_K formulas with at most one modality. To make evident when a system is run using Gasquet and Herzig's translation, we append the string "+GH" to the name of the system. Therefore, in the following, we will have the systems *SAT, *SAT+GH, DLP+GH, and TA+GH.

In E, the 3CNF_K test methodology is not suited. Indeed, it is no longer the case that for any modal formula φ there exists a 3CNF_K formula which is E-satisfiable iff φ is E-satisfiable. Furthermore, checking the consistency of an assignment μ in E amounts to determine the consistency of $(\alpha \equiv \neg\beta)$ for each pair of conjuncts $\Box\alpha$ and $\neg\Box\beta$ in μ: most of these tests, in case α and β are 3CNF_K clauses, can be trivially satisfied.

We therefore consider sets of 3CNF_E formulas. A 3CNF_E formula is a conjunction of 3CNF_E clauses, each with three disjuncts. Each disjunct in a 3CNF_E clause is either a propositional literal or a formula having the form $\Box C$ or $\neg\Box C$, where C is a 3CNF_E formula. For example,

$$\Box((\Box(l_1 \vee l_2 \vee l_3) \vee l_4 \vee l_5) \wedge (l_6 \vee l_7 \vee l_8))$$

where each l_i $(1 \leq i \leq 8)$ is a propositional literal, is a 3CNF_E formula. For any formula φ, there exist a 3CNF_E formula logically equivalent to φ in E.

Sets of 3CNF_E formulas can be randomly generated according to the parameters used to generate 3CNF_K formulas, and a new parameter C representing the number of clauses at depth $d > 0$. A 3CNF_K is thus a 3CNF_E formula in which $C = 1$. As in the previous subsection, a problem set is characterized by N and p: d and C are fixed to 1 and L respectively; L is given increasing values in such a way to empirically cover the "100% satisfiable – 100% unsatisfiable" transition. We also

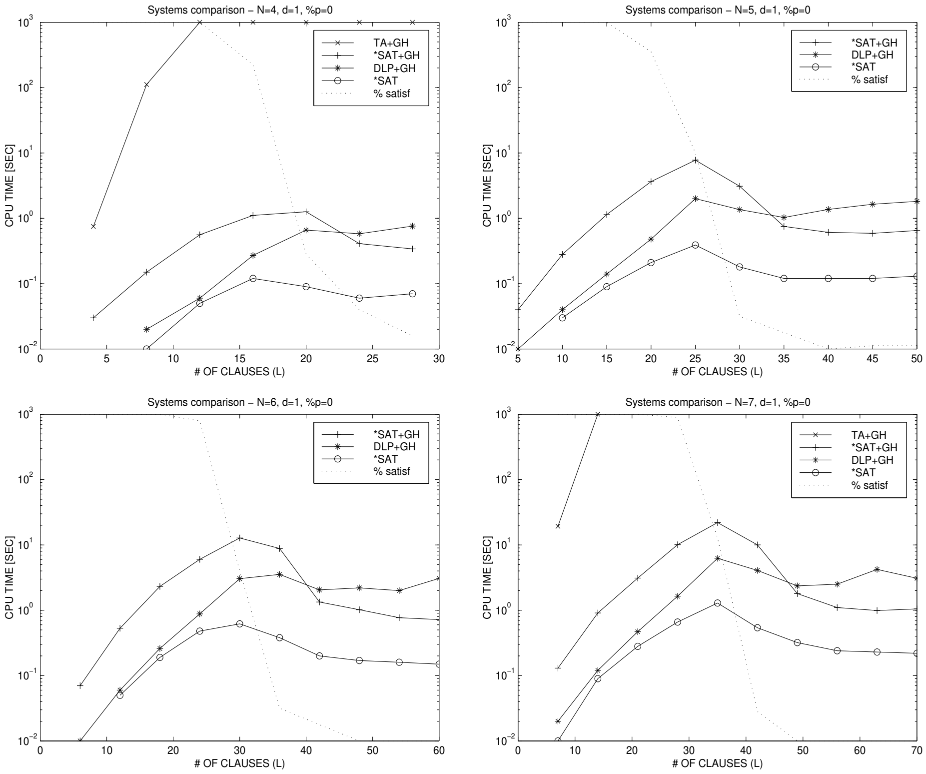

Figure 8. Logic E. *SAT, *SAT+GH, DLP+GH, and TA+GH median CPU time. $N = 4, 5, 6, 7$. $p = 0\%$. 100 samples/point. Background: satisfiability percentage.

