KT and S4 Satisfiability in a Constraint Logic Environment

Lynn Stevenson[1,2], Katarina Britz[1,2], and Tertia Hörne[2]

[1] Meraka Institute, CSIR, South Africa
[2] School of Computing, University of South Africa
mrslstevenson@gmail.com
arina.britz@meraka.org.za
hornet@unisa.ac.za

Abstract. The modal satisfiability problem is solved either by using a specifically designed algorithm, or by translating the modal logic formula into an instance of a different class of problem, such as a first-order logic problem, a propositional satisfiability problem, or, more recently, a constraint satisfaction problem. In the latter approach, the modal formula is translated into layered propositional formulae. Each layer is translated into a constraint satisfaction problem which is solved using a constraint solver. We extend this approach to the modal logics *KT* and *S4* and introduce a range of optimizations of the basic prototype. The results compare favorably with those of other solvers, and support the adoption of constraint programming as implementation platform for modal and other related satisfiability solvers.

1 Introduction

One of the reasoning tasks associated with modal logic is the modal satisfiability problem. This is a decision problem that returns 'yes' if an algorithm can generate some model in which a given formula is satisfiable. This problem has been researched extensively using different automated approaches. These approaches fall into two distinct categories. Either a special, purpose-build algorithm is used, or the modal formula is translated into an instance of a different class of problem, and solved with the highly optimized solvers available for that class. A widely used class of purpose-built algorithms are based on tableau methods, and include the tableau solvers FaCT [1] and DLP [2]. A well-known translation method is the translation of modal formulae into first-order logic [3]. Other translation-based approaches include translation into a propositional satisfiability problem (SAT) [4], or a constraint satisfaction problem (CSP) [5, 6].

In this paper, we further investigate the feasibility of the constraint-based approach proposed by Brand et al. [5, 6]. A modal formula is stratified into layers, each of which is solved using the constraint logic programming (CLP) language, ECL^iPS^e [7]. The benefit of this approach is the ability to make use of an existing, well-developed constraint programming language, with its mature solvers

[1] This paper was published in LNCS 5351 and is available at www.springerlink.com.

and predefined libraries of functions. A key contribution of this approach is that the domains of variables are assigned values over and above the Booleans 0 and 1: a value of u can be assigned to a variable, thereby allowing partial assignments. This has the effect of considerably speeding up finding a solution. As in many other application domains, the advantage of a constraint-based approach to the satisfiability problem is its support for sophisticated conceptual modelling by means of constraints.

The solver developed by Brand et al. is, however, limited to the modal logic K. It only deals with formulae that are in conjunctive normal form (CNF), and have not been optimized using any of the standard techniques such as caching. In this paper, we discuss the extension of this solver to the modal logics KT and $S4$. A number of enhancements to the original prototype are discussed, the most significant of which are the following: We relax the requirement that formulae be in CNF; instead, formulae are kept in a negation normal form (NNF). Extensive simplification is applied to NNF clauses using propositional unit literals (l and $\neg l$) and unit modal literals ($\Box l$ and $\neg\Box l$). Where a propositional variable occurs only positively or only negatively in a layered formula, all clauses in which it occurs are excluded from the translation into the CSP. This significantly prunes the search space. Caching of formulae and their status is introduced, which reduces reprocessing.

These enhancements return favorable results that compare well with the results of the solvers FaCT, DLP and KSAT [8]. These solvers are highly optimized, whereas our results have been obtained without optimized data structures. This means that there is a strong case for incorporating constraint methods into tableau solvers, and to develop tableau solvers using constraint programming. Our findings therefore support the use of constraint programming as a feasible environment for the implementation of modal satisfiability solvers. This approach is also applicable to related areas such as description logic reasoners.

The remainder of this paper is organized as follows: Section 2 provides the theoretical background of the KCSP solver developed by Brand et al. [6]. Sections 3 and 4 provide details of the KT_CSP and S4_CSP solvers for KT and $S4$ respectively, and discuss the results obtained using the Heuerding / Schwendimann test data sets. In Section 5, the exponential nature of the results is discussed and they are compared with those of the TANCS '98 conference. The final section (Section 6) includes details of possible further areas of research.

