CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e−mail: prdoc@itc.it − url: http://www.itc.it

Modelling Modal Satisfiability in Constraing Logic Programming

Brand S., Gennari R., Rijke M.

June 2003

Technical Report # P03−12−51

Modelling Modal Satisfiability in Constraint Logic Programming

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

[1] CWI, The Netherlands, {sebastian.brand,rosella.gennari}@cwi.nl
[2] Language and Inference Technology Group, ILLC, U. of Amsterdam, The Netherlands, mdr@science.uva.nl

Abstract. We present a novel encoding of modal satisfiability problems as Constraint Satisfaction Problems. We allow the domains of the resulting constraints to contain other values than just the Boolean 0 or 1, and add various constraints to reason about these values. This modelling is pivotal to speeding up the performance of our constraint-based procedure for modal satisfiability in Constraint Logic Programming (CLP). Our encoding results in a correct solver that attempts to minimize the size of the tree model that it is implicitly trying to generate. An important advantage of our modelling is that we do not need to change the underlying CSP algorithms, and can use them almost as "black boxes".

1 Introduction

In various branches of artificial intelligence, modal and modal-like formalisms such as temporal or description logics are widely used for reasoning about trees, graphs, and other relational structures [5]. Over the past decade, there have been various initiatives aimed at developing algorithms for solving the satisfiability problem for modal logic; this has resulted in a series of implementations. Some of these implement special purpose algorithms for modal logic, such as DLP [16], FaCT [11], RACER [9], *SAT [17], while others exploit existing tools or provers for either first-order logic (MSPASS [14]) or propositional logic (KSAT [8], KBDD [15]) through some encoding. In this paper we follow the second approach: we model and solve the modal satisfiability problem via Constraint Programming (CP).

More precisely, we build on the schema for KSAT, following the intuitions in [3]. First we encode modal input formulas into layers of finite constraint satisfaction problems (CSPs) with *more values than just the Boolean ones*; then we show that *any complete constraint solver* for finite CSPs can be used to solve them, and, hence, to determine modal satisfiability, in stead of the Davis-Logemann-Loveland procedure (usually denoted by DP, where P stands for Putnam).

The main novelty of our work is this: encoding modal satisfiability problems as CSPs with enlarged domains that can contain other values than just the Boolean 0 or 1, together with constraints to reason about these valuesOur approach has a number of benefits; for instance, we can set various strategies on the variables to split on in the constraint solver, simply by adding suitable

constraints. In particular, the additional values, together with the appropriate constraints, allow a constraint solver to return a partial Boolean assignment so that the number of modal reasoning steps does not explode, loosely speaking. All this is done without changing the underlying CSP algorithms: these are simply "black boxes" for us. While we are not aiming to be competitive with today's highly optimized modal provers, our experimental evaluations suggest that the approach is very promising in general, and even excellent in some cases.

In this paper, we focus on our modelling of modal satisfiability in Constraint Logic Programming (CLP); for further details concerning the modal satisfiability solver not included here, we refer the reader to [6]. The remainder of the paper is organized as follows. We start by laying the propositional groundwork in Section 2. We turn to modal matters in Section 3. In Section 4 we report on an experimental assessment and compare our implementation with a C version of `KSAT` on a benchmark test set used in the TANCS '98 comparison of provers for modal logic. We conclude in Section 4.

2 Propositional Formulas as Finite CSPs

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 $CNF(\psi)$.

Consider a set X of n variables, and assume that X is ordered by $<$; a *scheme* of X is a sequence $s := x_1, \ldots, x_m$ of variables in X, where $x_j < x_{j+1}$ for each $j = 0, \ldots, m$. 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 $P := \langle X, \boldsymbol{D}, \boldsymbol{C} \rangle$ is a *constraint satisfaction problem* (CSP). A CSP is *finite* if all the variable domains 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 is in $C(s)$; if d satisfies all the constraints of the CSP P, then P is a *consistent* or *satisfiable CSP*.

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 $x := 1, y := 0, z := 0$; and the binary constraint $C(x, y)$ for $x \vee \neg y$ to rule out the assignment $x := 0, y := 1$. In [19] the encoding in (1) is used to prove that a version of forward checking is superior to the basic `DP` procedure for deciding propositional satisfiability.

However, an algorithm such as forward checking will always return a total assignment given the encoding of formulas into CSPs above; whilst we aim at

a *partial* Boolean assignment. How do we get that without modifying the underlying CSP algorithm? 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 propositional variables in the considered formula; so we avoid formulas that only differ in the order of occurrence of their atoms, such as $y \vee x$ and $x \vee y$.