check that in each sample there are no multiple occurrences of a formula in a clause, at the same ensuring that the propositional vs. the modal structure of the formula only depends on p. Notice that while increasing L also C is increased. As a consequence, for each pair of formulas $\Box\alpha_i$ and $\neg\Box\beta_j$ in an assignment satisfying a $3\mathrm{CNF}_E$ formula, the recursive E-consistency check for $(\alpha_i \equiv \neg\beta_j)$ has itself a phase transition from 100% satisfiable to 100% unsatisfiable when increasing L. Overall, for low [resp. high] values of L we expect that each satisfying assignment should be trivially determined to be E-consistent [resp. not E-consistent].

As before, for each value N in a problem set, 100 $3\mathrm{CNF}_E$ formulas are randomly generated, and the resulting formulas are given in input to the procedure under test. A timeout stops the execution of the system on a formula after 1000 seconds of CPU time. We consider the following problems sets:

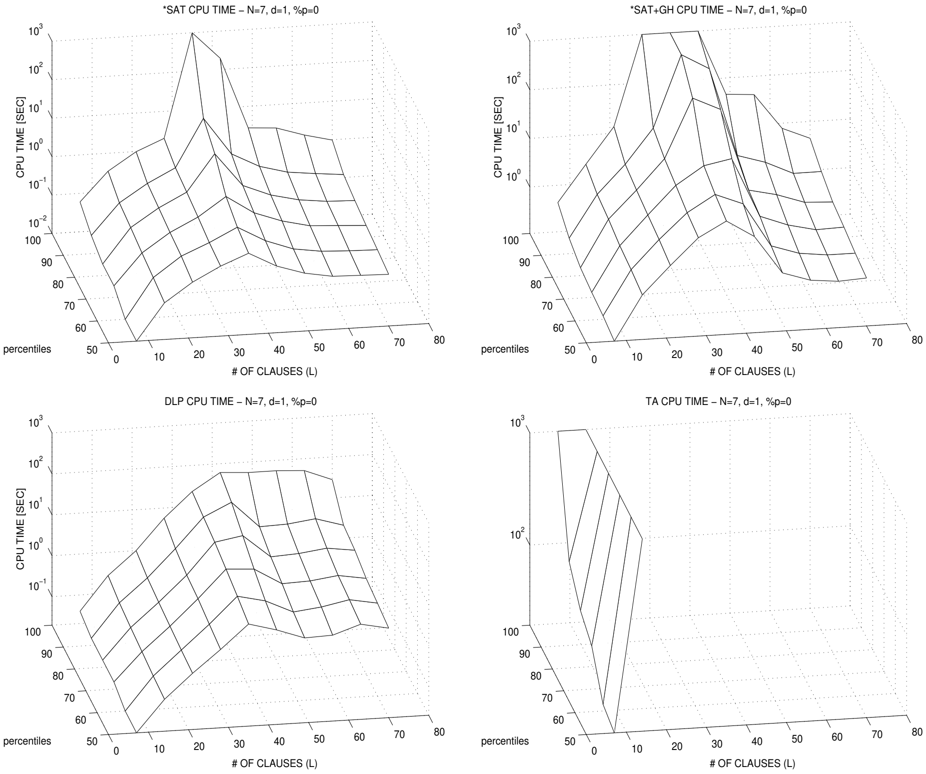

Figure 9. Logic E. 50%–100% percentile CPU times of *SAT, *SAT+GH, DLP+GH, and TA+GH. $N = 7$. $p = 0\%$. 100 samples/point.

- PEN4p0, PEN5p0, PEN6p0, PEN7p0 in which $p = 0\%$ while $L = 4, 5, 6, 7$ respectively, and
- PEN4p50, PEN5p50, PEN6p50, PEN7p50 in which $p = 50\%$ while $L = 4, 5, 6, 7$ respectively.

Given the huge amount of time that FLOTTER takes to prepare the formula for SPASS, we run TA+GH only on the problems sets PEN4p0 and PEN4p50. For PEN7p0 and PEN7p50, we run TA+GH only on the initial points. As in the preceding subsection, we only take into account the time the systems take for the main processing of the formula. In particular, for each system, we do not take into account the time needed to perform the Gasquet and Herzig's conversion; and for TA+GH we take into account only the time taken by SPASS. The median and the percentile plots of the systems on PEN4p0, PEN5p0, PEN6p0, PEN7p0 are shown in Figure 8 and Figure 9 respectively.