2 Background to the KCSP Solver

We introduce some of the terminology used in the work that follows. The reader is referred to texts such as Blackburn et al. [9] for in-depth details of modal logic.

Definition 1. *The basic modal language K is defined using a set Φ of atomic propositions, the elements of which are denoted p, p_1, …, q, q_1, …, the propositional connectives $\neg$ and $\wedge$, and the unary modality $\Box$. The set of well-formed formulae generated from Φ, denoted Fma(Φ), is generated by the rule*

$$\varphi ::= p \mid \bot \mid \neg\varphi \mid \varphi \wedge \psi \mid \Box\varphi$$

where p ranges over the elements of Φ, $\bot$ is the falsum and φ, $\psi, \ldots$ are modal formulae.

A *propositional atom* is any propositional formula that cannot be decomposed propositionally. A *propositional literal* is either a propositional atom or its negation. A *modal atom* is any modal formula that cannot be decomposed propositionally – that is, any formula whose main connective is not propositional. A *modal literal* is either a modal atom or its negation.

In the context of this paper, a modal formula can be normalized into either NNF or CNF. A CNF formula consists of the conjunction of clauses, where each CNF clause is the disjunction of propositional literals and / or modal literals. A modal formula is in NNF if negation occurs only immediately before propositional and modal atoms and the only Boolean connectives it contains are $\{\neg, \wedge, \vee\}$. A modal NNF formula consists of the conjunction of NNF clauses, where each NNF clause consists of a disjunction of NNF formulae. A *propositional unit clause* is any clause l where l is a propositional literal. A *modal unit clause* is any clause $\Box\varphi$ or $\neg\Box\varphi$ where φ is any formula. A *unit modal literal* is any clause $\Box l$ or $\neg\Box l$, where l is a propositional literal.

The satisfiability of a modal formula can be determined by translating it into layers of propositional formulae. This approach was first proposed by F. Giunchiglia and R. Sebastiani [4], and implemented in their KSAT solver. The modal atoms of a particular layer are processed as though they are propositional atoms, and truth values are assigned to them. Whenever a modal layer contains a $\neg\Box$-modality, further processing at the next modal layer is required. The KSAT solver makes use of the well-known DPLL SAT algorithm to determine satisfiability of the propositional formula at each layer.

A related approach was proposed by Brand et al. [6], and implemented in their KCSP solver. In this case, the stratified modal formula is translated into a CSP. A CSP consists of a set of variables, a domain for each variable and a set of constraints. The variables can be assigned any value in their corresponding domain, with the limitation that the constraints on the variables need to be satisfied. The constraints therefore limit the scope of the variables.

In both KSAT and KCSP, the modal formula needs to be in conjunctive normal form. The propositional approximation of the modal formula is fed to the solver, which returns a set of truth assignments to the propositional and modal atoms, termed the *top-level* atoms, at the current modal level. This is defined formally as follows:

Definition 2. *A total truth assignment μ for a modal K formula φ is a set of literals*

$$\mu = \{\Box\alpha_1, \ldots, \Box\alpha_n, \neg\Box\beta_1, \ldots, \neg\Box\beta_m, p_1, \ldots, p_r, \neg q_1, \ldots, \neg q_s\}$$

such that every top-level atom of φ occurs either positively or negatively in μ.

Theorem 1. *[4] A modal formula φ is K-satisfiable if and only if there exists a K-satisfiable truth assignment μ such that $\mu \models_p \varphi$.*

This means that the K-satisfiability of a formula φ can be reduced to determining the K-satisfiability of its truth assignments. Such an assignment is termed total when a truth value is assigned to each top-level atom; it is termed a partial assignment when the truth values ensure the satisfiability of φ. The satisfiability of the modal portion of φ is then determined as follows.

