Chapter 9

Diamonds and Constraints

9.1 Introduction

9.1.1 Motivations

In Chapter 8 we base the satisfiability procedure for $K$ and $K(Index)$ on a translation from modal logic into the layered fragment of first-order logic. This translation goes through an intermediate translation which maps modal formulas into formulas of an intermediate multimodal language. This intermediate translation itself possesses interesting features:

- it encodes a very strong form of the tree model property,
- it preserves satisfiability,
- and, as a consequence, it could be directly employed to check the satisfiability of the original modal formulas.

The work of [GS00] exploits similar semantic intuitions but without appealing to the tree model property directly. In fact, Giunchiglia and Sebastiani prove that a refinement of a SAT solver, called $K$-SAT, can be used for $K(Index)$ theorem proving: this decision procedure for $K(Index)$ logics is built on top of the Davis-Logemann-Loveland (DP) procedure for propositional logics. The SAT solver $\text{DP}$ is called on a modal formula. The propositional satisfiability of the input formula is first checked. If the result is positive, search is not abandoned, but the modal components of the formula are examined: one by one, each subformula with a diamond as its main operator is checked against all the subformulas that have a box as their main operator; this is done by means of $\text{DP}$ again.

In this chapter, we follow a similar approach: however, instead of using $\text{DP}$, we want to use constraint propagation (see Chapter 4) and solving algorithms. At this early stage of our work, we are not aiming to be competitive with today's high-performance modal provers, such as DLP [PS02], FaCT [Hor02] and RACER [HM02].
Our aim in this chapter is to explore to what extent existing constraint satisfaction techniques, developed for propositional satisfiability, can be used in automated theorem proving for modal logics.

### 9.1.2 Outline and Structure

In this chapter, we propose to use constraint solving and propagation algorithms to determine the satisfiability of modal logics. These procedures are all based on constraint algorithms for propositional formulas, as the aforementioned \( K \)-SAT algorithm is based on \( \text{DP} \).

Our work in this chapter consists of two stages. The modal formula is first transformed into a CSP, see Subsection 9.4.2. Then a preliminary constraint based procedure for this encoding is proposed in Subsection 9.4.3. Search for solutions is alternated with a hyper-arc consistency algorithm, as described in Chapter 4. We sketch a proof of the correctness and completeness of the procedure for \( K \) formulas. A refinement of this procedure and alternatives to it, still based on CSP solvers, are proposed in Section 9.6.

In the literature, a number of constraint propagation and solving algorithms have been studied for reasoning about satisfiability problems. Thus, our work constitutes a first attempt to exploit well-known and corroborated techniques of constraint propagation and satisfaction for these problems to tackle modal satisfiability. We also report on preliminary experimental work aimed at testing the procedures proposed in this chapter on benchmark modal formulas.

### 9.2 The SAT Based Approach

In the remainder of this chapter, we implicitly assume that we are dealing with modal languages in which every occurrence of the \( \Diamond \) operator has been replaced by \( \neg \Box \).

**Convention 9.2.1.** We assume that, in the modal language \( \mathcal{ML}(P) \), each occurrence of all modal operators \( \Diamond \) is replaced by the equivalent \( \neg \Box \neg \).

This will avoid that the solver treats modal formulas such as \( \Diamond \neg p \) and \( \Box p \) in a different way.

The following further convention on propositional formulas is consistent with the actual implementations of the \( \text{DP} \) procedure as in [Seb97], and does not constitute a theoretical limitation; see also *ib*.

**Convention 9.2.2.** The set of atoms in propositions or clause sets are totally ordered.
Convention 9.2.2 avoids that theorem provers based on DP treat formulas such as $p \lor q$ and $q \lor p$ as different. The original DP procedure receives as input a CNF formula and determines whether the formula is satisfiable or not. The basic K-SAT algorithm, displayed as Algorithm 9.2.1, applies a modified version of the DP procedure, recursively, on sequences of modal formulas.

We interpret what the K-SAT procedure does by resorting to the tree model property, see Definition 8.3.2. We recall that the intermediate translation (see Definition 8.4.1) is already based on this property: i.e., a multimodal $K(\text{Index})$ formula $\psi$ is satisfiable iff it is so at the root of a tree-like model. This means that subformulas of $\psi$ can be evaluated layer by layer: first subformulas in layer 0 are evaluated; if these are found consistent, subformulas at deeper layers are evaluated, in a top-down manner. This is also what Algorithm 9.2.1 does in tree-like terms.

