Constraint Methods for Modal Satisfiability

Sebastian Brand[1], Rosella Gennari[2], and Maarten de Rijke[3]

[1] CWI, Amsterdam, The Netherlands
Sebastian.Brand@cwi.nl
[2] ITC-irst, Trento, Italy
gennari@itc.it
[3] Language and Inference Technology Group, ILLC, U. of Amsterdam, The Netherlands
mdr@science.uva.nl

Abstract. Modal and modal-like formalisms such as temporal or description logics go beyond propositional logic by introducing operators that allow for a guarded form of quantification over states or paths of transition systems. Thus, they are more expressive than propositional logic, yet computationally better behaved than first-order logic. We propose constraint-based methods to model and solve modal satisfiability problems. We model the satisfiability of basic modal formulas via appropriate sets of finite constraint satisfaction problems, and then resolve these via constraint solvers. The domains of the constraint satisfaction problems contain other values than just the Boolean 0 or 1; for these values, we create specialised constraints that help us steer the decision procedure and so keep the modal search tree as small as possible. We show experimentally that this constraint modelling gives us a better control over the decision procedure than existing SAT-based models.

1 Introduction

In many areas of artificial intelligence and computer science, trees, graphs, transition systems, and other types of relational structures provide the natural mathematical means to model evolving systems or to encode information about such systems. One may have to deal with such structures for a variety of reasons, e.g., to evaluate queries, to check requirements, or to make implicit information explicit. Modal and modal-like logics such as temporal logic and description logic [6] provide a convenient and computationally well-behaved formalism in which such reasoning may be represented [12].

Driven by the increased *computational* usage and usefulness of modal and modal-like logics, the past decade has seen a wide range of initiatives aimed at developing, refining, and optimising algorithms for solving the satisfiability problem of basic modal logic. This has resulted in a series of implementations. Some of these implement special purpose algorithms for modal logic, while others exploit existing tools or provers for either first-order logic or propositional logic through some encoding. In this paper we follow the second approach: we put

K.R. Apt et al. (Eds.): CSCLP 2003, LNAI 3010, pp. 66–86, 2004.
© Springer-Verlag Berlin Heidelberg 2004

forward a proposal to model and solve the modal satisfiability problem as a set of constraint satisfaction problems.

Specifically, we stratify a modal satisfiability problem, which is PSPACE-complete, into layers of "simpler" constraint satisfaction problems, which are NP-complete. On top of this, we add a refinement that exploits the restricted syntactic nature of modal problems, and that enables us to make efficient use of existing constraint solvers to decide modal satisfiability. Using the constraint logic programming system ECL^{i}PS^e [27], we inherit for free all the solvers for finite constraint satisfaction problems (e.g., generalised arc-consistency plus backtracking) and primitive constraints (e.g., at_most_one) already implemented in ECL^{i}PS^e; hence we can run them "as is" on top of our modelling to decide modal satisfiability. While we cannot yet fully compete with today's highly optimised modal provers, our experimental evaluations suggest that the approach is very promising in general, and even excellent in some cases.

The main contributions of our work derive from our modelling of modal satisfiability problems: modal formulas are translated into layers of finite constraint problems that have domains with possibly *further values than the Boolean* 0 or 1 (see Section 4), together with *appropriate constraints* to reason about these values (see Section 5). As amply shown and discussed in Sections 5 and 6 below, our modelling has a number of benefits over existing encodings of modal formulas into sets of propositions. For instance, the extended domains together with appropriate constraints give us a *better control over the modal search procedure*: they allow us to set strategies on the variables to split on in the constraint solver in a compact manner. In particular, by means of appropriate constraints and heuristics for our model, we can obtain partial Boolean assignments instead of total assignments; see Subsection 5.2.

The rest of the paper is organized as follows. After having provided background material concerning the motivation of the work reported here and work related to ours in Section 2, we lay the propositional groundwork in Section 3. We turn to modal matters in Section 4. The main contributions of this paper start in Section 5, which presents our constraint model. Then in Section 6, we report on an experimental assessment on a benchmark test set used in the TANCS '98 comparison of provers for modal logic. We conclude in Section 7.

2 Background

In this section we address two aspects of our work. First, we provide some motivations for studying the satisfiability problem for modal and modal-like logic. And second, we relate our approach to existing work in the literature.

Motivations. We have a broad view of what modal logic is. On this view, modal logic encompasses such formalisms as temporal logic, description logic, feature logic, dynamic logic... While originating from philosophy, for the past three decades the main innovations in the area of modal logic have come from computer science and artificial intelligence. The modern, computationally motivated

view of modal logic is one that takes modal logics to be expressive, yet computationally well-behaved fragments of first-order or second-order logic. Other computer science influences on modal logic include the introduction of many new formalisms, new algorithms for deciding reasoning tasks, and, overall, a strong focus on the interplay between expressive power and computational complexity. We now give examples of modern computational uses of modal-like logics.

We start with a brief look at the use of modal-like logics in the area of formal specification and verification; see [18] for a comprehensive introduction. Requirements such as "the system is `always` dead-lock free" or "the system `eventually` waits for a signal" can be compactly expressed in the basic modal logic by augmenting propositional logic with two operators: $\Box$ for the guarded universal quantifier over states (commonly read as `always`, meaning "in all the reachable states"), and $\Diamond$ for its existential counterpart (commonly read as `eventually`, meaning "in some reachable state"). Formalising "the system is dead-lock free" with the proposition s_free and "the system waits for a signal" with s_wait, the above two requirements correspond to the modal formulas $\Box s_free$ and $\Diamond s_wait$, respectively.

So-called multi-modal logics are popular in the agent-based community (e.g., see [24]); here, each agent is endowed with beliefs and knowledge, and with goals that it needs to meet. The beliefs and knowledge can be expressed by means of multi-modal operators: $\Box^b_A$ for "agent A believes" and $\Diamond^b_B$ for "agent B disbelieves"; $\Box^k_B$ for "agent B knows" and $\Diamond^k_A$ for "agent A ignores". More complex modal formulas involving until operators or path quantifiers are used to reason about agents' plans, in particular to express and verify specifications on plans (see, e.g., [5]) or extended goals (see, e.g., [23]).

Description logics are a family of modal-like logics that are used to represent knowledge in a highly structured manner [3], using (mostly) unary and binary relations on a domain of objects. Knowledge is organized in terminological information (capturing definitions and structural aspects of the relations) and assertional information (capturing facts about objects in the domain being modelled). For instance, an object satisfies $\Diamond_R A$ if it is R-related to some object satisfying A. In the area of description logic, a wide range of algorithms has been developed for a wide variety of reasoning tasks.

While there are many more areas where modal-like logics are currently being used, including semi-structured data [19], game theory [13], or mobile systems [7], due to space limitations we have to omit further details. What all of these computational applications of modal-like logics have in common is that they use relational structures of one kind or another to model a problem or domain of interest, and that a modal-like logic is used to reason about these structures. Moreover, for many of the above applications, *modal satisfiability checking* is the appropriate reasoning task: given a modal formula, is there a model on which this formula can be satisfied? In this paper we propose a new, constraint-based method for checking modal satisfiability.