Definition 3. *The restricted truth assignment μ^r for a modal K formula φ is defined as*

$$\mu^r = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j$$

Theorem 2. *[4] The restricted truth assignment μ^r is satisfiable if and only if the formula*

$$\varphi_j = \bigwedge_i \alpha_i \wedge \neg\beta_j$$

is K-satisfiable for every $\neg\Box\beta_j$ occurring in μ^r.

Constraints are defined on clauses in the modal formula. A clause $(\neg p_1 \vee p_2)$ has the constraints that $p_1 = 0$ and / or $p_2 = 1$. A clause $\Box p_4$ has the constraint that $\Box p_4 = 1$.

A constraint solver always returns a *total assignment* to its variables. Because less computational effort is required to return a partial assignment, Brand et al. [6] followed an approach of setting the domain of each variable to $\{0, 1, u\}$, where u indicates that a truth value has not been assigned to the associated variable.

A modal formula is negated and converted into CNF before being passed to KCSP.

Algorithm 1 [6] The KCSP algorithm schema:

```
function KCSP($\varphi$)                    // succeeds if $\varphi$ is satisfiable
    $\varphi_{csp} = to\_csp(\varphi)$;
    $\mu := csp(\varphi_{csp})$;
    $\Theta = \bigwedge\{\alpha : \Box\alpha = 1$ is in $\mu\}$;
    for each $\Box\beta = 0$ in $\mu$ do
        KCSP($\Theta \wedge \neg\beta$);        // backtrack if this fails
end;
```

The procedure *to_csp* identifies the top-level atoms of φ, sets their domains to $\{0,\ 1,\ u\}$ and defines the constraints on each clause. The resulting formula is then passed to the ECL^iPS^e constraint solver with the call to *csp* which returns a truth assignment. If it contains negative literals, that is, literals to which a value of 0 has been assigned, further processing is required. Each of the modal literals $\neg\Box\beta_j$ effectively generates a new branch of the modal tree. The conjunction of each β_j and the α_i variables having $\Box\alpha_i = 1$, form the modal formula that will be processed at the next modal layer. If there are no negative modal literals, the formula is satisfiable and no further processing is required. If there are negative modal literals, the algorithm backtracks.

Using a domain of $\{0,\ 1,\ u\}$ reduces the processing requirement. If all the $\neg\Box\beta_j$ variables at a particular modal layer can be assigned a value of u, no further processing will be required.

Various optimizations have been applied to the solver to reduce the search space. These included simplification of the initial formula by applying *unit subsumption* and *unit resolution*. When unit subsumption is applied to a modal formula containing a unit clause l, every clause containing l is removed. When unit resolution is applied to a modal formula, $\neg l$ is removed from every clause in which it occurs. Further details of these optimizations are provided in [6, 5].

3 The KT_CSP Solver

The KCSP prototype described in the previous section yielded some promising results, but did not yet establish the constraint-based approach to modal satisfiability as viable alternative to existing solvers. This is due to two factors: Firstly, few of the standard optimisations to tableau solvers were implemented, so the modelling benefits could not be appreciated fully, and secondly, the prototype only addressed the modal logic K. We address these issues below.

To extend KCSP to the modal logic KT, the solver was modified to allow for a reflexive accessibility relation. This introduces two challenges – the number of clauses in a formula is significantly increased, and the formula is no longer in CNF. To address the first challenge, we propose the following lemma, which has the effect of reducing the number of clauses generated. Full details including further examples and proofs of lemmas are available in [10].

Lemma 1. *Applying the axiom $\Box^n\varphi \rightarrow \varphi$ at each modal layer, to each occurrence of $\Box^n\varphi$, is a sound and complete strategy to enforce the reflexivity of R in the KT_CSP algorithm.*

However, applying the lemma yields a formula which is no longer in CNF. We find that, when Lemma 1 is applied to a modal clause with n positive modal literals, and the resulting formula is converted to CNF, the original modal clause is replaced by 2^n modal clauses. There is thus an exponential increase in the number of clauses when a modal formula is converted to CNF.

Two prototypes were developed to deal with reflexivity. In the one, the formulae were retained in CNF and in the other, they were not. We discuss only the second prototype since it produced better results. The approach followed when dealing with clauses that are not in CNF is illustrated by the following example.