**Algorithm 9.2.1: K-SAT($\psi$)**

```plaintext
procedure K-SAT($\psi$)
return K-SAT_W($\psi$, true);

procedure K-SAT_W($\psi$, $\mu$)
if $\mu = false$ then return false; % backtrack
if $\mu = true$ then return K-SAT_A($\mu$);
if a unit disjunct $L$ is in $\psi$ then
    return K-SAT_W(Unit-propagate($L$, $\psi$, $\mu \land L$));
L := Select-branch-variable($\psi$);
return K-SAT_W(Unit-propagate($\psi$, $L$, $\nu \land L$)) or
K-SAT_W(Unit-propagate($\psi$, $\neg L$, $\nu \land \neg L$))

procedure K-SAT_A($\mu$)
$\Phi := \land \{ \phi : \Box \phi \text{ occurs in } \mu \}$;
for each $\Box \theta$ such that $\neg \Box \theta$ occurs in $\mu$ do
    $\theta := \neg \theta$;
if not K-SAT($\theta \land \Phi$) then return false; % backtrack
return true;
```

Algorithm 9.2.1 presents only the basic version for $K$, restricted to CNF formulas. Thus, given a CNF formula $\phi$ and a literal $L$ of $\phi$, Unit-propagate performs unit propagation, which is explained as below.

**Definition 9.2.3.** Given a CNF formula $\phi$ and a literal $L$ in $\phi$, unit propagation of $L$ in $\phi$ consists of the following procedures:

- unit resolution: replace each disjunct of the form $\neg L \lor \psi$ by $\psi$ in $\phi$;
• unit substitution: remove each disjunct of the form \( L \lor \psi \) from \( \phi \).

The original algorithm \( K\text{-SAT} \) can deal with formulas of any format by refining \( \text{Unite-propagate} \), see [GS00]. The correctness of Algorithm 9.2.1 is based on the result below. To formulate it, we need to briefly explain how a propositional assignment can generate a formula.

**Definition 9.2.4.** Consider a finite propositional language and an assignment \( \mu \) of truth values to the set of its proposition letters \( P \): i.e., \( \mu : P \mapsto \{0,1\} \). Then the formula generated by \( \mu \) is

\[
\bigwedge \{ p_i : \mu(p_i) = 1 \} \land \bigwedge \{ \neg p_j : \mu(p_j) = 0 \}.
\]

We denote the formula generated by a propositional assignment \( \mu \) by \( P(\mu) \). The above definition is essential for the formulation of the following result; for a proof, see [Seb97].

**Theorem 9.2.5.** Consider a modal language \( \mathcal{ML} \) and an \( \mathcal{ML} \) formula \( \phi \) of the form

\[
\bigvee_{j} \Box \phi_{j}^{i} \lor \bigvee_{j'} \neg \theta_{j'}^{i} \lor \bigvee_{l} L_{l}^{i}.
\]

Then \( \phi \) is satisfiable in an \( \mathcal{ML} \) model iff there exists a truth-value assignment \( \mu \) to the propositional variables in the propositional language

\[
P := \{ p : p \text{ occurs in some } L_{l}^{i} \text{ in } \phi \} \cup \{ \Box \phi_{j}^{i}, \Box \theta_{j'}^{i} \in \phi \}
\]

such that the \( P \) formula \( P(\mu) \) generated by \( \mu \) is \( \mathcal{ML} \) satisfiable, and \( \mu \) satisfies the \( P \) formula \( \phi \).

### 9.3 Constraint Satisfaction and SAT Formulas

There is one obvious way of reformulating a SAT problem as a CSP. It requires a proposition to be reduced to its CNF; hence each resulting conjunct is regarded as a constraint. For instance, the disjunction

\[
\neg x \lor y \lor z
\]

is regarded as the constraint \( C(x, y, z) \), the explicit description of which only rules out the triple \((1, 0, 0)\) from the interpretation domains of the variables \( x, y \) and \( z \). In this formulation, arc consistency or hyper-arc consistency can take the place of unit propagation as proved in [Apt00b].
In the remainder of this section, we focus on the aforementioned encoding of formulas as (9.1) into constraints, and analyze a constraint propagation and solving algorithm for it. In Section 9.6, we suggest possible improvements to this, and how a different encoding of formulas as CSPs could be used for performing modal automated reasoning.

At this point, we fix the type of CSPs we deal with in the remainder of the present chapter.

**Definition 9.3.1.** A Boolean CSP $P := (X, D, C)$ has domains that only contain 0 or 1.

Hence, Boolean constraints have the same domains and only differ in the chosen representation of constraints in this chapter; see [Wal00] for three other different encodings of propositional formulas as CSPs.

### 9.3.1 Mapping CNF Formulas into Constraints

Given Definition 9.3.1, it is not difficult to map CNF formulas into equivalent Boolean constraints. In this subsection, a constraint corresponds to a disjunction of literals — these are atoms or their negation, see Subsection 8.2.1. An explicit representation of such a constraint gives just the truth assignments that satisfy the clause. We provide the translation below.