Definition 1. Consider a formula ψ and let X be the ordered set of distinct propositional variables that occur in ψ':

1. first, transform ψ in CNF; let ψ' be the resulting CNF formula;
2. create a domain $D_i := \{0, 1, 2\}$ 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 X variables in θ defined as follows; consider a tuple $d := (d_1, \ldots, d_m)$ in $\prod_{j=1}^{m} D_j$, and the function f given by $f(x_j) = d_j$, for $j = 1, \ldots, m$; then d is consistent with C_θ iff the restriction of f to $f^{-1}(\{0, 1\})$ satisfies θ.

Denote the resulting CSP by $CSP(\psi)$.

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 implementation details are discussed in Section 4 below.

Our new modelling of propositional formulas as in Definition 1 allows us to make a complete constraint solver return a partial Boolean assignment that satisfies a propositional formula ψ iff this is satisfiable. This result follows from Definition 1 and the following fact: a partial Boolean assignment μ satisfies ψ if, for each clause ϕ of $CNF(\psi)$, μ assigns 0 to at least one negative literal in ϕ, or 1 to at least one positive literal in ϕ.

Theorem 1 ([6]). *Consider a propositional formula ψ and let X be its ordered set of variables. Given a function f that assigns to each $x_i \in X$ a value in the domain D_i of $CSP(\psi)$, we use $\mu[f]$ to denote the restriction of f to the X subset $f^{-1}(\{0, 1\}) := \{x \in X : f(x) \in \{0, 1\}\}$. Then*

1. *f satisfies $CSP(\psi)$ iff $\mu[f]$ satisfies ψ;*
2. *a complete constraint solver for finite CSPs returns a function f such that $\mu[f]$ satisfies ψ iff ψ is satisfiable.* □

Note 1. It is sufficient that each $CSP(\psi)$ domain contains the Boolean 0 and 1 for the above result to hold. Thus, one could have values other than 2 (and 0 and 1) in the CSP modelling to mark some variables as "unnecessary" for deciding the satisfiability of a propositional formula; these values can also be used to devise several heuristics for modal reasoning, see Section 4.

3 Modal Formulas as Layers of CSPs

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

Modal Logic. We refer to [5] for extensive details on modal logic. To simplify matters, we will focus on the basic mono-modal logic **K**, even though our results can easily be generalized to a multi-modal version. Let P be a finite set of propositional variables. **K**-formulas are produced by the following rule $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Box\phi$, where $p \in P$. A *boxed formula* is a formula of the form $\Box\phi$.

A *modal model* is a triple $\mathcal{M} = (W, R, V)$ where W is a non-empty set (the model's domain), R is a binary relation on W, and $V : P \to 2^W$ is a valuation, assigning subsets of W to proposition letters. *Satisfaction* of a formula ϕ at a state w in a model $\mathcal{M}$ ($\mathcal{M}, w \models \phi$) is defined by induction on ϕ: $\mathcal{M}, w \models p$ if $w \in V(p)$; $\mathcal{M}, w \models \neg\phi$ iff $\mathcal{M}, w \not\models \phi$; $\mathcal{M}, w \models \phi \wedge \psi$ iff $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$; and $\mathcal{M}, w \models \Box\phi$ iff for all v such that Rwv, $\mathcal{M}, v \models \phi$. A formula ϕ is *satisfiable* if for some model $\mathcal{M}$ and state w in $\mathcal{M}$ we have that $\mathcal{M}, w \models \phi$. **K**-*satisfiability* is the following problem: given a mono-modal formula ϕ, is ϕ satisfiable?