Example 1. Consider the modal formula

$$\varphi = \Box\psi \wedge (\neg\Box\psi_1 \vee (\neg\Box\psi_2 \wedge \neg\Box\psi_3)).$$

When we convert it to CNF and apply Lemma 1, we get

$$\varphi' = \Box\psi \wedge \psi \wedge (\neg\Box\psi_1 \vee \neg\Box\psi_2) \wedge (\neg\Box\psi_1 \vee \neg\Box\psi_3).$$

There are several possible truth assignments that the constraint solver can return: $\mu_1 = \{\Box\psi, \psi, \neg\Box\psi_1\}$, $\mu_2 = \{\Box\psi, \psi, \neg\Box\psi_1, \neg\Box\psi_2\}$, $\mu_3 = \{\Box\psi, \psi, \neg\Box\psi_1, \neg\Box\psi_3\}$ and $\mu_4 = \{\Box\psi, \psi, \neg\Box\psi_2, \neg\Box\psi_3\}$.

If we do not convert the formula to CNF, and apply Lemma 1, we can verify the satisfiability of $\varphi_1 = \Box\psi \wedge \psi \wedge \neg\Box\psi_1$ and $\varphi_2 = \Box\psi \wedge \psi \wedge \neg\Box\psi_2 \wedge \neg\Box\psi_3$, with the proviso that φ_2 is processed only if φ_1 is not satisfiable. The possible truth assignments the constraint solver will return, are $\mu_5 = \{\Box\psi, \psi, \neg\Box\psi_1\}$ and $\mu_6 = \{\Box\psi, \psi, \neg\Box\psi_2, \neg\Box\psi_3\}$.

Now suppose $\neg\psi_1$ is unsatisfiable. For a complex formula, this could take considerable resources to establish. When the formula is converted to CNF, $\neg\psi_1$ is processed three times, causing backtracking each time; if it is not converted, it is processed only once. $\dashv$

We therefore propose that, instead of converting the formula into CNF, it is retained in NNF. We first apply Lemma 1, and then *selectively construct* the formula to convert to a CSP.

Definition 4. *An NNF clause ψ in a modal formula φ is represented as*

$$\psi = \psi' \vee \theta_1 \vee \ldots \vee \theta_n, \textit{ where}$$

$$\psi' = \bigvee\{l : l \in P\} \vee \bigvee\{\Box\alpha : \Box\alpha \in B^+\} \vee \bigvee\{\neg\Box\beta : \Box\beta \in B^-\},$$

P is a set of propositional literals, B^+ and B^- are sets of modal atoms, and θ_1, …, θ_n are NNF formulae.

After the modal formula has been negated and converted to NNF, the following algorithm is applied.

Algorithm 2 The KT_CSP algorithm schema:

```
function KT_CSP($\varphi$)                          // succeeds if $\varphi$ is satisfiable
    $\varphi_{kt} = apply\_reflexivity(\varphi)$;
    $\varphi_{formula} = construct\_formula(\varphi_{kt})$;
    $\varphi_{csp} = to\_csp(\varphi_{formula})$;
    $\mu := csp(\varphi_{csp})$;                     // backtrack if this fails
    $\Theta = \bigwedge\{\alpha : \Box\alpha = 1$ is in $\mu\}$;
    for each $\Box\beta = 0$ in $\mu$ do
        KT_CSP($\Theta \wedge \neg\beta$);            // backtrack if this fails
end;
```

Lemma 1 is applied to the input formula, after which the function *construct_formula* proceeds as follows: Each clause in φ_{kt} is grouped as per Definition 4. The formula $\varphi_{formula}$ is constructed as the conjunction of the ψ' components of each NNF clause. If there is no ψ' component in an NNF clause, the θ_1 component is used instead. This formula is then converted into a CSP and fed to the constraint solver. If the constraint solver cannot find a solution, it backtracks to *construct_formula* and a new formula is built using the remaining θ_i clauses.

