Constraint Programming for Modelling and Solving 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@irst.itc.it
[3] Language and Inference Technology Group, ILLC, University of Amsterdam, The Netherlands, mdr@science.uva.nl

Abstract. We explore to what extent and how efficiently constraint programming can be used in the context of automated reasoning for modal logics. We encode modal satisfiability problems as constraint satisfaction problems with non-boolean domains, together with suitable constraints. Experiments show that the approach is very promising.

1 Introduction

In various branches of artificial intelligence, modal and modal-like formalisms are used for reasoning about relational structures [3], such as transition systems. Recently, there have been increased efforts to develop algorithms for solving the satisfiability problem for modal logic. Some implementations use special purpose algorithms for modal logic, such as DLP [11], FaCT [7], RACER [5], *SAT [12], while others exploit existing tools or provers for either first-order (MSPASS [9]) or propositional logic (KSAT [4], KBDD [10]) through some encoding.

We follow the latter approach: we model and solve the modal satisfiability problem via Constraint Programming (CP). We build on the fact that a modal formula is satisfiable in the basic logic $\boldsymbol{K}$ only if it is so on a tree-like model [2, 1]. This property allows us to stratify $\boldsymbol{K}$-satisfiability problems into "layers" of propositional satisfiability problems. In [1] this layering was encoded into a translation from modal into first-order logic. We build on the schema for KSAT [4], following the intuitions in [1]. We encode modal input formula into layers of finite constraint satisfaction problems (CSPs) with additional non-Boolean values; we show that *any complete constraint solver* for finite CSPs can be used to solve them (and, hence, to determine modal satisfiability).

Our aim in this paper is to explore to what extent and how efficiently CP can be used in the context of automated reasoning for modal logics. To the best of our knowledge, our work constitutes the first attempt in this direction. The novelty of the paper is two-fold: first, *encoding* modal satisfiability problems as CSPs with enlarged domains; and second, *solving* such CSPs by means of suitable propagation algorithms in a CP environment.

We turn to modal logic matters in Section 2. In Section 3 we report on an experimental assessment and comparison. We conclude in Section 4.

2 Modal Logic and CSPs

Modal Logic. We focus on the basic mono-modal logic $\boldsymbol{K}$, even though our results can easily be generalized to a multi-modal version. Let P be a finite set of propositional variables. $\boldsymbol{K}$-formulas are produced by the rule $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Box\phi$, where $p \in P$. A formula is *boxed* if it is 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 $\mathcal{M}, w \models \phi$. $\boldsymbol{K}$*-satisfiability* is the following problem: given a mono-modal formula ϕ, is ϕ satisfiable?

A *tree model* is a model $\mathcal{M} = (W, R, V)$ such that (W, R) is a tree. $\boldsymbol{K}$-formulas satisfy the *tree model property*: they are satisfiable only if they are satisfiable at the root of a tree model; see [3, Chapter 2] for details.

Let ψ be a modal formula on P. A ψ subformula of the form $p \in P$ or $\Box\psi'$ is a *layer*-0 *variable* (of ψ). A formula ϕ is 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. In general, 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 *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.

The k_sat Schema. The algorithm schema *k_sat* on the right-hand side, on which `KSAT` [4] is based, determines the satisfiability of formulas in $\boldsymbol{K}$: the *sat* procedure determines the satisfiability of ψ as a proposition by returning a propositional assignment, if no exists backtracking takes place. Thus, the modal search space is explored layer by layer, in a depth-first manner.

```
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);
```

The `KCSP` *Algorithm.* Our next aim is to devise a modal decision procedure based on the *k_sat* schema, with CSP algorithms as the underlying propositional solver.

Definition. 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. Let us denote the CSP of the modal formula ϕ with $CSP(\phi)$.