**Definition 9.3.2.** Consider a CNF formula $\psi$. First, remove all propositional tautologies from $\psi$ and let $\psi'$ be the resulting CNF formula. Denote by $At$ the ordered set of atoms that occur in $\psi'$ (see Convention 9.2.2). Then apply the following procedure on all disjuncts $\phi$ in $\psi'$:

- if $\phi$ is a unit disjunct $L$, then: if $L$ is a propositional atom $p_i$, set $D_i := \{1\}$; if $L$ is the negation of a propositional atom $p_i$, then set $D_i := \{0\}$;
- else, create a constraint $C(\phi)$ on the ordered set of variables $\{p_1, \ldots, p_m\}$ that occur in $\phi$: a tuple $d$ of 0's and 1's belongs to $C(\phi)$ iff the set of truth assignments
  \[
  \mu := \{ p_{ij} \mapsto d[i_j] : j = 1, \ldots, m \}
  \]
  satisfies the formula $\phi$;
- associate the domain $\{0, 1\}$ with all the atoms that do not occur in unit clauses in $\psi'$.

Denote by $T_{SAT}^{CSP}(\psi)$ the resulting Boolean CSP.

The following result clearly holds.

**Fact 9.3.3.** An assignment $\mu$ satisfies a CNF formula $\psi$ iff its restriction to the variables in $T_{SAT}^{CSP}(\psi)$ satisfies $\psi$ viewed as a Boolean CSP. \qed
9.3.2 Constraint Solving Algorithms

In Subsection 4.1.1, we briefly discussed constraint solving algorithms, by mentioning two well-known schemas: generate and test (GT) and backtracking (BT). We now provide some more details on them. In GT, all variables are instantiated (generation), and the resulting total assignment is tested against the problem constraint. In the BT schema, variables are instantiated sequentially; as soon as all the variables relevant to a constraint of the problem are instantiated, the resulting partial assignment is checked against the constraint. Whenever a partial assignment violates any of the constraints, BT backtracks to the last instantiated variable, whose current domain is non-empty. Thus BT searches through the space for solutions in a depth-first manner, see [Kum92].

In this subsection, we discuss a constraint solving methodology based on BT: this amounts to embedding one of the constraint propagation algorithms that we explained in Chapter 2 in BT. We explain this below, following the presentation of [Kum92].

The generic BT schema with some form of constraint propagation receives a CSP as input and computes a new CSP at each step. For instance, suppose that constraint propagation amounts to enforcing hyper-arc consistency in BT; then hyper-arc consistency is performed on each CSP that is step-by-step computed in the BT algorithm. If the current CSP has singleton domains (i.e., variable domains have precisely 1 element) and is hyper-arc consistent, then the problem is solved: i.e., the solution consists in assigning to the variables the unique value found in their respective domains. If during constraint propagation the domain of any variable becomes empty, then this CSP is removed from the search space. Otherwise, one of the variables, whose current domain has more than one element, is selected and a new CSP is computed, for each possible assignment of this variable. A BT algorithm checks the consistency of these computed CSPs in a depth-first manner until a solution is generated.

This brief outline is just one of many algorithms based on the schema “BT + constraint propagation”. Those algorithms differ in the backtracking method used and, more interestingly for us, in the specific constraint propagation algorithm used. Moreover, there is not just the issue of which constraint propagation algorithm should be employed, we also have to decide when to use the algorithm, and till what point in the search space. In what follows, we shall not concern ourselves with this last aspect of backtracking schemas: a good introduction to this is still, to our knowledge, [Kum92]. Instead, in Subsection 9.3.3 below, we shall focus our attention on a specific algorithm schema that fits in the “BT+constraint propagation” methodology: forward checking.
9.3.3 The Forward Checking Algorithm Schema

Forward Checking (FC) is an algorithm schema that works like BT, except that domains of still unassigned variables change dynamically: when a variable \( x_i \) is assigned a value, the algorithm performs hyper-arc consistency checks on domains of variables that are still unassigned, by inspecting constraints on those variables and \( x_i \). When hyper-arc inconsistency is detected, backtracking to the last assignment for \( x_i \) occurs; another value is assigned and, if this is inconsistent with a constraint on it, backtracking resorts to the variable instantiated before \( x_i \). Nowadays, there are a number of variants of this basic schema; they mainly differ in the choice of domains and constraints on which hyper-arc consistency is performed, see [BMFL02]. The algorithm schema FC is displayed as Algorithm 9.3.1 below. Variables are partitioned in two sets, \( A \) and \( U \):

- \( A \) stores the set of variables that occur in the current assignment;
- \( U \) stores the remaining set of variables.