Related Work. The past decade has seen a wide range of initiatives aimed at developing, refining, and optimising algorithms for solving the satisfiability prob-

lem of basic modal logic. Some of these implement special purpose algorithms for modal logic, such as `DLP` [22], `FaCT` [15], `RACER` [11], *`SAT` [25], while others exploit existing tools or provers for either first-order logic (`MSPASS` [20]) or propositional logic (`KSAT` [10], `KBDD` [21]) through some encoding. In this paper we follow the second approach: we propose to model and solve modal satisfiability problems as constraint problems.

The starting-points of our work are [10] and [2]. In [10], modal formulas are modelled and solved as sets of propositions (i.e., Boolean formulas) stratified into layers; the propositions are processed starting from the top layer in a depth-first left-most manner.

But we add a refinement that builds on ideas due to [2]. In [2] a refinement of an existing encoding of modal formulas into first-order formulas was introduced. This refinement enables one to re-use existing first-order theorem provers for deciding modal satisfiability, and, at the same time, to 'inform' the prover about the restricted syntactic nature of first-order translations of modal formulas, which resulted in a significant improvement in performance. We build on this intuition: we improve on the modelling of modal formulas in [10] so as to be able to make efficient use of existing constraint solvers to decide modal satisfiability. Specifically, modal formulas are translated into layers of finite constraint satisfaction problems that have domains with possibly further values than the Boolean 0 or 1, together with appropriate constraints to reason about these values. The well-known DPLL algorithm can also return partial Boolean assignments for propositions. But, in this respect, there are two key add-ons of our modelling. First, the use of extended domains and constraints allow us more control over the partial assignments to be returned by the adopted constraint solver than unit propagation allows for in DPLL. And second, we can run any constraint solver on top of our modelling to obtain partial assignments, i.e., it is by modelling that we obtain partial assignments, and not by modifying existing constraint solvers nor by choosing a specific solver to do so such as DPLL.

3 Propositions as Finite Constraint Problems

Constraint Satisfaction Problems. We begin with constraint satisfaction terminology. Consider a set $X := \{x_1, \ldots, x_n\}$ of n variables, and assume that X is ordered by $\prec$; a *scheme* of X is a sequence $s : x_1, \ldots, x_m$ of variables in X, where $x_{j-1} \prec x_j$ for each $j = 2, \ldots, m$. Associate one set D_i with each variable $x_i \in X$; then D_i is the *domain of* x_i; let $\boldsymbol{D}$ be the set of all such domain and variable pairs $\langle D_i, x_i \rangle$. Given a scheme $s := x_1, \ldots, x_m$, a relation $C(s)$ on the Cartesian product $\prod_{j=1}^{m} D_j$ is a *constraint on* s; let $\boldsymbol{C}$ be a set of constraint and scheme pairs $\langle C(s), s \rangle$ on X. Then $\langle X, \boldsymbol{D}, \boldsymbol{C} \rangle$ is a *constraint satisfaction problem* (CSP). A CSP is *finite* if all D_i in $\boldsymbol{D}$ are so. A tuple $d \in D_1 \times \cdots \times D_n$ is *consistent* or *satisfies* a constraint $C(s)$ if the projection of d on s, denoted by $\Pi_s(d)$, is in $C(s)$; if d satisfies all the constraints of the CSP P, then P is a *consistent* or *satisfiable CSP*. The projection of a constraint $C(t)$ over a subscheme s of t is denoted by $\Pi_s(C(t))$. Finally, a *total assignment for a CSP* $\langle X, \boldsymbol{D}, \boldsymbol{C} \rangle$ is a

function $\mu : X \mapsto \bigcup_{i=1}^{n} D_i$ that maps each $x_i \in X$ to a value in the domain D_i of x_i; μ *satisfies the CSP* if the tuple $(\mu(x_1), \ldots, \mu(x_n))$ does so.

Propositions. When a Boolean-valued assignment μ satisfies a propositional formula ϕ, we write $\mu \models \phi$. We write $CNF(\phi)$ for the result of ordering the propositional variables in ϕ and transforming ϕ into a conjunctive normal form: i.e., a conjunction of disjunctions of literals without repeated occurrences; a *clause* of ψ is a conjunct of $CNF(\psi)$.

From Propositions to CSPs. It is not difficult to transform a propositional formula into a CSP so that this is satisfiable iff the formula is: first the formula is transformed to its CNF; then each resulting clause is considered as a constraint. E.g., the CNF formula

$$(\neg x \vee y \vee z) \wedge (x \vee \neg y) \tag{1}$$

is the CSP with variables x, y and z, domains equal to $\{0, 1\}$, and two constraints: $C(x, y, z)$ for $\neg x \vee y \vee z$, that forbids the assignment $\{x \mapsto 1, y \mapsto 0, z \mapsto 0\}$; and the constraint $C(x, y)$ for $x \vee \neg y$ to rule out the assignment $\{x \mapsto 0, y \mapsto 1\}$. In [28] the encoding in (1) is used to prove that a version of forward checking performs more inferences than the basic DP procedure for deciding propositional satisfiability.

However, a constraint solver returns a total assignment given the modelling of formulas as CSPs above, while we aim at *partial* Boolean assignments. For example, a partial assignment of only two variables suffices to satisfy the formula in (1), such as $\{x \mapsto 1, z \mapsto 1\}$. How do we get such without modifying the underlying constraint solver? One way is to encode the propositional formula into a CSP with values other than 0 and 1. The additional values are then used to mark variables that the solver does not need to satisfy (yet). Let us give a precise definition of this new encoding. We assume an implicit total order on the variables in the considered propositions; so, we identify formulas that only differ in the order of occurrence of their atoms, such as $y \vee x$ and $x \vee y$.

Definition 1. Given a propositional formula ψ, $CSP(\psi)$ is the CSP associated with ψ defined as follows:

1. construct $\psi' := CNF(\psi)$ and let X be the ordered set of propositional variables occurring in ψ';
2. create a domain $D_i := \{0, 1, \mathsf{u}\}$ for each x_i in X;
3. for each clause θ in ψ', there is a constraint C_θ on the scheme $s := x_1, \ldots, x_m$ of all the variables of θ; a tuple $d := (d_1, \ldots, d_m)$ in $\prod_{j=1}^{m} D_j$ satisfies C_θ iff there is a non-empty subscheme $s' := x_{i_1}, \ldots, x_{i_n}$ of s such that $d_{i_k} \in \{0, 1\}$ for all $k = 1 \ldots n$ and $d' := \Pi_{s'}(d)$ satisfies $\Pi_{s'}(C_\theta)$.

In Definition 1, we do not give any details on how constraints are represented and implemented; this is done on purpose, since these are not necessary for our theoretical results concerning the modal satisfiability solver. Nevertheless, some modelling choices and implementation details are discussed in Section 5 below.

Our modelling of propositional formulas as in Definition 1 allows us to make *any complete solver for finite CSPs return a partial Boolean assignment* that satisfies a propositional formula ψ iff ψ is satisfiable. To prove this, we need some notational shorthands. Given $CSP(\psi)$ on X as in Definition 1 above, let μ be a total assignment for $CSP(\psi)$, and $X|_{Bool}$ the subset of all $x_i \in X$ for which $\mu(x_i) \in \{0, 1\}$. Then the restriction of μ to $X|_{Bool}$ is denoted by $\mu|_{Bool}$: i.e., $\mu|_{Bool} : X|_{Bool} \mapsto \{0, 1\}$ and $\mu|_{Bool}(x_i) = \mu(x_i)$ for each $x_i \in X|_{Bool}$.