We instantiate *sat* with a complete constraint solver for finite CSPs in *k_sat* and transform ψ into $CSP(\psi)$ before passing it on to the constraint solver; the result is the `KCSP` *algorithm.*

KCSP is a decision procedure due to the fact that *k_sat* is so if *sat* returns a Boolean assignment whenever the input formula is satisfiable, otherwise the empty assignment; see [4].

Theorem (Total Correctness of KCSP). KCSP *is a decision procedure for* $\boldsymbol{K}$*-satisfiability.*

The solver adopted as *sat* in our implementation of KCSP is backtracking search interleaved with constraint propagation for generalized arc-consistency. Furthermore, the input formula is transformed into conjunctive normal form.

3 Experimental Assessment and Optimisations of KCSP

We provide an empirical evaluation of KCSP, using the Heuerding and Schwendimann (HS) test set [6] that was used at the TANCS'98 comparison of systems for non-classical logics [13]. The HS test set consists of classes of formulas for $\boldsymbol{K}$, which are either provably false (labelled with p) or satisfiable (labelled with n). One tests formulas from each class, starting with the easiest instance, until the satisfiability status of a formula can not be determined within 100 seconds. The result from this class will then be a parameter (ranging from 0 to 21) of the largest formula that can be solved within the time limit. It is important to note that the formula size is exponential in this parameter. A linear speed-up in processor or program speed does not change in essence the benchmark results.

Optimisations and Analysis. We implemented the KCSP algorithm in the Constraint Logic Programming (CLP) system ECL^{i}PS^e, version 5.5. We ran our experiments on an AMD Athlon Processor (1 GHz), with 512MB RAM, under Red Hat Linux 7.1. The HS formulas used in the experiments and the code for KCSP are at available at `http://www.cwi.nl/~sbrand/Research/kcsp/`.

We turn to a brief discussion of our optimisations and their impact. To get partial Boolean assignments in KCSP so that the reasoning on the boxed formulas is "delayed" (and possibly never done), we ensure that propositional variables have as domains $\{0, 1\}$, while boxed formulas have as domains $\{0, 1, 2\}$, where 2 describes "irrelevance". We add constraints to obtain a partial assignment with a small number of boxed formulas "switched on" (i.e., with a value $\neq 2$). We call these the *(assignment-) minimising constraints*. We also add heuristics to reduce the size of the KCSP search tree: the value 2 is preferred for boxed formulas, and among them for positively occurring ones. Additionally, the instantiation ordering of boxed formulas is along their increasing modal depth, that is, shallow boxed formulas are assigned first.

Minimising constraints make a substantial difference, especially in the case of the so-called *branch* formulas within the HS test set: KCSP with total assignments can only solve the first two formulas in *branch_p*, whereas KCSP with minimising constraints solves all of them in less than 2 seconds; a similar result holds for *branch_n* — see also the comparison table below.

Another optimization concerns disjunctive information. In the KCSP algorithm, formulas are transformed in CNF form before being converted into CSPs; in particular, every time 0 is assigned to a formula $\neg\Box\psi$, the subformula $\neg\psi$ is first transformed in CNF and then into CSP form. This CNF-conversion is not an efficient choice; it can be avoided by treating $\neg\psi$ as a disjunctive constraint $\neg\psi = \bigvee_{i=1}^{n} \phi_i$. The clauses ϕ_i are reified by means of link variables L_i, which are constrained to contain at least one that is set to true. Avoiding CNF conversion by means of disjunctive constraints has a substantial effect; e.g., $ph\_n(4)$ — an instance of the pigeon-hole problem — is now solved in a few seconds but with CNF conversion KCSP halts due to a lack of memory.

Next, we added constraints for factoring. Consider a subformula $\Box\psi$ of ϕ, the input 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 avoid this, we add a constraint $C_{\Box\psi}$ on the CSP variables for $\Box\psi$ which states that no two variables x_i, x_k (representing $\Box\psi$) exist with $x_i = 0$ and $x_k = 1$. This form of factoring is beneficial for formulas with the same boxed subformula occurring repeatedly.