Consider Figure 8. As can be observed, *SAT is the fastest: the gap with the other systems is of more than one order of magnitude for

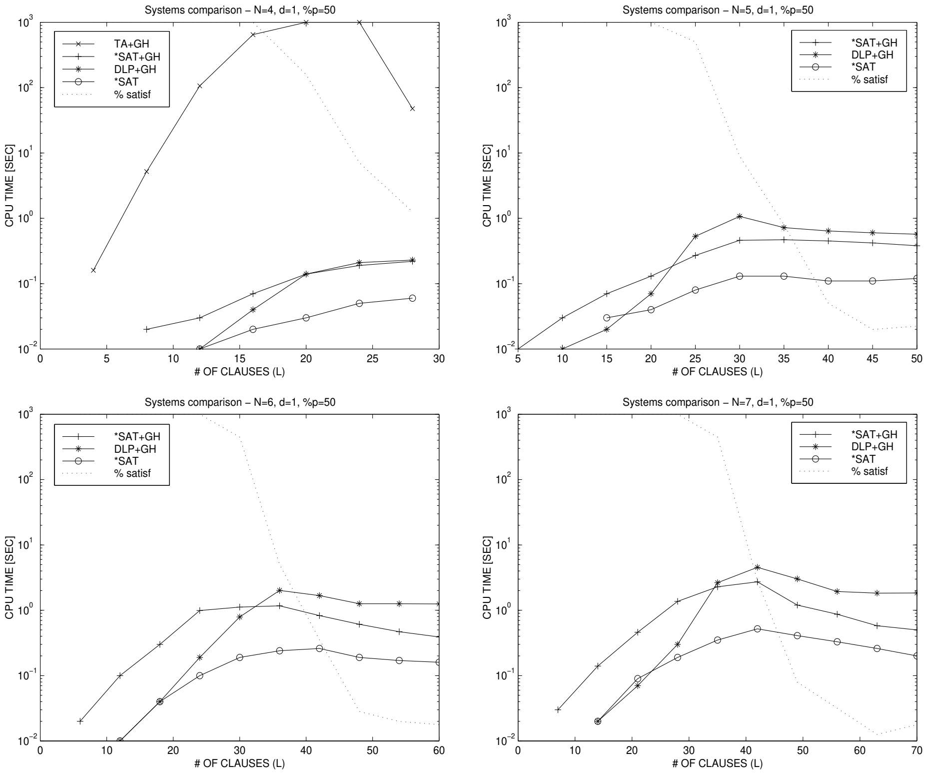

Figure 10. Logic E. *SAT, *SAT+GH, DLP+GH, and TA+GH median CPU time. $N = 4, 5, 6, 7$. $p = 50\%$. 100 samples/point. Background: satisfiability percentage.

certain values of L. However, both *SAT+GH and DLP+GH perform quite well, better than one could have imagined given that the consistency problem for E and K_2 belongs to two different complexity classes. However, a closer look to Gasquet and Herzig's reduction reveals that, considering a $3CNF_E$ formula φ, and an assignment $\mu = \bigwedge_{i=1}^{m} \Box\alpha_i \wedge \bigwedge_{j=1}^{n} \neg\Box\beta_j \wedge \gamma$ (as usual we assume that γ is a propositional formula) in the language of φ,

1. μ satisfies φ iff μ_{GH} satisfies φ_{GH}.

2. for checking the E-consistency of μ, *SAT performs at most $2mn$ consistency checks involving the formulas $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m$.

3. for checking the K_2-consistency of μ_{GH}, both *SAT+GH and DLP+GH perform at most $2mn$ consistency checks involving the formulas $\alpha_{1GH}, \ldots, \alpha_{mGH}, \beta_{1GH}, \ldots, \beta_{mGH}$.