Theorem 1. *Consider a propositional formula ψ and let X be its ordered set of variables.*

1. *a total assignment μ for $CSP(\psi)$ satisfies $CSP(\psi)$ iff $\mu|_{Bool}$ satisfies ψ;*
2. *ψ is satisfiable iff a complete constraint solver for finite CSPs returns a total assignment μ for $CSP(\psi)$ such that $\mu|_{Bool}$ satisfies ψ.*

Proof. First notice that a proposition and its CNF are equivalent; in particular, a Boolean assignment satisfies the one iff it satisfies the other. Then item 1 follows from this, Definition 1 and a property of CNF formulas: a partial Boolean assignment μ satisfies $CNF(\psi)$ iff, for each clause ϕ of $CNF(\psi)$, μ assigns 1 to at least one positive literal in ϕ, or 0 to at least one negative literal in ϕ. Item 2 follows from the former. •

Note 1. It is sufficient that each domain D_i of $CSP(\psi)$ contains the Boolean values 0 and 1 for the above result to hold. Thus, one could have values other than u (and 0 and 1) in the CSP modelling to mark some variables with different "levels of importance" for deciding the satisfiability of a propositional formula. However, our choice as in Definition 1 will suffice for the purposes in this paper.

4 Modal Formulas as Layers of Constraint Problems

In this section we recall the basics of modal logic and provide a link between solving modal satisfiability and CSPs.

4.1 Modal Formulas as Layers of Propositions

We refer to [6] for extensive details on modal logic. To simplify matters, we will focus on the basic mono-modal logic $\mathcal{K}$, even though our results can easily be generalized to a multi-modal version.

Modal Formulas. $\mathcal{K}$-formulas are defined as follows. Let P be a finite set of propositional variables. Then $\mathcal{K}$-formulas over P are produced by the rule

$$\phi \quad ::= \quad p \mid \neg\phi \mid \phi \wedge \phi \mid \Box\phi$$

where $p \in P$. The formula $\Diamond p$ abbreviates $\neg\Box\neg p$, and the other Boolean connectives are explained in terms of $\neg, \wedge$ as usual. For instance, all of p, q, $p \vee q$, $\Box(p \vee \Box q) \wedge \Diamond\Box p$ are $\mathcal{K}$-formulas over $\{p, q\}$. A formula of the form $\Box\phi$ is called a *box formula*.

Note 2. Here and in the remainder, we always assume that P is implicitly ordered to avoid modal formulas only differing in the order of their propositional variables; also, standard propositional simplifications such as the removal of double occurrences of $\neg$ are implicitly performed on modal formulas.

Modal Layers and Propositional Approximations. The satisfiability procedure for $\mathcal{K}$-formulas in this paper (see Subsection 4.2 below) revolves around two main ideas:

- the stratification of a modal formula into layers of formulas of decreasing "modal depth";
- the "approximation" and resolution of such formulas as propositions.

Let us make those ideas more precise, starting with the former. The "modal depth" of a formula counts its maximum number of nested boxes, that is it measures "how deeply" we can descend into the formula by peeling off nested boxes. Formally, the *modal depth* of ϕ, denoted by $\mathrm{md}(\phi)$, is defined as follows:

$$\mathrm{md}(p) := 0 \qquad \mathrm{md}(\Box\phi) := \mathrm{md}(\phi) + 1$$
$$\mathrm{md}(\neg\phi) := \mathrm{md}(\phi, i) \qquad \mathrm{md}(\phi_1 \wedge \phi_2) := \max\{\mathrm{md}(\phi_1), \mathrm{md}(\phi_2)\}.$$

For instance, consider $\phi = \Box p \vee \neg q$ which intuitively means that "`always` p or q fails." Then $\mathrm{md}(q, \phi) = 0$, $\mathrm{md}(\Box p) = \mathrm{md}(p) + 1 = 1$, hence $\mathrm{md}(\phi) = 1$; in other words, the maximum number of nested boxes we can peel off from ϕ is 1. Testing if a modal formula is satisfiable involves stratifying it into layers of subformulas (or Boolean combinations of these) of decreasing modal depth. At each such layer, modal formulas get "approximated" and solved as propositions. Formally, given a modal formula ϕ, the *propositional approximation* of ϕ, denoted by $\mathit{Prop}(\phi)$, is the proposition inductively defined as follows:

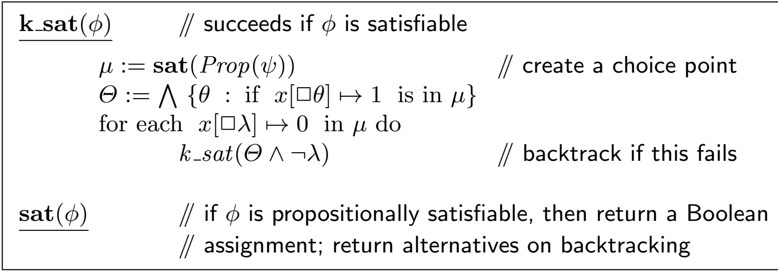
$$\mathit{Prop}(p) := p \qquad \mathit{Prop}(\Box\phi) := x_i[\Box\phi]$$
$$\mathit{Prop}(\neg\phi) := \neg\mathit{Prop}(\phi) \qquad \mathit{Prop}(\phi_1 \wedge \phi_2) := \mathit{Prop}(\phi_1) \wedge \mathit{Prop}(\phi_2).$$

We denote here by $x_i[\Box\phi]$ a fresh propositional variable that is associated with one occurrence of $\Box\phi$. Note that in this way different occurrences of $\Box\phi$ are distinguished by introducing different new variables. For instance: the modal formula $\phi = p \wedge \Box q \vee \neg\Box q$ is approximated by the proposition $\mathit{Prop}(\phi) = p \wedge x_1[\Box q] \vee \neg x_2[\Box q]$. The variables of ϕ are $\{p, q, r\}$, while $\mathit{Prop}(\phi)$ is over the variables $\{p, x_1[\Box q], x_2[\Box q]\}$.

Now that the ideas of modal depth and approximation of modal formulas as propositions are made precise, we can put them to work in the $\mathcal{K}$-satisfiability procedure below.

4.2 $\mathcal{K}$-Satisfiability and the General *k_sat* Schema

In Figure 1 below, we formalise $\mathcal{K}$-satisfiability and present the general algorithm schema *k_sat*, on which `KSAT` [10] is based, for deciding the satisfiability of $\mathcal{K}$-formulas.



```
k_sat(φ)      // succeeds if φ is satisfiable
        μ := sat(Prop(ψ))                         // create a choice point
        Θ := ⋀ {θ : if x[□θ] ↦ 1 is in μ}
        for each x[□λ] ↦ 0 in μ do
                k_sat(Θ ∧ ¬λ)                     // backtrack if this fails

sat(φ)        // if φ is propositionally satisfiable, then return a Boolean
              // assignment; return alternatives on backtracking
```

Fig. 1. The *k_sat* algorithm schema.

$\mathcal{K}$-Satisfiability. At this point we have to make a choice between a more "standard" characterisation of the semantics of $\mathcal{K}$-formulas or another that is closer to the semantics of the solving algorithm. Here we choose for the latter which allows us to arrive more quickly and concisely at the matters of this paper.