Finally, we added simplifications. These take place only once, upon reading the formula. We use standard simplification rules for propositional formulas, at all layers, in a bottom-up fashion. Simplification makes an important different in the case of the *lin* formulas in the HS test set.

Results and a Comparison. The table on the right-hand side 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 results for KSATC are taken from [8]; there KSATC was run on the Heuerding and Schwendimann test set, on a 350 MHz PentiumII with 128 MB 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 out. For some classes, KCSP clearly outperforms KSATC, for some it is the other way around, and for yet others the differences do not seem significant. E.g., KCSP is superior in the case of *lin* and *branch* formulas; *branch_n* is often considered to be the hardest "truly modal test class" for 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. In the case of *t4*, KSATC is superior to KCSP; notice, however, that KSATC features a number of optimisations for early *modal* pruning that we have not (yet) added to 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	15	10	7	10

4 Finale

We have described a method for modeling and solving modal satisfiability problems using a constraint-based approach. Guided by the tree model property for

modal logic, the method works by stratifying modal satisfiability problems into sequences of propositional satisfiability problems, each of which is encoded as a non-Boolean CSP. Our implementation, KCSP, is competitive with the best modal-theorem provers on the hardest "truly modal class" in the Heuerding and Schwendimann test set, namely *branch*. An important advantage of KCSP is that encoding optimisations (e.g., for factoring or partial assignments) can be done very elegantly and compactly in our constraint-based setting.

Our ongoing and future work focuses on: CNF-free modelling, modal learning heuristics, the use of stronger forms of constraint propagation, and an extension of our CSP-based approach to more expressive modal-like logics.

Acknowledgments. We thank our referees for their useful comments. 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. C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based Heuristics in Modal Theorem Proving. In *Proc. ECAI 2000*, pages 199–203. IOS Press, 2000.
2. P. Blackburn and M. De Rijke. Zooming in, zooming out. *Journal of Logic, Language and Information*, 6:5–31, 1997.
3. P. Blackburn, M. De Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
4. F. Giunchiglia and R. Sebastiani. Building Decision Procedures for Modal Logics from Propositional Decision Procedures. The Case Study of Modal $\boldsymbol{K}(m)$. *Information and Computation*, 162(1–2):158–178, 2000.
5. V. Haarslev and R. Möller. RACER. URL: `http://kogs-www.informatik.uni-hamburg.de/~race/`, September 2002.
6. A. Heuerding and S. Schwendimann. A Benchmark Method for the Propositional Modal Logics $\boldsymbol{K}$, $\boldsymbol{KT}$, $\boldsymbol{S4}$. Technical Report IAM-96-015, University of Bern, 1996.
7. I. Horrocks. FaCT. URL: `http://www.cs.man.ac.uk/ ~horrocks/FaCT/`, September 2002.
8. I. Horrocks, P.F. Patel-Schneider, and R. Sebastiani. An Analysis of Empirical Testing for Modal Decision Procedures. *Logic J. of the IGPL*, 8(3):293–323, 2000.
9. MSPASS V 1.0.0t.1.2.a. URL: `http://www.cs.man.ac.uk/~schmidt/mspass`, February 23, 2001.
10. G. Pan, U. Sattler, and M. Y. Vardi. BDD-Based Decision Procedures for $\boldsymbol{K}$. In *Proceedings of CADE 2002*, pages 16–30. Springer LINK, 2002.
11. P.F. Patel-Schneider. DLP. URL: `http://www.bell-labs. com.user/pfps/dlp/`, September 2002.
12. A. Tacchella. *SAT System Description. In *Collected Papers from the International Description Logics Workshop 1999, CEUR*, 1999.
13. TANCS: Tableaux Non-Classical Systems Comparison. URL: `http://www.dis.uniroma1.it/~tancs`, January 17, 2000.