Given a formula ψ whose set of proposition letters is P, the *variables at layer* 0, or *layer*-0 *variables* of ψ are the subformulas of ψ defined as follows: ψ is a layer-0 variable if ψ is a boxed formula or a propositional variable $p \in P$; π is a layer-0 variable if $\neg\pi$ is so; π and χ are layer-0 variables if $\pi \vee \chi$ or $\pi \wedge \chi$ are so; nothing else is a layer-0 variable. A formula ϕ is a *proposition at layer* 0, or a *layer*-0 *proposition* (of ψ) iff it is a layer-0 variable of ψ or its negation, the conjunction or disjunction of layer-0 propositions of ψ; denote by $Lay(\psi, 0)$ the set of such formulas. In general, a *variable at layer* $(i+1)$, or a *layer*-$(i+1)$ *variable* θ (of ψ) is a subformula of ψ of the form θ' where $\Box\theta'$ is a layer-i proposition. A *proposition at layer* $(i+1)$, or a *layer*-$(i+1)$ *proposition* (of ψ) is a layer-$(i+1)$ variable of ψ or its negation, a conjunction or disjunction of layer-$(i+1)$ propositions of ψ; denote by $Lay(\psi, i+1)$ the set of such formulas. For instance, given $\psi = \Box(p \vee \Box q)$, the only layer-0 variable is ψ itself; p and $\Box q$ are layer-1 variables, and $p \vee \Box q$ is a layer-1 proposition; q is the only layer-2 variable. Intuitively, a layer-n variable of ψ occurs in the scope of n modal operators in ψ; a layer-n proposition is a combination of layer-n variables or their negation via the propositional connectives $\wedge$ or $\vee$. Subformulas of ψ are thus stratified into layers of propositions. If n is the maximum for which $Lay(\psi, n)$ is defined, then n is the *modal depth* of ψ; this is denoted by $\mathrm{md}(\psi)$. Intuitively, $\mathrm{md}(\psi)$ gives us the "deepest layer" of propositions that we need to satisfy.

The General *k_sat* Schema. In Table 1, we present the general algorithm schema *k_sat*, on which `KSAT` [8] is based, for deciding the satisfiability of formulas in **K**. The *sat* procedure (called at line 5) determines the satisfiability of ψ as a layer-0 proposition by returning a propositional assignment μ; if μ is empty, backtracking takes place. Thereby, the modal search space gets stratified into layers and thus explored, layer by layer, in a depth-first manner. To achieve this, we use stacks in *k_sat* (in [8] recursive procedures are used instead). For each layer-i variable of the form $\Box\phi$ to which the assignment μ, returned by *sat*, assigns 0, precisely one layer-$(i+1)$ proposition is pushed on top of the stack.

Table 1. The *k_sat* algorithm schema.

```
μ := ∅;
Propositions := stack_init([ψ]);
while not stack_empty(Propositions) do
    ψ := stack_pop(Propositions);
    sat(ψ, μ);        % return μ ≠ ∅ else backtrack
    Θ := ⋀{θ : □θ = 1 in μ};
    for each □ν = 0 in μ do
        Propositions := stack_push(¬ν ∧ Θ, Propositions);
```

Each iteration of a while-loop pops a proposition from the stack and tries to prove it, which may add sub-propositions in turn.

Consider a Boolean assignment μ for a set X of layer-i variables in ψ; then the formula

$$\bigwedge\{\psi_n \,:\, \psi_n \in X \text{ and } \mu(\psi_n) = 1\} \wedge \bigwedge\{\neg\psi_j \,:\, \psi_j \in X \text{ and } \mu(\psi_j) = 0\} \qquad (2)$$

is called the *proposition generated by* μ. The following property will be used to show that *k_sat* is a decision procedure for **K**-satisfiability.

Property 1. The input formula ψ to *sat* is **K** satisfiable iff *sat* returns a non-empty Boolean assignment μ such that μ satisfies ψ and $P(\mu)$ is **K**-satisfiable.

Theorem 2 ([6]). *If Property 1 holds, then k_sat is a decision procedure for* **K***-satisfiability.* □

The `KCSP` Algorithm. We now devise a modal decision procedure based on the *k_sat* schema, but with CSP algorithms as the underlying propositional solver. We start by explaining the underlying intuitions by considering an example; then we define and prove how any complete solver for finite CSPs can be correctly used as the *sat* procedure in *k_sat* to do modal reasoning.

Example. Consider the following modal formula ψ:

$$\psi := (\Box p \vee \Box r) \wedge \neg\Box(p \vee q) \wedge p.$$

Viewing ψ as a layer-0 proposition, the following $CSP(\psi)$ is obtained:

(a) four layer-0 variables: $\Box p$; $\Box r$; $\Box(p \vee q)$; p;
(b) all the respective domains are set equal to $\{0, 1, 2\}$;
(c) two unary constraints to assign 1 to p and 0 to $\Box(p \vee q)$; a binary constraint to forbid that $\Box p$ and $\Box r$ get both instantiated to 0.