Definition 2. The $\mathcal{K}$-formula ϕ is *$\mathcal{K}$-satisfiable* iff there is a Boolean assignment μ that satisfies $Prop(\phi)$, and for every $\Box\lambda$ with $\mu(x[\Box\lambda]) = 0$, the $\mathcal{K}$-formula

$$\neg\lambda \wedge \bigwedge \{\theta \;:\; \mu(x[\Box\theta]) = 1\}$$

is $\mathcal{K}$-satisfiable.

Algorithm Schema. In the *k_sat* schema, the *sat* procedure determines the satisfiability of the propositional approximation of ϕ by returning a Boolean assignment μ as in Definition 2. Alternative satisfying assignments are generated upon backtracking. If there is no alternative assignment, then this call to *k_sat* fails and backtracking takes place, except on the input formula where "formula is unsatisfiable" is reported. In this manner, the modal search space gets stratified into modal formulas of decreasing modal depth and explored in a depth-first manner. A variable of the form $x[\Box\lambda]$ to which μ assigns 0 means that we *must* "remove the box" and check λ against all the formulas θ that come with variables of the form $x[\Box\theta]$ to which μ assigns 1; precisely one proposition is so created and tested satisfiable.

Theorem 2. *In the k_sat algorithm schema in Figure 1, if sat is a complete solver for Boolean formulas, then k_sat is a decision procedure for $\mathcal{K}$-satisfiability.*

Proof. The characterisation of $\mathcal{K}$-satisfiability in Definition 2 is responsible for the correctness and completeness of *k_sat*; this terminates since the modal depth and the number of propositional variables of a modal formula are bounded. •

4.3 The KCSP Algorithm

We now devise a modal decision procedure based on the *k_sat* schema, but with a constraint solver as the underlying propositional solver *sat*. We first provide the reader with an example, and then formalise the procedure.

Example 1. Consider the following modal formula

$$\phi \;=\; \neg\Box(p \vee \bot) \;\wedge\; (\Box r \vee \Box p),$$

which intuitively means "it is never that p fails or that false holds, and it is `always` r or it is `always` p". Approximating ϕ as the proposition $Prop(\phi)$, the following CSP is obtained:

(a) three variables: $x[\Box(p \vee \bot)]$; $x[\Box r]$; $x[\Box p]$;
(b) three domains, all equal to $\{0, 1, \mathsf{u}\}$;
(c) two constraints: the one for $\Box(p \vee \bot)$ that forces the assignment 0 to $x[\Box p \vee \bot]$; the other for $(\Box r \vee \Box p)$ that requires 1 to be assigned to $x[\Box r]$ or $x[\Box p]$.

Assigning the value u to a variable means not committing to any decision concerning its Boolean values, 0 and 1. The above CSP is given to the constraint solver, and this may return the assignment

$$\mu = \{x[\Box(p \vee \bot)] \mapsto 0, x[\Box r] \mapsto \mathsf{u}, x[\Box p] \mapsto 1\}.$$

Then, for all the variables $x[\Box \ldots]$ to which μ assigns 1 (in this case only $x[\Box p]$), the formulas within the scope of $\Box$ are joined in a conjunction Φ, in this case

$$\Phi := p. \tag{UT}$$

Then all the box variables to which μ assigns 0 are considered, in this case only $x[\Box(p \vee \bot)]$; thus $p \vee \bot$ gets negated, simplified (translated in CNF when needed) and the result is the formula

$$\Theta := \neg p. \tag{ET}$$

The conjunction $\Phi \wedge \Theta$ is given to the *sat* solver; in this case, the clause that is passed on is $p \wedge \neg p$. This is translated into a new CSP and its inconsistency is determined. On subsequent backtracking, we may obtain μ' instead of the above assignment μ:

$$\mu' = \{x[\Box(p \vee \bot)] \mapsto 0, x[\Box r] \mapsto 1, x[\Box p] \mapsto \mathsf{u}\}.$$

The new (UT) $\Phi := r$ is created; the satisfiability of $\neg p \wedge r$ is determined, and thus that of ϕ. •

Notice the key points about (UT) and (ET): we *only* consider the box variables $x[\Box \ldots]$ to which a Boolean value, 0 or 1, is assigned. The box variables to which u is assigned are disregarded, safely so because of Theorem 1. As we will see below (Section 5), the availability of values other than 0 and 1 has a number of advantages.

As the above example illustrates, we first approximate ϕ as a proposition and then translate this into a CSP. Recall from Definition 1 how the CSP of a proposition ϕ is obtained: $CSP(\phi)$ is the CSP with domains containing another value than the Boolean 0 and 1.

Definition 3. The KCSP *algorithm* is defined as follows. In the *k_sat* schema we instantiate the *sat* function with a complete solver for finite CSPs and we preprocess ϕ into $CSP(Prop(\phi))$ before passing it on to the constraint solver.

Theorems 1 and 2 yield the following result concerning KCSP as in Definition 3.

Corollary 1. KCSP *is a decision procedure for $\mathcal{K}$-satisfiability.* •

In particular, notice again that modelling $Prop(\phi)$ as a CSP with an additional non-Boolean value allows us to instantiate *sat* to *any* constraint solver in KCSP and still obtain *partial* Boolean assignments.

5 Constraint-Based Modelling

In this section we discuss the constraints into which we translate a modal formula. We begin with a base modelling, and proceed to an improved modelling that possesses some desirable properties.

5.1 A Base Modelling

The input to KCSP is a formula in conjunctive normal form (CNF). Hence we translate a formula into a CSP clause-wise, each clause contributing one constraint (see Definition 3 above).

Aspect 1: Clauses as Constraints. For modelling a clause as a constraint, we distinguish four disjoint sets of variables: propositional variables and variables representing box formulas, and both subdivided according to polarity. We denote these sets P^+, P^-, B^+, B^-, respectively. This means a clause can be written as

$$\begin{array}{llll} \bigvee\{\, p : p \in P^+\} & \vee & \bigvee\{\, \neg p : p \in P^-\} & \vee \\ \bigvee\{\, x[\Box\phi] : x[\Box\phi] \in B^+\} & \vee & \bigvee\{\, \neg x[\Box\phi] : x[\Box\phi] \in B^-\} & \end{array}$$

This clause is viewed as a constraint on variables in the four sets:

$$clause_constraint(P^+, P^-, B^+, B^-).$$

It holds if *at least one* variable in the set $P^+ \cup B^+$ is assigned a 1 or one in $P^- \cup B^-$ is assigned a 0 — see Definition 1 above. We explain now this constraint in terms of the primitive constraint at_least_one, which is defined on a set of variables and parametrised by a constant, and which requires the latter to occur in the variable set. This, or a closely related constraint, is available in many constraint programming languages. The constraint library of ECL^{i}PS^e [27] contains a predefined constraint with the meaning of at_most_one, which can be employed to imitate at_least_one. We reformulate the *clause_constraint* as the disjunction

$$\mathsf{at_least_one}(P^+ \cup B^+, 1) \quad \vee \quad \mathsf{at_least_one}(P^- \cup B^-, 0).$$

Aspect 2: Disjunctions as Conjunctions. Propagating disjunctive constraints is generally difficult for constraint solvers. Therefore it is preferable to avoid them when modelling; and in our situation we can do so elegantly. The disjunction is transformed into a conjunction with the help of a single auxiliary link variable $\ell \in \{0, 1\}$. We obtain

$$\mathsf{at_least_one}(P^+ \cup B^+ \cup \{\ell\}, 1) \quad \wedge \quad \mathsf{at_least_one}(P^- \cup B^- \cup \{\ell\}, 0).$$

The link variable ℓ selects implicitly which of the two constraints must hold. For example, observe that $\ell = 0$ selects the constraint on the left. It forces $\mathsf{at_least_one}(P^+ \cup B^+, 1)$ and satisfies $\mathsf{at_least_one}(P^- \cup B^- \cup \{0\}, 0)$. It is useful to remark the following fact.