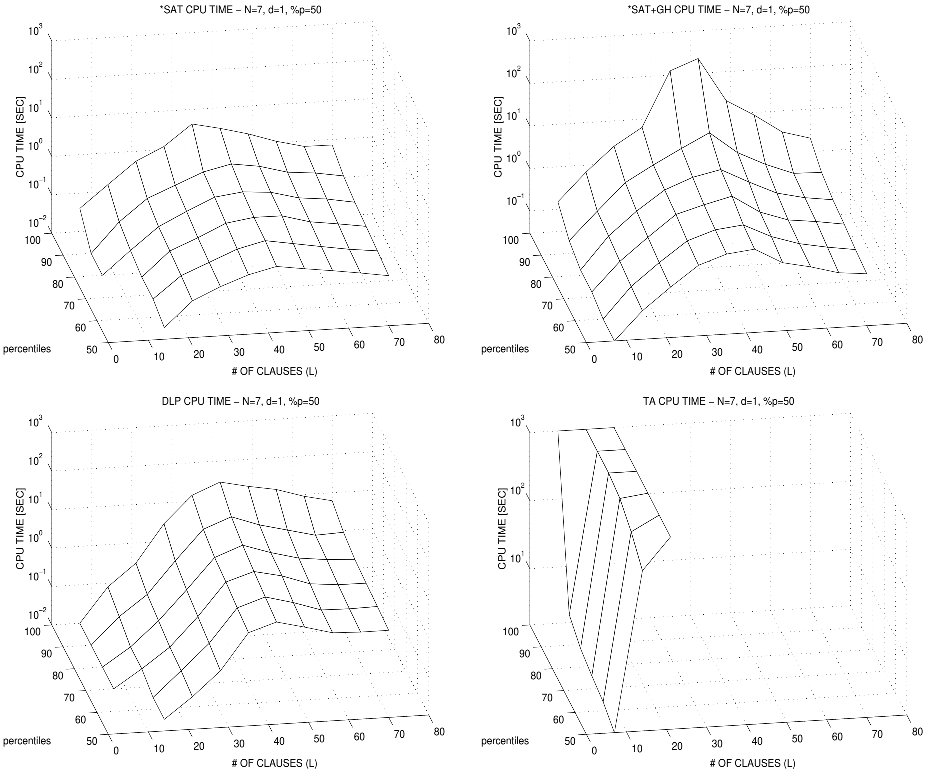

Figure 11. Logic E. 50%–100% percentile CPU times of *SAT, *SAT+GH, DLP+GH, and TA+GH. $N = 7$. $p = 50\%$. 100 samples/point.

The first two points are obvious. To understand the last, it suffices to notice that μ_{GH} is propositionally equivalent to

$$\bigwedge_{i=1}^{m} \neg\Box_1\neg(\Box_2\alpha_{iGH} \wedge \Box_1\neg\alpha_{iGH}) \;\wedge\; \bigwedge_{j=1}^{n} \Box_1(\neg\Box_2\beta_{jGH} \vee \neg\Box_1\neg\beta_{jGH}) \;\wedge\; \gamma.$$

Given that both *SAT+GH and DLP+GH use caching, these procedures will perform at most a quadratic number of checks in the number of subformulas of φ_{GH}. This is not the case for TA+GH, since SPASS does not have any caching mechanism. This explains the good behavior of *SAT+GH and DLP+GH, and the bad behavior of TA+GH.

It is interesting to compare *SAT+GH and DLP+GH performances. As can be observed, DLP+GH performs better than *SAT+GH for low values of L, but worse for high values of L. This behavior reflects the different mechanisms used by *SAT+GH and DLP+GH to prune the search space when checking the K-consistency of an assignment. As we said in Section 4, in all the tests we have set *SAT+GH as to use the early pruning strategy. DLP+GH instead implements a backjumping schema in the spirit of (Baker, 1995): when an assignment is discovered to be not K-consistent, backtracking to a point which does not lead to the same contradiction is enforced. While implementing early pruning does not introduce overheads, this is not the case for backjumping, where a dependency set of each derived clause has to be maintained (see (Patel-Schneider, 1998) for more details). Despite the additional costs introduced, backjumping clearly wins if compared to early pruning in logic K, for low values of L. In this case, almost each assignment is K-consistent and early pruning may cause additional (i.e., not performed by a backjumping strategy) checks. On the other hand, for high values of L, when the formula under test is not K-consistent but there are still assignments satisfying it, *SAT+GH is able to greatly cut off the search by checking the inconsistency of the assignment generated so far. DLP+GH instead checks the consistency of an assignment only when it satisfies the current formula. DLP+GH may therefore generate many assignments which, even though they satisfy the input formula, are not K-consistent. When L is so high that the input formulas become propositionally unsatisfiable, *SAT+GH may still perform additional K-consistency checks, but these get compensated by (*i*) *SAT+GH SAT-based nature and (*ii*) the costs DLP+GH has because of backjumping. Considering Figure 9, we see that DLP+GH has a better behavior than the other systems on some of the hardest instances. Evidently, on these tests, backjumping leads to a more uniform behavior than early pruning. Horrocks and Patel-Schneider (1999a) show that for some randomly generate $3\mathrm{CNF}_K$ formulas, early pruning leads to a more uniform behavior than backjumping.