Assigning 2 to a variable means not committing to any decision concerning its Boolean values, 0 and 1. The CSP is given to the propositional CSP solver that returns one out of the two possible assignments: μ that maps $\Box p$ to 1, $\Box r$ to 2,

$\Box(p \vee q)$ to 0 and p to 1; μ' that only differs from μ in that it assigns 2 to $\Box p$ and 1 to $\Box r$. Suppose that μ is the returned assignment; then, for all the boxed formulas to which μ assigns 1, the formulas within the scope of $\Box$ are joined in a conjunction Φ:

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

This is the *universal theory*: in model-theoretic terms, this is the formula that is to be satisfied by each layer-1 modal successor of a state satisfying the layer-0 variable $\Box p$. Then the algorithm considers all the boxed formulas to which μ assigns 0, in this case only $\Box(p \vee q)$; thus $p \vee q$ gets negated, translated in CNF if needed, and the result is the formula

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

There may of course be multiple such existential theories Θ, which have to be satisfied at layer-1, though not necessarily at the same state. The conjunction $\Phi \wedge \Theta$ is passed to the propositional CSP solver; in this case, the clause that is passed on is $p \wedge \neg p \wedge \neg q$. This is translated into a new CSP and its inconsistency determined; we backtrack to μ' and the new (UT) r is created; the satisfiability of $r \wedge \neg p \wedge \neg q$ is determined, and that of ψ, by returning `true`.

Notice the key points about (UT) and (ET): we *only* consider the boxed formulas to which the assignment assigns a Boolean value, 0 or 1. The boxed formulas to which 2 is assigned are disregarded, safely so because of Theorem 1. As we will see below, the availability of values other than 0 and 1 has a number of advantages. E.g., a sensible heuristic could be to postpone the modal reasoning part, that is, the creation of (UT) and (ET), "as much as possible." This can be obtained by instantiating the propositional variable domains with $\{0, 1\}$ and the others with $\{0, 1, 2\}$; then, when a value has to be chosen for a boxed formula in the *sat* procedure, the non-Boolean value 2 should be always tried first. In this manner, fewer formulas occur in (UT) and (ET), and this may reduce the modal search space. This and related heuristics are discussed in Section 4.

As the above example illustrates, when ψ is encoded as a CSP, it is viewed as a layer-0 proposition. Formally, we have the following definition.

Definition 2. Let ϕ be a modal formula and X the set of all layer-0 variables in ϕ. Consider ϕ as a layer-0 proposition with variables in X; then the *CSP of the modal formula* ϕ is the CSP of the layer-0 proposition ϕ.

We use $CSP(\phi)$ to denote the CSP of the modal formula ϕ. Given ϕ and X as in Definition 2, and a total assignment f for $CSP(\phi)$, let $\mu[f]$ be the restriction of f to $f^{-1}\{0, 1\}$ and $P(\mu[f])$ the proposition generated by $\mu[f]$, see (2).

Definition 3. The `KCSP` *algorithm* is defined as follows. In the *k_sat* schema we instantiate *sat* with a complete constraint solver for finite CSPs and we transform ψ as $CSP(\psi)$ before passing it on to the constraint solver, where $CSP(\psi)$ is as in Definition 1.

It is not difficult to prove that a constraint solver for finite CSPs enjoys Property 1, thanks to the second item of Theorem 1, see [6]. Thus Theorem 2 yields the following result concerning KCSP as in Definition 3.

Theorem 3 ([6]). KCSP *is a decision procedure for* **K***-satisfiability.* □

4 Experimental Assessment and Optimisations of KCSP: When the Modelling Pays Off

In this section we introduce our implementation of KCSP; the complete solver adopted as *sat* in the implementation of KCSP is a *backtracking-based algorithm with hyper-arc consistency*. When a value is assigned to a variable x_i, the algorithm performs hyper-arc consistency by inspecting constraints involving the unassigned variables and x_i. When an inconsistency is detected, backtracking to the last assignment for x_i occurs; another value is assigned and, if all the available values are inconsistent with a constraint on x_i, backtracking to one of the previously instantiated variables takes place. Nowadays, there are a number of variants of this basic backtracking-based algorithm. Walsh [19] provides a theoretical analysis of CSP-based approaches to SAT formulas, and shows that a version of forward checking for non-binary CSPs, called nFC1, outperforms the basic DP procedure on the encoding given in Example 1.

We also present several optimisations for KCSP; our implementation of KCSP incorporates some switches that allow us to turn the optimisations on or off. Often, theoretical studies do not provide an indication of the effectiveness and behavior of complex systems such as satisfiability solvers and their optimisations. Instead, empirical evaluations have to be used. Therefore, in this section, we also provide an experimental assessment of the various optimisations for KCSP, using a test developed by Heuerding and Schwendimann. No matter what other optimisations we adopt, we will see that we get the best results by using partial assignments. We conclude this section by comparing the version of KCSP that features all our optimisations with KSATC, an implementation of KSAT in C++.