This prototype was tested using the Heuerding / Schwendimann data sets [11] which consist of nine classes of 21 provable formulae (the 'p' data sets) and 21 unprovable formulae (the 'n' datasets). Data sets are available for each of the logics *K*, *KT* and *S4* and can be downloaded off the Internet [12]. The formulae in each class get progressively more complex. The capability of a solver is measured in terms of the number of 'p' and 'n' data sets it can solve in a particular class in *less than* 100 CPU seconds. Thus, if a result of 3 is recorded for the k_branch_n data sets, this means that only the first 3 of the 21 data sets could be solved in under 100 CPU seconds. Whenever all 21 data sets are solved, the symbol '>' is used. The initial prototype did not return particularly good results. Table 1 lists the results per class, for each category. A series of enhancements was therefore applied.

Table 1. Initial Results of the KT_CSP Prototype

	kt_branch	kt_45	kt_dum	kt_grz	kt_md	kt_path	kt_ph	kt_poly	kt_t4p
n	3	8	14	>	5	10	7	2	3
p	2	8	5	9	4	2	4	>	3

Propositional and modal simplification: In the KCSP solver, both subsumption and unit resolution are applied to the initial modal formula. In the KT_CSP prototype, the modal formula is no longer in CNF, making this approach more complex. In addition, because the application of Lemma 1 generates further propositional variables, simplification is now required at each modal layer. We therefore reconsider the unit subsumption and unit resolution rules applied in the DPLL SAT procedure and extend them to include NNF clauses and unit modal literals. The resulting reduction in the number of choice points leads to a significant improvement in the results obtained.

Enhancement 1 *We apply the following rules after the application of Lemma 1 to the modal formula at each modal layer:*

1. *For every clause ψ that is either a propositional unit clause or a unit modal literal, unit subsumption is applied to every other NNF clause containing ψ and such clauses are removed, provided that ψ does not occur in a modal formula within the NNF clause.*
2. *For every clause ψ that is either a propositional unit clause or a unit modal literal, unit resolution is applied and $\neg\psi$ is removed from every other NNF clause in which it occurs, provided that $\neg\psi$ does not occur in a modal formula within the NNF clause.*

This process is repeated until no further simplification is possible.

Early pruning: Lemma 1 and then simplification are applied to each modal formula. An analysis of some of the data sets showed that we can have a scenario at the next modal layer where, after processing a number of the $\varphi_j = \bigwedge_i \alpha_i \wedge \neg\beta_j$ formulae, a φ_j can be encountered which has p in some α_i and $\neg p$ in $\neg\beta_j$. Such a formula is unsatisfiable. Its early detection prevents unnecessary processing. This observation led to the following enhancement:

Enhancement 2 *Let $\varphi' = \psi_1 \wedge \bigwedge_j \neg\beta_j$ where $\psi_1 = \bigwedge_i \alpha_i$. If a propositional unit clause l in $\bigwedge_j \neg\beta_j$ also occurs in ψ_1, force a backtrack to the previous modal layer.*

Grouping of clauses: By grouping disjoint clauses and processing each group separately, the number of choice points can be reduced.

Enhancement 3 *Suppose we have a modal formula φ. Let $\gamma = \{p_1, \ldots, p_n\}$ be the set of propositional atoms p_i occurring in φ, where p_i can occur at any modal layer in φ. Partition the clauses in φ as follows:*

$$\varphi = \psi_1 \wedge \ldots \wedge \psi_m,$$

where each ψ_i is a conjunction of NNF clauses and for each $p_k \in \gamma$, if p_k occurs in ψ_i, then p_k does not occur in any other ψ_j, where $j \neq i$. By determining the satisfiability of each ψ_i, we determine the satisfiability of φ.