Fact 1 *A conjunctive constraint built from two conjuncts that share at most one variable is generalised arc-consistent if its two constituent constraints are.* •

In our case the two conjuncts share no variables except ℓ.

5.2 A More Advanced Modelling

In our advanced modelling we add on additional features to the base modelling; those features are meant to address a number of aspects of the base modelling.

Aspect 3: Partial Assignments by Constraints. While any solution of the CSP induced by a formula at some layer satisfies the formula at that layer, it is useful to obtain satisfying, partial Boolean assignments that mark as irrelevant as many box formulas in this layer as possible. This will cause fewer subformulas to enter the propositions generated for the subsequent layer. In our model, we employ the extra value u to mark irrelevance. In such a CSP, consider a clause constraint c on propositional variables $P = P^+ \cup P^-$ and variables representing box formulas $B = B^+ \cup B^-$. By definition, the variables in B are constrained only by c. Given this and Theorem 1, we can conclude the following:

Fact 2 *Suppose μ is a partial assignment that can be extended to a total assignment satisfying the CSP. Suppose μ is not on the variables $P \cup B$.*

- *The assignment $\mu \cup \{p \mapsto 1\} \cup \{x[\Box\phi] \mapsto \mathsf{u} : x[\Box\phi] \in B\}$, where $p \in P^+$, satisfies c and can be extended to a total assignment satisfying the CSP. An analogous result holds for $p \in P^-$.*
- *The assignment $\mu \cup \{x[\Box\psi] \mapsto 1\} \cup \{x[\Box\phi] \mapsto \mathsf{u} : x[\Box\phi] \in B - \{x[\Box\psi]\}\}$, where $x[\Box\psi] \in B^+$, satisfies c and can be extended to a total assignment satisfying the CSP. Again, an analogous result holds for $x[\Box\psi] \in B^-$.* •

In other words, if satisfying the propositional part of a clause suffices to satisfy the whole clause, then all box formulas in it can be marked irrelevant. Otherwise, all box formulas except one can be marked irrelevant.

Let us transfer this idea into a clause constraint model. First, we rewrite the base model so as to

- separate the groups of variables (in propositional and box variables),
- and convert the resulting disjunctions into conjunctions, again with the help of extra linking variables.

Next, we replace the at_least_one constraint for variables representing box formulas by an exactly_one constraint. This simple constraint is commonly available as well. ECL^{i}PS^e offers the more general occurrences constraint, which forces a certain number of variables in a set to be assigned to a specific value. We obtain

$$\begin{array}{llll} \mathsf{at_least_one}(P^+ \cup \{\ell_P^+\}, 1) & \wedge & \mathsf{exactly_one}(B^+ \cup \{\ell_B^+\}, 1) & \wedge \\ \mathsf{at_least_one}(P^- \cup \{\ell_P^-\}, 0) & \wedge & \mathsf{exactly_one}(B^- \cup \{\ell_B^-\}, 0). & \end{array}$$

The variable domains are: $P^+, P^- \in \{0, 1\}$, $B^+ \in \{1, \mathsf{u}\}$, $B^- \in \{0, \mathsf{u}\}$. The essential four linking variables are constrained as in the following formula, or the equivalent table.

$$\begin{array}{l} (\ \ell_P^+ = 1 \wedge \ell_P^- = 0\ ) \ \leftrightarrow\ (\ \ell_B^- = \mathsf{u}\ \vee\ \ell_B^+ = \mathsf{u}\ ) \\ \wedge \\ \ell_B^+ = 1\ \vee\ \ell_B^- = 0 \end{array}$$

ℓ_P^+	ℓ_P^-	ℓ_B^+	ℓ_B^-
1	0	1	u
1	0	u	0
0	1	1	0
0	0	1	0
1	1	1	0

Observe that the 5 tuples in the table correspond to the situations that we wish to permit — the clause is satisfied by either a positive or a negative box formula (but not both at the same time) or a positive or a negative propositional variable (maybe both at the same time).

ECL^{i}PS^e accepts the linking constraint in propositional form, and rewrites it internally into several arithmetic constraints. Alternative methods operate on the defining table. For our implementation we compiled it into a set of domain reduction rules, which can be executed efficiently [1]. This proved to be the fastest way of propagating this constraint among several methods we tested. We found that this linking constraint, among all constraints, is the one whose propagation is executed most often, hence propagating it efficiently is relevant.

Aspect 4: A Negated-CNF Constraint. Except for the initial input formula to KCSP which is in conjunctive normal form, the input to an intermediate call to the *sat* function of KCSP (see the algorithm in Figure 1) has the form $\Theta \wedge \neg\lambda$ where both Θ and λ are in CNF. A naive transformation of $\neg\lambda$ into CNF will result in an exponential increase in the size of the formula. We deal with this problem by treating $\neg\lambda$ as a constraint. Then the following holds.

Fact 3 *The constraint $\neg\lambda$ is satisfiable iff λ (which is a conjunction of clauses) has at least one unsatisfiable clause.* •

We formulate the constraint corresponding to $\neg\lambda$ consequently as a disjunction of constraints, each standing for a negated clause. The disjunction is converted into a conjunction with a set L of linking variables, one for each disjunct. The

$\ell_i \in L$ have the domain $\{0, 1\}$, where $\ell_i = 1$ means that the i-th disjunct holds — that is, the i-th clause in λ is unsatisfied. Instead of imposing $\mathsf{at_least_one}(L, 1)$ to select one disjunct, however, we require $\mathsf{exactly_one}(L, 1)$, in line with our goal of obtaining small partial Boolean assignments. In irrelevant disjuncts/clauses ($\ell_i = 0$) we force the box formulas to u. The definition for the constraint corresponding to a negated clause on sets of variables P^+, P^-, B^+, B^-, together with the linking variable ℓ, is

$$\begin{array}{lll} \ell = 1 & \leftrightarrow & (\ \forall b \in B^+.\ b = 0 \quad \wedge \quad \forall b \in B^-.\ b = 1\ ) \\ \wedge & & \\ \ell = 1 & \rightarrow & (\ \forall p \in P^+.\ p = 0 \quad \wedge \quad \forall p \in P^-.\ p = 1\ ) \\ \wedge & & \\ \ell = 0 & \leftrightarrow & (\ \forall b \in B^+.\ b = \mathsf{u} \quad \wedge \quad \forall b \in B^-.\ b = \mathsf{u}\ ). \end{array}$$

There is no need to constrain the propositional variables in a clause that is not selected by $\ell = 1$. Therefore, the propositional variables in P^+, P^- occur here less often than the variables in B^+, B^- which represent box formulas.

In `KCSP`, the propagation of this constraint is implemented as a user-defined constraint in the constraint library of ECL^{i}PS^e, and achieves generalised arc-consistency.