Test Environment. In the area of propositional satisfiability checking there is large and rapidly expanding body of experimental knowledge; see, e.g., [7]. 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, 10, 8, 13, 12]. In addition, we also have a clear set of guidelines for performing empirical testing in the setting of modal logic [10, 12].

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 [10], which was used at the TANCS '98 comparison of systems for non-classical logics [18].

The HS test set consists of several classes of formulas for **K**, and for other modal logics that we are not concerned with in this paper. Each class has been generated from a (relatively) simple parameterized logical formula; this parameterized formula is either a **K**-theorem (hence the generated class is labelled with p) or only **K**-satisfiable (hence the generated class is labelled with n) in case of **K**. Some of these parameterized 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.

The classes of test formulas for **K** are: *branch_p* and *branch_n*; *d4_p* and *d4_n*; *dum_p* and *dum_n*; *grz_p* and *grz_n*; *lin_p* and *lin_n*; *path_p* and *path_n*; *ph_p* and *ph_n*; *poly_p* and *poly_n*; *t4_p* and *t4_n*. 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 seconds. The result from this class will then be the parameter of the largest formula that can be solved within the time limit. The parameter ranges only from 1 to 21 and the modal theorem provers are given a time out of 100 CPU seconds for each of the 21 formulas.

Implementation Issues. Let us turn to details of our implementation of the `KCSP` algorithm. We used the constraint logic programming system ECL^{i}PS^e. A translator from the HS formula format into the format of `KCSP` was provided by Juan Heguiabehere. We ran all our experiments on an AMD Athlon Processor (1 GHz), with 512MB RAM, under Red Hat Linux 7.1. The HS formulas that we used in the experiments as well as the ECL^{i}PS^e code for `KCSP` are available at the web page `http://www.cwi.nl/~sbrand/Research/KCSP/`.

Modelling, Optimisations and Analysis. In our CSP-modelling of a clause in a formula, we distinguish the four (disjoint) sets of literals: propositional literals $LitP$, and literals $LitB$ representing boxed formulas, and both subdivided by polarity:

$$Clause \quad = \quad LitP^+ \vee LitP^- \vee LitB^+ \vee LitB^-.$$

A clause is then viewed as a constraint on the corresponding four sets of variables:

$$clause\_constraint(P^+, P^-, B^+, B^-).$$

This 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 also Definition 1. We use the constraint at_least_one, which is defined on a set of variables and parameterised by a constant, and which requires the latter to occur in the set. We can formulate

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

The constraint library of ECL^{i}PS^e contains a predefined constraint with the meaning of at_most_one. It can easily be employed to imitate at_least_one, using the number of involved variables. Finally, the disjunction

$$\mathsf{at\_least\_one}(X^+, 1) \vee \mathsf{at\_least\_one}(X^-, 0)$$

is elegantly transformed into a conjunction with the help of a single link variable l. We find $\mathsf{at\_least\_one}(X^+ \cup \{l\}, 1) \wedge \mathsf{at\_least\_one}(X^- \cup \{l\}, 0)$. Observe for example that $l = 0$ forces $\mathsf{at\_least\_one}(X^+, 1)$, in which case it is irrelevant whether $\mathsf{at\_least\_one}(X^-, 0)$ holds — and indeed $\mathsf{at\_least\_one}(X^- \cup \{l\}, 0)$ is satisfied.

Optimisation 1*: constraints for partial assignments.* To get partial Boolean assignments in KCSP so that the reasoning on the boxed formulas is "delayed" (and possibly never done), we let the variable domains of the input CSP be as follows: the propositional variables have as domains $\{0, 1\}$, whereas the boxed formulas have as domains $\{0, 1, 2\}$. Notice that each boxed formula is translated into a distinct propositional variable for the constraint solver. It occurs not only once per clause but once per formula.

In a similar way, we impose constraints to obtain a partial assignment in KCSP with a small number of boxed formulas "switched on": i.e., *at most one* boxed formula per clause may be assigned a Boolean value. These are referred to as the (*assignment-*)*minimising* constraints. To use conjunction instead of disjunction in these constraints we use several linking variables instead of a single one. More precisely, a minimising constraint is of the form

$$\begin{array}{lll} \mathsf{at\_least\_one}(P^+ \cup \{l_P^+\}, 1) & \wedge & \mathsf{exactly\_one}(B^+ \cup \{l_B^+\}, 1) \quad \wedge \\ \mathsf{at\_least\_one}(P^- \cup \{l_P^-\}, 0) & \wedge & \mathsf{exactly\_one}(B^- \cup \{l_B^-\}, 0), \end{array}$$