For $N = 4, 5, 6, 7$ and $p = 50\%$, *SAT, *SAT+GH, DLP+GH and TA+GH median and percentile times are plotted in Figure 10 and Figure 11 respectively. As it can be observed, the situation is very similar to the case in which $p = 0\%$. The only difference is that now *SAT+GH performs better than DLP+GH for a lower value of L. This is reasonable, since for each L, the number of consistency checks performed by *SAT+GH because of early pruning, diminishes when p increases.

Finally, notice the easy-hard-easy pattern of *SAT. To better appreciate it, Figure 12 shows the number of calls to LSAT done by LCONSIST on PEN4p0-PEN7p0 (left), PEN4p50-PEN7p50 (right) against the ratio $\frac{L}{N}$ between L and N. As can be observed, *SAT performs a number of LSAT calls whose maximum roughly correspond to the 50% of satisfiable formulas. This transition happens when $\frac{L}{N}$ is close to 5 for $p = 0\%$ and to 6 for $p = 50\%$. This behavior reflects the above stated intuition

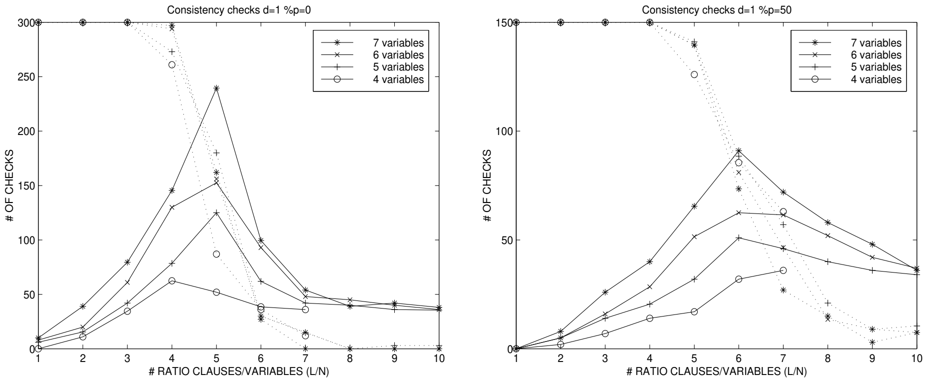

Figure 12. Logic E. *SAT median number of calls to LSAT. $N = 4, 5, 6, 7$. $p = 0\%$ (left) and $p = 50\%$ (right). 100 samples/point. Background: satisfiability percentages.

according to which for low [resp. high] values of L all the formulas should be easily determined to be E-consistent [resp. not E-consistent].

6. Conclusions and future work

We have presented a set of SAT-based decision procedures for eight classical modal logics. We have shown how the SAT-based approach allows for efficient and modular implementations for these logics. We have presented *SAT. *SAT is the only system that is able to deal with EN, EC, ECN and EMN. In the case of the logic E, we have defined a testing methodology which generalizes the $3\mathrm{CNF}_K$ methodology by Giunchiglia and Sebastiani (1996a), and which is suitable for testing systems for non-normal modal logics. The experimental evaluation shows that *SAT performances are superior to or comparable to the performances of other state-of-the-art systems.

In the future, we plan to conduct an extensive experimental analysis (similar to that presented in (Horrocks and Patel-Schneider, 1998; Horrocks and Patel-Schneider, 1999b)) to understand, for each class of formulas, which combination of *SAT options leads to the best results. We also plan to extend *SAT in order to handle more expressive decidable logics. We will also consider logics, like S4, for which more sophisticated methods than that described in Section 4 have to be employed in order to ensure the termination of the decision procedure.

Acknowledgments

We are grateful to Hullrich Hustadt, Peter F. Patel-Schneider, and Hantao Zhang for the assistance they provided on their systems. Special thanks to Roberto Sebastiani for the many useful discussions related to the subject of this paper. Thanks also to the anonymous reviewers for their helpful comments and suggestions. The first and last authors are partially supported by the Italian Spatial Agency.

References

F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

A. Baker. Intelligent backtracking on the hardest constraint problems. *Journal of Artificial Intelligence Research*, 1995.