Aspect 5: A Constraint for Factoring. In our base model, we have treated and constrained each occurrence of a box formula as a distinct propositional variable. For instance, the two occurrences of $\Box p$ in the formula $\Box p \wedge \neg\Box p$ would be treated as two distinct propositional variables in our base model. We consider here the case that a box formula occurs several times, in several clauses, in any polarity. We then prevent assigning conflicting values to different occurrences.

Let us collect in $B_{\Box\phi}$ all variables $x_i[\Box\phi]$ representing the formula $\Box\phi$ in the entire CSP. We state as a constraint on these variables that

$$\forall x_1, x_2 \in B_{\Box\phi}. \quad \neg\,(\quad x_1 = 1 \wedge x_2 = 0 \quad \vee \quad x_1 = 0 \wedge x_2 = 1 \quad).$$

To see the effect, suppose there is a pair $x_1, x_2 \in B_{\Box\phi}$ with $x_1 \mapsto 0, x_2 \mapsto 1$ in a solution to the CSP without this factoring constraint. This means we obtain both $\Box\phi \mapsto 0$ and $\Box\phi \mapsto 1$ in the assignment returned, which in turn results in an unsatisfiable proposition being generated. The factoring constraint detects such failures earlier. Notice that the straightforward modelling idea, namely using one unique variable for representing a box formula in all clauses, clashes with the assumption made for the other partial-assignment constraints, i.e. that each box formula variable is unique.

6 Experimental Assessment

Theoretical studies often do not provide sufficient information about the effectiveness and behaviour of complex systems such as satisfiability solvers and

their optimisations. Empirical evaluations must then be used. In this section we provide an experimental comparison of our advanced modelling (Section 5.2) against the base model (Section 5.1), using a test developed by Heuerding and Schwendimann [14].

We will find that, no matter what other models and search strategies we commit to, we always get the best results by using constraints for partial assignments as in Subsection 5.2, Aspect 3. In the remainder of this paper, these are referred to as the *assignment-minimising constraints* or simply as the *minimising constraints*. As we will see below, these minimising constraints allows us to better direct the modal search procedure.

We conclude this section by comparing the version of KCSP that features the advanced modelling (Section 5.2) with KSAT. The constraint solver that we use as the *sat* function in KCSP is based on search with chronological backtracking and constraint propagation. The propagation algorithms are specialised for their respective constraints and enforce generalised arc-consistency on them, as discussed in Section 5 above.

6.1 Test Environment

State of the Art. In the area of propositional satisfiability checking there is a large and rapidly expanding body of experimental knowledge; see, e.g., [9]. In contrast, empirical aspects of modal satisfiability checking have only recently drawn the attention of researchers. We now have a number of test sets, some of which have been evaluated extensively [4,14,10,17,16]. In addition, we also have a clear set of guidelines for performing empirical testing in the setting of modal logic [14,16]. Currently, there are three main test methodologies for modal satisfiability solvers, one based on hand-crafted formulas, the other two based on randomly generating problems. To understand on what kinds of problems a particular prover does or does not do well, it helps to work with test formulas whose meaning can (to some extent) be understood. For this reason we opted to carry out our tests using the Heuerding and Schwendimann (HS) test set [14], which was used at the TANCS '98 comparison of systems for non-classical logics [26].

The HS Test Set. The HS test set consists of several classes of formulas for $\mathcal{K}$, and other modal logics we do not consider here; e.g., some problem classes for $\mathcal{K}$ are based on the pigeon-hole principle (*ph*) and a two-colouring problem on polygons (*poly*). Each class is generated from a parametrised logical formula. This formula is either a $\mathcal{K}$-theorem, that is provable, or only $\mathcal{K}$-satisfiable, that is non-provable; consequently the generated class only contains either *p*rovable formulas or *n*on-provable formulas, and is labelled accordingly. The table at the right lists all such classes for $\mathcal{K}$.

*p*rovable	*n*on-provable
branch_p	*branch_n*
d4_p	*d4_n*
dum_p	*dum_n*
grz_p	*grz_n*
lin_p	*lin_n*
path_p	*path_n*
ph_p	*ph_n*
poly_p	*poly_n*
t4_p	*t4_n*

Some of these parametrised formulas are made harder by hiding their structure or adding extra pieces. The parameters allow for the creation of modal formulas, in the same class, of differing difficulty. The idea behind the parameter is that the difficulty of proving formulas in the same class should be exponential in the parameter. This kind of increase in difficulty will make differences in the speed of the machines used to run the benchmarks relatively insignificant.

Benchmark Methodology. The benchmark methodology is to test formulas from each class, *starting with the easiest instance*, until the provability status of a formula can not be correctly determined within 100 CPU seconds. The result from this class will then be the parameter of the largest formula that can be solved within this time limit. The parameter ranges from 1 to 21.

6.2 Implementation

Let us turn to details of our implementation of the `KCSP` algorithm. We used the constraint logic programming system ECL^{i}PS^e [27]. The HS formulas are first negated, reduced in CNF and then translated into the format of `KCSP`.

We add the following heuristics to `KCSP` with minimising constraints (in an attempt) to reduce the depth of the `KCSP` search tree: the value `u` is preferred for box formulas, and among them for positively occurring ones. Furthermore, the instantiation ordering of box formulas is along the increasing number of nested boxes in them: e.g., $x[\Box p]$ is instantiated before $x[\Box\Box p]$.

6.3 Assessment

In this subsection we provide an assessment of the contributions made by the various "aspects" provided by our advanced modelling.

Aspect 3: Partial Assignments by Constraints. Do minimising constraints make a difference in practice? To address this question, here we focus on the so-called *branch* formulas in the HS test set. It is worth noticing the relevance of *branch_n* for automated modal theorem proving: the class of non-provable branch formulas, *branch_n*, is recognized as the hardest class of "truly modal formulas" for today's modal theorem provers, cf. [16]. These are the so-called Halpern and Moses branching formulas that "have an exponentially large counter-model but no disjunction [...] and systems that will try to store the entire model at once will find these formulae even more difficult" [16].

Figure 2 plots the run times of `KCSP` (with and without minimising constraints) on *branch* formulas. Clearly, minimising constraints do make difference. The superiority of `KCSP` with minimising constraints over `KCSP` with total assignments is particularly evident in the case of *branch* formulas. `KCSP` with minimising constraints manages to solve 13 instances of *branch_n* and all 21 of *branch_p* (in less than 2 seconds), and without only 2 instances are solved, for both flavours.

Why is it so? To understand the reasons for the superiority of KCSP with minimising constraints, let us first consider what happens with *branch_p*(3), which KCSP with total assignments is already unable to solve (see Figure 2). In KCSP with minimising constraints, there are two choices for box formulas at layer 0 (i.e., with the same modal depth as *branch_p*(3)), and none at the subsequent layers of modal formulas obtained by "peeling off" one box from *branch_p*(3) (see Subsection 4.1). This results in a modal search tree of exactly two branches. Instead, with total assignments there are 6 extra box formulas at layer 0, which implies an extra branching factor of $2^6 = 64$ at the root of the modal search tree only. All 6 box formulas will always be carried over to subsequent layers, positively or negatively.