Value assignments: Any top-level propositional literal that *only* occurs positively or that *only* occurs negatively in the modal formula may, without loss of generality, be assigned a value of 1 or 0 respectively. Therefore, when the CSP for these literals is constructed, their domains can be limited to $\{1\}$ and $\{0\}$ respectively, instead of to $\{0, 1, u\}$. If a clause contains a positive propositional literal p to which a value of 1 has been assigned, this clause is $True$ since the clause is the disjunction of variables. We therefore do not need to pass this clause to the constraint solver. Again, by reducing the number of clauses in the CSP, we reduce the number of choice points. Note that this case differs from unit subsumption in that we are now dealing with propositional literals that are not propositional unit clauses.

Enhancement 4 *Suppose we have a modal formula φ. Let $\gamma = \{p_1, \ldots, p_{n1}, \neg q_1, \ldots, \neg q_{n2}\}$, where if $p_i \in \gamma$, then p_i occurs only positively in φ and if $q_j \in \gamma$, then q_j occurs only negatively in φ. We apply the following rule to the clauses in φ: For each clause ψ in φ, if ψ contains a single propositional literal p and $p \in \gamma$, then this clause is removed from φ.*

Caching: Some formulae repeat at various nodes – in some cases, there is a prolific propagation of the same formula. A solution to this problem is to be found in the introduction of a cache.

Enhancement 5 *Formulae, together with their satisfiability status, are stored in a cache. Before a new formula φ is processed, the store is checked to see if it has already been cached. If it has, and has been marked as satisfiable, no further processing is required. If it has been marked as unsatisfiable, a backtrack is forced.*

If φ has not yet been processed, check to see if it is a subformula of any modal formula φ' that has been cached. If this is the case and φ' has been marked as satisfiable, no further processing is required. Otherwise, if a subformula of φ has been cached and marked as unsatisfiable, a backtrack is forced.

If no information is available in the cache for φ, it is added to the cache with a status of unsatisfiable. It is then processed, and only if it is found to be satisfiable is its status in the store updated.

Enhancements 1–5 significantly improved the results of Table 1. The final results are provided in Table 2.

Table 2. Final results of the KT_CSP prototype

	kt_branch	kt_45	kt_dum	kt_grz	kt_md	kt_path	kt_ph	kt_poly	kt_t4p
n	10	>	>	>	6	>	7	10	>
p	>	>	>	>	5	>	4	>	>

4 The S4_CSP Solver

Applying Lemma 1 to enforce reflexivity, and the axiom $\Box\varphi \rightarrow \Box\Box\varphi$ to enforce transitivity of the accessibility relation in *S4*, leads to the same looping behavior experienced by other solvers. In addition to looping, we note that the modal depth of a formula can remain unchanged at each successive modal layer. To deal with modal *S4* formulae, we introduce the following two lemmas, the latter of which defines the *stopping condition* which prevents the algorithm from looping when enforcing transitivity.

Lemma 2. *Applying the axiom $\Box^n\varphi \rightarrow (\Box\Box\varphi \wedge \Box\varphi \wedge \varphi)$ at each modal layer to each occurrence of $\Box^n\varphi$ is a sound and complete strategy to enforce reflexivity and transitivity of R in the S4_CSP algorithm.*

Lemma 3. *Suppose we have a modal formula φ at modal layer n and suppose φ is also the modal formula generated at modal layer $n+1$. No further processing of this branch is required, as it is satisfiable.*

The S4_CSP algorithm differs from the KT_CSP algorithm in two ways - Lemma 2 is applied to the modal formula at each modal layer, instead of Lemma 1 and loop checking is implemented, as per Lemma 3.

The results obtained by the initial S4_CSP prototype are listed in Table 3. The results of the *s4_45* data sets are particularly bad. Note that the p-data sets generally returned better results than the n-data sets.