where we use a constraint exactly_one. It can be modelled by the occurrences constraint of ECL^{i}PS^e, which forces a number of variables in a set to be assigned to a specific value. The important four linking variables are constrained by

$$(l_P^+ = 1 \wedge l_P^- = 0 \;\leftrightarrow\; (\, l_B^- = 2 \;\vee\; l_B^+ = 2\,)) \quad \wedge \quad (\, l_B^+ = 1 \;\vee\; l_B^- = 0\,)$$

or, equivalently, as in this table:

l_P^+	l_P^-	l_B^+	l_B^-
1	0	1	2
1	0	2	0
0	1	1	0
0	0	1	0
1	1	1	0

We found that, among all constraints, this is the one whose propagation is executed most often. Therefore we experimented with several implementations. ECL^{i}PS^e accepts its logical form, and reifies it internally into several arithmetic constraints. This turned out to be slower than pre-compiling the defining table into domain reduction rules that can be scheduled efficiently [2, 1].

We add the following heuristics to KCSP with minimising constraints to (attempt to) reduce the depth of the KCSP search tree: the value 2 is preferred for boxed formulas, and among those, for boxed formulas occurring negatively.

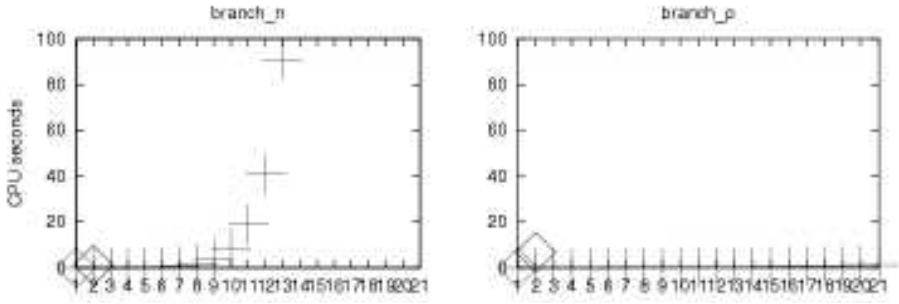

Fig. 1. The *branch* formulas: KCSP with minimising constraints for partial assignments (+) vs. KCSP with total assignments ($\diamond$).

Assessment. The superiority of KCSP with minimising constraints over KCSP with total assignments is particularly evident in the case of *branch* formulas; see Figure 1: KCSP with minimising constraints manages to solve 13 instances of *branch_n* and all 21 of *branch_n* (in less than 2 seconds!), and without only 2 instances are solved, for both flavors. It is worth pointing out that *branch_n* (non-provable) is recognized as the hardest class of "truly modal formulas" for today's modal theorem provers[3], and that our results of KCSP with minimising constraints are competitive with the best optimized modal theorem provers *SAT and DLP on this class. What seems to be going on is that the tree-like model that the solver (implicitly) attempts to construct while trying to satisfy a formula is kept as small as possible by KCSP with minimising constraints. To understand why KCSP with minimising constraints seems to be optimal here, observe, first of all, that constraints allow us to represent in a very compact manner the requirement of reducing the number of boxed formulas to which a Boolean value is assigned. Secondly, searching for this kind of partial assignments yields other benefits *per se*: fewer boxed formulas to which a Boolean value is assigned at the current layer means fewer propositions in the subsequent layer; in this manner fewer choice points and therefore fewer search tree branches are created. Consider, for instance, what happens with *branch_p*(3). In KCSP with minimising constraints, there are two choices for boxed formulas at layer 0, and none at the

[3] 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"; see [12].

subsequent layers, resulting in a modal search tree of exactly two branches. With total assignments, there are 6 extra boxed 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 boxed formulas will always be carried over to the next layers, positively or negatively.

There is lots of room for additional semantically motivated refinements in the encoding of formulas as CSPs and the use of these refinements for devising various searching strategies in KCSP with minimising constraints. We have started experimenting with several heuristics based on the modal depth of the formula, for instance:

- assign a Boolean value first to the boxed formulas with minimal modal depth;
- assign a Boolean value first to the boxed formulas with maximal modal depth.

We compared the implementations of KCSP with and without both heuristics. The results obtained are not astonishing, even if, in general, choosing a boxed formula with minimal modal depth usually gives the best results; in fact, this tends to reduce the depth of the modal model that KCSP is implicitly attempting to construct. However, these results clearly show that more refined learning heuristics will be needed to split on modal formulas and take care of the modal depth information.