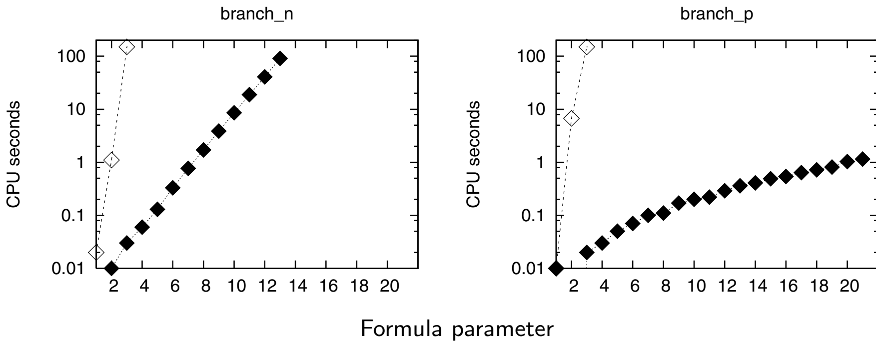


Fig. 2. KCSP with (♦) and without (◊) minimising constraints *branch* formulas. (Left): CPU time for $branch_n$ (seconds, log scale); (Right): $branch_p$ (seconds, log scale).

More in general, the superiority of KCSP with minimising constraints can be explained as follows: *the tree-like model that the solver (implicitly) attempts to construct while trying to satisfy a formula is kept as small as possible by the minimising constraints.* In this sense, constraints allows us to direct the modal search better than, for instance, unit propagation allows for in DPLL. We refer the reader to Subsection 6.4 below for more on this point.

Notice also that the results of KCSP with minimising constraints on the *branch* class are competitive with the best optimized modal theorem provers *SAT and DLP on the this class.

Aspect 4: Negated-CNF Constraint. In all the HS formula classes, having disjunctive constraints in place of CNF conversions increases the number of decided formulas, or, at least, does not decrease it. Avoiding CNF conversion by means of negated-CNF constraints may have a substantial effect, for example in the case of *ph_n*(4) — an instance of the pigeon-hole problem — which can now be solved in a few seconds. In contrast, by requiring CNF conversion (even with minimising constraints), ECL^{i}PS^e terminates the execution of KCSP pre-emptively for lack of memory. Unfortunately, the CNF conversion still necessary

at the top level remains, and even prevents entering KCSP for formulas $ph(k)$ with $k > 4$.

Aspect 5: Factoring Constraint. This constraint avoids simple contradictory occurrences of a formula in the subsequent layer. We remark that this consideration of multiple occurrences of a subformula does not always provide a strictly minimal number of box formulas with a Boolean value. Nevertheless, it proved beneficial for formulas with the same variables hidden and repeated inside boxes. In fact, it proved useful in all of the following cases: *grz*, *d4*, *dum_p*, *path_p*, *t4p_p*. In the remaining cases the contribution of factoring with constraints is insignificant, except for *path_n* where searching for candidate formulas to be factored slightly slows down search.

Formula Simplifications. As a preprocess to KCSP, the top-level input formula may be simplified to a logically equivalent formula. We use standard simplification rules for propositional formulas, at all layers, in a bottom-up fashion. Also, in the same manner, the following modal equivalences are used in simplifying a CNF formula: $\neg\Box\top \wedge \psi \leftrightarrow \bot \wedge \psi$ and its dual. Simplification in KCSP plays a relevant role in the case of *lin* formulas. E.g., consider *lin_n*(3): without simplifications and minimising constraints, KCSP takes longer than 5 minutes to return an answer; by adding simplifications and minimising constraints, KCSP takes less than 0.4 seconds; besides, by also adding factoring, KCSP solves the most difficult formula of *lin_n* in 0.06 seconds, that of *lin_p* in 0.01.

6.4 Results and a Comparison

In this part, we compare the performances of KSATC and KCSP on the HS test-set; see Table 1 below. Each column in the table lists a formula class and the number of the most difficult formula decided within 100 CPU seconds by each prover; we write > when all 21 formulas in the test set are solved within this time slot. We now explain the systems being compared in Table 1. The rows of the table are explained as follows.

First Row: KSATC**.** The results for KSATC (KSAT implemented in C++) are taken from [16]; there, KSATC was run with the HS test set on a 350 MHz Pentium II with 128 MB of main memory.

Second Row: KCSP**.** We used KCSP with all advanced aspects considered: i.e., partial assignments by constraints; negated-CNF constraints; factoring constraints; and formula simplifications. In the remainder, we refer to this as KCSP. The time taken by the translator from the HS format into that of KCSP is insignificant, the worst case among those in the comparison Table 1 taking less than 1 CPU second; these timings are included in the table entries for KCSP. We ran our experiments on a 1.2 GHz AMD Athlon Processor, with 512 MB RAM, under Red Hat Linux 8 and $\mathrm{ECL}^i\mathrm{PS}^e$ 5.5.

Third Row: KCSP/speed. To account partially for the different platforms for KSATC and KCSP, we scaled the measured times of KCSP by a factor 350/1200, the ratio of the processor speeds. The results are given in the bottom line KCSP/speed, and emphasized where different from KCSP.

Table 1. The top two rows give the most difficult formula of each HS class decided by KSATC and KCSP, respectively, in 100 CPU/s; > means that all formulas in that class are decided. In the bottom row, KCSP/speed, the figures for KCSP are obtained after scaling the measured times by the ratio of the processor speeds of KSATC and KCSP.

	branch		d4		dum		grz		lin		path		ph		poly		t4p	
	n	p	n	p	n	p	n	p	n	p	n	p	n	p	n	p	n	p
KSATC	8	8	5	8	>	11	>	17	3	>	8	4	5	5	12	13	18	10
KCSP	13	>	6	9	19	12	>	13	>	>	11	4	4	4	16	10	7	10
KCSP/speed	*11*	>	6	*8*	*17*	*11*	>	*10*	>	>	*9*	4	4	4	16	*9*	*6*	*8*

Result Analysis. Note, first of all, that KSATC is compiled C++ code while KCSP is interpreted ECL^{i}PS^{e} (i.e., PROLOG); this makes it very interesting to see that the performance of KCSP is often competitive with that of KSATC. There are some interesting similarities and differences in performance between KSATC and KCSP. For some classes, KCSP clearly outperforms KSATC, for some it is the other way around, and for yet others the differences do not seem to be significant.

For instance, KCSP is superior in the case of *lin* and *branch* formulas. In particular, as pointed out in Subsection 6.3 above, *branch_n* is the hardest "truly modal test class" for the current modal provers, and KCSP with partial assignments performs very well on this class. Now, KSATC features partial assignments, just like KCSP does, in that its underlying propositional solver is DPLL. So, why such differences in performance? The differences are due to our modelling and the reasons for these can be explained as follows in more general terms:

- extended domains and constraints allows for *more control over the partial assignments* to be returned by the adopted constraint solver than unit propagation allows for in DPLL;
- constraints allow us to represent, in a very compact manner, certain requirements such as that of reducing the number of box formulas to which a Boolean value is assigned.

Consequently, the models that KCSP (implicitly) tries to generate when attempting to satisfy a formula remain very small. In particular, in the case of *branch*, searching for partial assignments with minimising constraints yields other benefits *per se*: the smaller the number of box formulas to which a Boolean value is assigned at the current layer, the smaller the number of propositions in the subsequent layer; in this manner fewer choice points and therefore fewer search tree branches are created. Thereby adding constraints to limit the number of

box formulas to reason on, while still exploring the truly propositional search space, seems to be a winning idea on the *branch* class.