Table 3. Initial results of the S4_CSP prototype

	s4_branch	s4_45	s4_grz	s4_ipc	s4_md	s4_path	s4_ph	s4_s5	s4_t4p
n	8	2	>	8	8	9	7	5	10
p	>	1	>	>	10	10	4	5	13

Simplification and early pruning revisited: Consider the two clauses $(p_4 \vee (p_2 \wedge (\Box p_1 \vee \Box p_0))) \wedge \Box p_0$. When Lemma 2 is applied, they become $(p_4 \vee (p_2 \wedge ((\Box\Box p_1 \wedge \Box p_1 \wedge p_1) \vee (\Box\Box p_0 \wedge \Box p_0 \wedge p_0)) \wedge (\Box\Box p_0 \wedge \Box p_0 \wedge p_0)$. We now have a set of clauses which is difficult to simplify. However, if simplification was applied *before* Lemma 2, we would have the formula $(p_4 \vee p_2) \wedge \Box p_0$ which is much easier to deal with. This leads to the following enhancements:

Enhancement 6 *For each unit modal literal ψ in a modal formula φ, we apply the rules of Enhancement 1 to every other clause containing ψ before Lemma 2 is applied to φ.*

Enhancement 7 *We apply the following two simplification rules to every propositional unit clause and unit modal literal ψ in a modal formula φ:*

1. *NNF-unit subsumption is applied to every other clause containing ψ:*
   A formula $\varphi_1 \wedge (\psi_1 \vee (\psi \wedge \psi_2)) \wedge \psi$, in which φ_1 consists of the conjunction of any number of NNF clauses and ψ_1 and ψ_2 are NNF clauses, is replaced with $\varphi_1 \wedge (\psi_1 \vee \psi_2) \wedge \psi$.
2. *NNF-unit resolution is applied to every other NNF clause containing $\neg\psi$:*
   A formula $\varphi_1 \wedge (\psi_1 \vee (\neg\psi \wedge \psi_2)) \wedge \psi$, whose variables are defined as in 1. above, is replaced with $\varphi_1 \wedge \psi_1 \wedge \psi$.

Enhancements 6–7, together with a re-implementation of enhancement 2, led to a considerable improvement in results, particularly for the *s4_45* datasets. The final results of the S4_CSP solver are listed in Table 4.

Table 4. Final results of the S4_CSP prototype

	s4_branch	s4_45	s4_grz	s4_ipc	s4_md	s4_path	s4_ph	s4_s5	s4_t4p
n	8	14	>	8	9	20	8	6	12
p	>	12	>	>	12	19	4	5	13

5 Comparison of Results

In order to gain further insights into these results, it is necessary to compare them with the results of existing state-of-the-art solvers. Few solvers can effectively deal with the modal logics *KT* and *S4* – our research showed that only FaCT and DLP have been optimized for *S4*. See [10] for further details.

As already discussed, the formulae in each class of data sets get progressively more complex. This causes results such as:

Data set number	9	10	11	12	13
CPU secs	28.17	95.14	360	1320	6605

in the case of *kt_branch_n*. These results clearly demonstrate exponential behavior, which justifies the comparison of our results with those of the TANCS-1998 competition, which was based on the Heuerding / Schwendimann data sets. The results obtained for the *KT* data sets by the KSAT, DLP, FaCT solvers [8], together with the results of the KT_CSP solver, are listed in Table 5. In Table 6, we list the results obtained for the *S4* data sets by the DLP, FaCT and S4_CSP solvers (the KSAT solver does not support *S4*). As can be seen, the results of the KT_CSP and S4_CSP prototypes compare favorably.

If we were to rerun the TANCS-1998 results on the hardware used for our benchmark results, the results would obviously be much better. For example, the DLP solver would inevitably solve all 21 *kt_ph_n* data sets. However, in the case of data sets such as *kt_branch_n*, the improvement would be in the order of 1 to at most 4 data sets. If one considers the results for *kt_branch_n* listed above, the hardware would need to be considerably faster to solve the 12^{th} data set in under 100 CPU seconds.