M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proc. AAAI*, 1998.

B. F. Chellas. *Modal Logic – an Introduction*. Cambridge University Press, 1980.

M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

T. Boy de la Tour. Minimizing the Number of Clauses by Renaming. In *Proc. of the 10th Conference on Automated Deduction*, pages 558–572. Springer-Verlag, 1990.

H. de Swart, editor. *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence. Springer-Verlag, May 1998.

R. Fagin, J.Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about knowledge*. MIT Press, 1995.

M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishing, 1983.

E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors. *Collected Papers from the International Description Logics Workshop (DL'98)*. CEUR, May 1998.

Jon W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, 1995.

O. Gasquet and A. Herzig. From classical to normal modal logics. In Heinrich Wansing, editor, *Proof Theory of Modal Logics*, volume 2 of *Applied Logic Series*, pages 293–311. Kluwer Academic Publishers, 1996.

E. Giunchiglia and F. Giunchiglia. Ideal and Real Belief about Belief. Technical Report 96-0138, DIST, University of Genoa, Italy, September 1997. Also IRST-Technical Report 9605-04. Submitted for the publication to JLC, Journal of Logic and Computation. A short version of the paper appeared in the Proc. International Conference on Formal and Applied Practical Reasoning, (FAPR'96).

F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. CADE-*

96, Lecture Notes in Artificial Intelligence, New Brunswick, NJ, USA, August 1996. Springer Verlag.

F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for ALC. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996. Also DIST-Technical Report 9607-08 and IRST-Technical Report 9601-02.

E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 1998.

G. Governatori and A. Luppi. Labelled tableaux for non-normal modal logics. In *Sixth Conference of the Italian Association for Artificial Intelligence, AI*IA '99, Bologna, Italy, September 14–17, 1999: proceedings*, 1999.

I. Horrocks and P. F. Patel-Schneider. Comparing subsumption optimizations. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 90–94. CEUR, May 1998.

I. Horrocks and P. F. Patel-Schneider. Advances in propositional modal satisfiability, 1999. Manuscript.

I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 1999. To appear.

I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

U. Hustadt and R.A. Schmidt. On evaluating decision procedures for modal logic. In *Proc. of the 15th International Joint Conference on Artificial Intelligence*, 1997.

U. Hustadt and R.A. Schmidt. On evaluating decision procedures for modal logic. Research report MPI-I-97-2-003, Max-Planck-Institut für Informatik, Saarbrücken, Germany, February 1997.

Henry Kautz and Bart Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98*, 1998.

F. Massacci. Strongly analytic tableaux for normal modal logics. In *Proc. CADE*, 1994.

Richard Montague. Pragmatics. In R. Klibansky, editor, *Contemporary Philosophy: A Survey. I*, pages 102–122. La Nuova Italia Editrice, Florence, 1968. Reprinted in *Formal Philosophy*, by Richard Montague, Yale University Press, New Haven, CT, 1974, pp. 95–118.

H. J. Ohlbach. Translation methods for non-classical logics - an overview. *Bulletin of the Interest Group in Pure and Applied Logic - IGPL*, 1(1), 1988.

P. F. Patel-Schneider. DLP system description. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 87–89. CEUR, May 1998.

P. F. Patel-Schneider. Personal communication, June 1999.

D.A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

R. Sebastiani and F. Giunchiglia. From Tableau-based to SAT-based procedures - preliminary report. Technical Report 9711-14, IRST, Trento, Italy, November 1997.

Krister Segerberg. *An Essay in Classical Modal Logic.* Philosophical Studies, Uppsala, 1 edition, 1971.

Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

R. M. Smullyan. *First-Order Logic.* Springer-Verlag, NY, 1968.

Armando Tacchella. *SAT system description. In *Collected Papers from the International Description Logics Workshop (DL'99).* CEUR, July 1999.

G. Tseitin. On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8, 1970. Reprinted in (Siekmann and Wrightson, 1983).

Johan van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic*, volume 165 of *Synthese Library*, chapter II.4, pages 167–247. D. Reidel Publ. Co., Dordrecht, 1984.

Moshe Y. Vardi. On the complexity of epistemic reasoning. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 243–252, Asilomar Conference Center, Pacific Grove, California, 5–8 June 1989. IEEE Computer Society Press.

C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction (CADE-13)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 141–145, New Brunswick, New Jersey, USA, July/August 1996. Springer.

H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, July13–17 1997. Springer.