In the cases of *grz* and *t*4, instead, KSATC is superior to KCSP. Notice that KSATC features a number of optimisations for early modal pruning that are absent in KCSP, and these are likely to be responsible for the better behaviour of KSATC on these classes.

7 Finale

7.1 Looking Back

We described a constraint-based model for modal satisfiability. Thanks to this model, we could embed modal reasoning cleanly into pre-existing constraint programming systems, and directly make use of these to decide modal satisfiability. In this paper, we adopted the constraint logic programming system ECL^{i}PS^e.

In our base-model for modal satisfiability, we have extended domains for box formulas and appropriate constraints to reason about them; we also implemented (Section 5) and experimentally compared (Section 6) KCSP with further modelling constraints and heuristics for modal reasonings. In particular, KCSP with minimising constraints results to be competitive with the best modal theorem provers on the hardest "truly modal class" in the Heuerding and Schwendimann test set, namely *branch*; here, the addition of minimising constraints results in a significant reduction of the size and especially the branching of the "the tree-model" that our solver implicitly tries to construct for the input formula (Subsections 6.3 and 6.4).

More in general, an important advantage of our constraint-based modelling is that encoding optimisations (e.g., for factoring or partial assignments) can be done very elegantly and in an "economical" manner: that is, it is sufficient to add appropriate, compact constraints to obtain specific improvements (e.g., factoring or minimising constraints). Besides compactness in the models, extended domains and constraints allow for more control over the assignments to be returned by constraint solvers than unit propagation allows for in DPLL, as amply discussed in Subsection 6.4.

7.2 Looking Ahead

We conclude by elaborating on some open questions.

Modelling. Our current modelling of propositional formulas as finite CSPs can perhaps be enhanced so as to completely avoid CNF conversions. This could be achieved by (negated) clauses that may contain propositional formulas as well, not just variables or box formulas. The factoring constraints for controlling 0/1/u assignments to multiple occurrences of the same box formula are currently not optimally restrictive. Integrating them better with the assignment-minimising clause constraints can further reduce the modal spanning factor in certain situations. Also, we want to consider the literal modelling of propositions [28] and

compare it with the one presented in this paper on modal formulas; in the literal modelling of ϕ, for each clause of $Prop(\phi)$ there is a variable domain containing the literals of $Prop(\phi)$; binary constraints would then be imposed among those domains that share a $Prop(\phi)$ variable of the form p or $x[\Box\phi]$. One advantage of this modelling is that partial Boolean assignments for $Prop(\phi)$ would come for free, i.e., without the need of enlarging the domains with an `u` value; yet, the control of the modal search procedure may require novel constraints.

Constraint Algorithm. Simple chronological backtracking may not be the optimal choice for the problem at hand. Efficiency can be expected to increase by remembering previously failed sub-propositions (nogood recording, intelligent backtracking), and also successfully solved sub-problems (lemma caching).

Logic. Many-valued modal logics [8] allow for propositional variables to have further values than the Boolean 0 and 1. Our approach to modal logics via constraint satisfaction can be easily and naturally extended to deal with finitely-valued modal logics.

Acknowledgements

We thank Juan Heguiabehere for supplying us with a translator from HS formulas into `KCSP` format. Rosella Gennari was supported by the post-doc project grant *Automated Reasoning by Constraint Satisfaction* from the Autonomous Province of Trento. Maarten de Rijke was supported by grants from the Netherlands Organization for Scientific Research (NWO), under project numbers 612-13-001, 365-20-005, 612.069.006, 612.000.106, 220-80-001, and 612.000.207.

References

1. K.R. Apt and S. Brand. Schedulers for rule-based constraint programming. In *Proc. of ACM Symposium on Applied Computing (SAC'03)*, pages 14–21. ACM, 2003.
2. C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based Heuristics in Modal Theorem Proving. In *Proc. of the 14th European Conference on Artificial Intelligence 2000*, pages 199–203. IOS Press, 2000.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
4. 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. In *Proc. KR-92*, pages 270–281, 1992.
5. F. Bacchus and F. Kabanza. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 116, 2000.
6. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
7. L. Cardelli and A.D. Gordon. Anytime, Anywhere. Modal Logics for Mobile Ambients. In *Proc. of the 27th ACM Symposium on Principles of Programming Languages*, 2000.

8. M.C. Fitting. Many-valued modal logics II. *Fundamenta Informaticae*, XVII:55–74, 1992.
9. I. Gent, H. Van Maaren, and T. Walsh, editors. *SAT 2000.* IOS Press, 2000.
10. F. Giunchiglia and R. Sebastiani. Building Decision Procedures for Modal Logics from Propositional Decision Procedures. The Case Study of Modal $\mathbf{K}(m)$. *Information and Computation*, 162(1–2):158–178, 2000.
11. V. Haarslev and R. Möller. RACER. Accessed via `http://kogs-www.informatik.uni-hamburg.de/~race/`, September 2002.
12. J.Y. Halpern, R. Harper, N. Immerman, P.G. Kolaitis, M.Y. Vardi, and V. Vianu. On the unusual effectiveness of logic in computer science. *The Bulletin of Symbolic Logic*, 7:213–236, 2001.
13. B.P. Harrenstein, W. van der Hoek, J.-J.Ch. Meyer, and C. Witteveen. On modal logic interpretations of games. In *Proc. of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, 2002.
14. A. Heuerding and S. Schwendimann. A Benchmark Method for the Propositional Modal Logics **K**, **KT**, **S4**. Technical Report IAM-96-015, University of Bern, 1996.
15. I. Horrocks. FaCT. Accessed via `http://www.cs.man.ac.uk/~horrocks/FaCT/`, September 2002.
16. I. Horrocks, P.F. Patel-Schneider, and R. Sebastiani. An Analysis of Empirical Testing for Modal Decision Procedures. *Logic Journal of the IGPL*, 8(3):293–323, 2000.
17. U. Hustadt and R.A. Schmidt. On Evaluating Decision Procedures for Modal Logic. In *Proc. IJCAI-97*, pages 202–207, 1997.
18. M.R.A. Huth and M.D. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems.* Cambridge University Press, 1999.
19. M. Marx. XPath with conditional axis relations. In *Proc. of the International Conference on Extending Database Technology*, 2004.
20. MSPASS V 1.0.0t.1.2.a. Accessed via `http://www.cs.man.ac.uk/~schmidt/mspass`, 2001.
21. G. Pan, U. Sattler, and M.Y. Vardi. BDD-Based Decision Procedures for **K**. In *Proc. of CADE 2002*, pages 16–30. Springer LINK, 2002.
22. P.F. Patel-Schneider. DLP. Accessed via `http://www.bell-labs.com.user/pfps/dlp/`, September 2002.
23. M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-Deterministic Domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.
24. A.S. Rao and M.P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8:293–342, 1998.
25. A. Tacchella. *SAT System Description. In *Collected Papers from the International Description Logics Workshop 1999, CEUR*, 1999.
26. TANCS: Tableaux Non-Classical Systems Comparison. Accessed via `http://www.dis.uniroma1.it/~tancs`, 2000.
27. M.G. Wallace, S. Novello, and J. Schimpf. ECLiPSe: A platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, 1997.
28. T. Walsh. SAT v CSP. In R. Dechter, editor, *Proc. of the 6th International Conference on Principles and Practice of Constraint Programming*, pages 441–456, 2000.