Table 5. Results of the Heuerding / Schwendimann KT data sets

	FaCT	**DLP**	**KSAT**	**KT_CSP**	**FaCT**	**DLP**	**KSAT**	**KT_CSP**
	n	**n**	**n**	**n**	**p**	**p**	**p**	**p**
kt_45	>	>	5	>	>	>	5	>
kt_branch	4	11	7	10	6	16	8	>
kt_dum	>	>	12	>	11	>	7	>
kt_grz	>	>	>	>	>	>	9	>
kt_md	5	>	4	6	4	3	2	5
kt_path	3	>	5	>	5	6	2	>
kt_ph	7	18	5	7	6	7	4	4
kt_poly	7	6	2	10	>	6	1	>
kt_t4p	2	>	1	>	4	3	1	>

Table 6. Results of the Heuerding / Schwendimann S4 data sets

	FaCT	**DLP**	**S4_CSP**	**FaCT**	**DLP**	**S4_CSP**
	n	**n**	**n**	**p**	**p**	**p**
s4_branch	4	8	8	4	10	>
s4_45	>	>	14	>	>	12
s4_grz	>	>	>	2	9	>
s4_ipc	4	>	8	5	10	>
s4_md	4	>	9	8	3	12
s4_path	1	>	20	2	3	19
s4_ph	4	18	8	5	7	4
s4_s5	2	>	6	>	3	5
s4_t4p	3	>	10	5	>	13

6 Conclusion and Future Work

A strength of translating each modal layer into a constraint satisfaction problem lies in the ability to limit the domain of a propositional variable which occurs only positively or only negatively to a single value. This allows us to reduce the number of conjunctive clauses in the modal layer, thereby significantly reducing the search space. Because the modal problem has been stratified into layers, this is easily implemented.

Although we obtained good results, we have identified areas for further improvement. Firstly, the selection criteria applied when constructing the CSP from the NNF formula can be further enhanced. Secondly, further improvement is possible by optimizing the data structures to exploit normal forms for modal logics. Finally, our results support the adoption of constraint programming as underlying formalism for description logic reasoners. This should lead to improved scope for taking advantage of the improved modelling opportunities provided by the constraint-approach, especially when dealing with more expressive description logics. We are currently investigating this further.

References

1. Horrocks, I.: The FaCT System. In: TABLEAUX'98. Volume 1397 of LNCS. Springer, Heidelberg (1998) 307–312
2. Patel-Schneider, P.F.: DLP system description. In: Proceedings of the 1998 International Workshop on Description Logics Workshop (DL'98). Volume 11. CEUR-WS.org (1998) 87–89
3. Ohlbach, H., Nonnengart, A., de Rijke, M., Gabbay, D.: Encoding two-valued nonclassical logics in classical logic. In: Handbook of Automated Reasoning. Elsevier Science (2001) 1403–1486
4. Giunchiglia, F., Sebastiani, R.: Building Decision Procedures for Modal Logics from Propositional Decision Procedure. The Case Study of Modal K. In: Cade-13. Volume 1104 of LNCS. Springer, Heidelberg (1996) 583–597
5. Brand, S., Gennari, R., de Rijke, M.: Constraint programming for modelling and solving modal satisfiability. In: CP 2003. Volume 2833 of LNCS. Springer, Heidelberg (2003) 795–800
6. Brand, S., Gennari, R., de Rijke, M.: Constraint methods for modal satisfiability. In: Recent Advances in Constraints. Volume 3010 of LNCS. Springer, Heidelberg (2004) 66–86
7. Wallace, M.G., Novello, S., Schimpf, J.: ECLiPSe: A platform for constraint logic programming. ICL Systems Journal **12**(1) (1997) 159–200
8. Horrocks, I., Patel-Schneider, P.: FaCT and DLP: Automated reasoning with analytic tableaux and related methods. In: TABLEAUX'98. Volume 1397 of LNCS. Springer, Heidelberg (1998) 27–30
9. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge, UK (2001)
10. Stevenson, L.: Modal Satisfiability in a Constraint Logic Environment. University of South Africa (2008) MSc dissertation.
11. Balsiger, P., Heuerding, A., Schwendimann, S.: A benchmark method for the propositional modal logics K, KT, S4. Journal of Automated Reasoning **24**(3) (2000) 297–317
12. Jaeger, G., Balsiger, P., Heuerding, A., Schwendimann, S.: K, KT, S4 test data sets Available at http://www.iam.unibe.ch/∼lwb/benchmarks/benchmarks.html, retrieved September 2007.