Optimisation 2: disjunctive constraints. In the basic KCSP algorithm, every time 0 is assigned to a formula $\neg\Box\psi$, the subformula $\neg\psi$ is first transformed in CNF before turning it into CSP form. This is not an efficient choice. In fact, it is well-known that CNF conversion can lead to an explosion in the size of the formula. This can be remedied by treating $\neg\psi$ as a special disjunctive constraint

$$\neg\psi = \bigvee_{i=1}^{n} \phi_i$$

by means of link variables l_i, one for each ϕ_i in ψ. This constraint is satisfied iff 1 is assigned to *at least* one of the l_i. For coherence with our minimising constraints, 1 is assigned to *exactly one* of the l_i, again by an exactly_one constraint. Setting a link variable to 1 forces all positively occurring variables in ϕ_i to 0, and all negative ones to 1. All other link variables get set to 2; this means that we set to 2 the variables representing boxed formulas, while any truth value is allowed for the propositional variables. Our implementation of this behaviour is a user-defined constraint in ECL^{i}PS^e, and not too difficult.

In all the HS formula classes, having disjunctive constraints in place of CNF conversions increases the number of returned formulas, or, at least, does not decrease it. Avoiding CNF conversion by means of disjunctive 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 allowing CNF conversion in KCSP (also with minimising constraints), ECL^{i}PS^e stopped before finishing for lack of memory after having allocating 100 MB.

Optimisation 3*: constraints for factoring.* Consider a subformula $\Box\psi$ of ϕ, the input to KCSP; suppose that $\Box\psi$ occurs several times in ϕ, positively and negatively. Then, in the KCSP algorithm, each occurrence at position i is encoded as a different variable in the corresponding CSP, say x_i. To obviate this, we add a new constraint $C_{\Box\psi}$ on the CSP variables for $\Box\psi$:

$$C_{\Box\psi} \text{ states that no two variables } x_i, x_k \text{ exist with } x_i = 0 \text{ and } x_k = 1 \ .$$

This constraint thus avoids contradictory occurrences of ψ in the subsequent layer. Notice that even this implementation of factoring may not always provide a *strictly* minimal number of boxed formulas with a Boolean value. Nevertheless, this form of factoring in KCSP turned out to be beneficial for formulas with the same variables hidden and repeated inside boxes. In fact this factoring proved useful in all of the following cases: *grz*, *d*4, *dum_p*, *path_p*, *t*4*p_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 slows down search slightly.

Optimisation 4*: simplifications.* Finally, we can turn on a number of simplifications in the implementation of KCSP. These take place only once, upon reading the input formula; after that, it is never simplified again as such. 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 adding also factoring, KCSP solves the most difficult formula of *lin_n* in 0.06 seconds, that of *lin_p* in 0.01.

Results and a Comparison. Table 2 displays a comparison of KSATC with KCSP in which all the optimisations above are switched on; from now on 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 Table 2 taking less than 1 CPU second; these timings are taken into account for KCSP in Table 2. The results for KSATC are taken from [12]; there, KSATC was run on the Heuerding and Schwendimann test set, on a 350MHz PentiumII with 128MB of main memory. In the table, we write $>$ when all 21 formulas in the test set are solved within 100 CPU seconds, else we write the number of the most difficult formula decided within the time available.

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 definitely superior in the case of *lin* and *branch* formulas; as explained in the analysis of the optimisation with constraints for partial assignments, *branch_n* is the hardest "truly modal test class" for the

Table 2. KCSP vs. KSATC.

	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	>	7	9	19	12	10	13	>	>	11	4	4	4	15	10	7	10

current modal theorem provers; thus adding constraints to limit the number of boxed formulas to reason on, while still exploring the truly propositional search space, seems to be a winning idea in this case. We should point out that KSATC features partial assignments too. Thus the fact that KCSP shows such a good behavior on *branch* formulas has to be attributed to the fact that the models that it (implicitly) tries to generate when attempting to satisfy a formula, remain very small. 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 we have not added to KCSP (yet), and these are likely to be responsible for the better behavior of KSATC on these classes. See the discussion in the final section for more on this point.

5 Finale

In this paper we have described a modelling and a solving procedure for modal satisfiability problems using off-the-shelf CSP algorithms, but mainly focussed on the modelling aspects of our work. As discussed at length in [6], the solving procedure works by decomposing modal satisfiability problems into "layers" of propositional satisfiability problems, each of which is encoded not as a Boolean CSP but as a finite CSP with values other than just the Boolean 0 and 1.

Our novel modelling of formulas as CSPs allows us to do modal reasoning by working on top of existing constraint algorithms for finite CSPs, without modifying the algorithms to obtain partial Boolean assignments. By using the additional non-Boolean values and appropriate constraints, we implemented KCSP with various optimisations and some heuristics for modal reasoning in the CLP system ECL^{i}PS^e.

As shown in Section 4, KCSP with our modelling and the various optimisation constraints is competitive with the best modal theorem provers on the hardest "truly modal class" in the Heuerding and Schwendimann test set, namely *branch*. The addition of CLP constraints for minimising the number of boxed formulas to be taken into account, seems to result in a reduction of the size (and especially the width) of the "the tree-model" that the solver implicitly tries to construct for the input formula. An important advantage of our CLP 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 constraints to obtain specific optimisations, and constraints provide a compact representation means in CLP.

We conclude by elaborating on some open questions.

– *Modelling*: the current encoding of propositional formulas into CSPs can be enhanced so as to completely avoid CNF conversions. That is, each modal formula should be directly encoded and solved, layer by layer, as a CSP with further values than 0 and 1. We are already avoiding CNF conversion in the restricted case of negative boxed formulas, by adding a disjunctive constraint; this results in a significant improvement of `KCSP`'s behavior, see Subsection 4. Also, the minimising constraints are now only loosely integrated with the factoring constraints. A better model is likely to further reduce the modal spanning factor in certain situations.
– *Algorithm*: we plan to enforce and experiment with stronger forms of constraint propagation, e.g., strong 3-consistency. Also, we want to add learning heuristics in `KCSP` to split on variables and some form of early modal pruning (such as incrementally adding formulas to the "universal theory" denoted by Θ in *k_sat* and every time enforcing hyper-arc consistency on Θ, before constructing a full partial assignment); this is likely to improve the behavior of `KCSP` on formulas such as the *grz* and *t*4 ones, that are characterized by repeated occurrences of the same sub-formulas at different layers.
– *Implementation*: we will work on the implementation of CSPs in CLP to obtain a more compact representation of CSPs. This will help us to solve our current problem with the *ph* formulas (each of these encodes the pigeon-hole problem, which is notoriously hard for SAT provers): right now ECL^iPS^e simply gets stuck when loading the 5th formula in the *ph* test sets.

Acknowledgements

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. Rosella Gennari was supported by the ERCIM fellowship 2002-27.

References

1. K.R. Apt and S. Brand. Schedulers for rule-based constraint programming. In *ACM Symposium on Applied Computing (SAC)*, 2003.
2. K.R. Apt and E. Monfroy. Constraint programming viewed as rule-based programming. *Theory and Practice of Logic Programming*, 2001.
3. 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.
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*, 1992.
5. P. Blackburn, M. De Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
6. S. Brand, R. Gennari, and M. de Rijke. Constraint Programming for Modelling and Solving Modal Satisfiability. Submitted 2003.

7. I. Gent, H. Van Maaren, and T. Walsh, editors. *SAT 2000*. IOS Press, 2000.
8. 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.
9. V. Haarslev and R. Möller. RACER. Accessed via `http://kogs-www.informatik.uni-hamburg.de/~race/`, September 2002.
10. 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.
11. I. Horrocks. FaCT. Accessed via `http://www.cs.man.ac.uk/ ~horrocks/FaCT/`, September 2002.
12. 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.
13. U. Hustadt and R. A. Schmidt. On Evaluating Decision Procedures for Modal Logic. In *Proc. IJCAI-97*, pages 202–207, 1997.
14. MSPASS V 1.0.0t.1.2.a. URL: `http://www.cs.man.ac.uk/~schmidt/mspass`. Accessed February 23, 2001.
15. G. Pan, U. Sattler, and M. Y. Vardi. BDD-Based Decision Procedures for **K**. In *Proceedings of CADE 2002*, pages 16–30. Springer LINK, 2002.
16. P.F. Patel-Schneider. DLP. Accessed via `http://www.bell-labs.com.user/pfps/dlp/`, September 2002.
17. A. Tacchella. *SAT System Description. In *Collected Papers from the International Description Logics Workshop 1999, CEUR*, 1999.
18. TANCS: Tableaux Non-Classical Systems Comparison. `http://www.dis.uniroma1.it/~tancs`. Accessed January 17, 2000.
19. T. Walsh. SAT v CSP. In R. Dechter, editor, *Proc. of the 6th International Conference on Principles and Practice of Constraint Programming*, volume 1894, pages 441–456, 2000.