The algorithm HAC is called when the currently inspected variable \( x_i \) is assigned a value \( a \in D_i \), and there are constraints in the current CSP \( P := (X,D,C) \) that involve \( x_i \). Thus HAC performs a limited form of hyper-arc consistency on the problem which has constraints \( C(s) \) such that \( x_i \) is in \( s \), and domains of the form \( D_j \) such that \( x_j \) is in \( s \). The specific choice of which and how many variables \( x_j \), from \( U \) or \( A \), can occur in the constraints \( C(s) \) of this problem varies according to the specific FC algorithm chosen.

Algorithm 9.3.1: FC(\( A, U, \mu, D \))

\[
\begin{align*}
\text{if } F = \emptyset \text{ then return } \mu; \\
\text{choose } x_i \text{ from } U; \\
\text{stop} := \text{false}; \\
\text{while } D_i \neq \emptyset \text{ and not stop do} \\
\text{choose } a \in D_i; \\
Temp\mu := \mu \cup \{ x_i := a \}; \\
TempD := \text{HAC}(x_i,a,U - \{ x_i \},D - \{ D_i \}); \\
\text{if not } TempD = \emptyset \text{ then} \\
\text{return FC}(A \cup \{ x_i \},U - \{ x_i \},Temp\mu,TempD); \\
\text{else stop} = \text{true}; \% \text{ backtrack}
\end{align*}
\]

Walsh [Wal00] provides a theoretical analysis of CSP-based approaches to SAT, and shows that a version of FC, called nFC1, outperforms the basic DP procedure on the encoding in Definition 9.3.2. The correctness and completeness of nFC1 is proved in [BMFL02]. To state the result, we need the following terminology, that also explains the name nFC1:
denote by \( C^n_{cl} \) the set of constraints on a scheme \( s \) in which the current variable \( x_i \) and exactly one variable from \( U \) occur;

- denote by \( CP^n_{cl} \) the set of constraint projections on a scheme \( t \) in which the current variable \( x_i \) and exactly one variable from \( U \) occur.

- If \( x_i \) is the current variable passed to HAC, then perform hyper-arc consistency on the problem with domain \( D_i = a \), constraints in \( C' := C^n_{cl} \cup CP^n_{cl} \), and domains of variables, different from \( x_i \), that occur in the schemes of \( C' \). If an empty domain is generated, then return an empty domain set, else return the hyper-arc consistent domain set so generated.

Hence nFC1 is the resulting version of FC. The name nFC1 is motivated by the choice of constraints: \( n \) means that hyper-arc consistency is performed, and this involves constraints of any arity \( n \); instead, 1 refers to the fact that exactly one uninstantiated variable is chosen to enforce hyper-arc consistency.

**Theorem 9.3.4 ([BMFL02]).** If the given CSP is consistent, then nFC1 returns a consistent assignment for it; else it reports that the problem is inconsistent.

### 9.4 The KCSP Algorithm

In the remainder of this chapter, we focus on a procedure for satisfying modal formulas via constraint propagation and satisfaction, that is based on the semantic intuitions underlying the intermediate translation in Definition 9.3.2. We suggest a number of other similar procedures in Section 9.6 below.

#### 9.4.1 Examples

Before explaining the procedure KCSP in Subsection 9.4.2 below, we start by considering an example formula in CNF, its encodings as CSP and how the procedure KCSP works on it. Let us consider the following modal formula:

\[
\psi := \Box(p \lor q) \land \neg\Box(p \land q) \land p.
\]

After the minimum layer of the formula is computed, i.e., 0, the following CSP is returned — with propositional formulas as variables:

1. three propositional variables: \( \Box(p \lor q); \Box(p \land q); p; \)

2. the variable domains of \( \Box(p \land q) \) and \( p \) are set to \( \{1\} \); the variable domain of \( \Box(p \land q) \) is set to \( \{0\} \); 

3. no constraints.
9.4. The KCSP Algorithm

Then the CSP is passed to the CSP propositional solver that returns the only possible assignment \( \mu \) (unique for this formula; in general, split may be needed to choose among alternative assignments): \( \mu \) maps the three variables, in the order given above, to the triple \((1,1,0)\).

The modal procedure, invoked on \( \mu \), does the following: selects all formulas, within the scope of a \( \Box \) operator, and join them in a conjunction \( \Phi \):

\[
\Phi := p \lor q.
\]

This is the universal theory: in model-theoretic terms, this is the formula that is to be satisfied by each modal successor at level 1 of a state satisfying \( \Box(p \lor q) \) at level 0. Then, each formula that occurs in the scope of a negative occurrence of a \( \Box \) operator is negated, hence transformed in CNF: in this case, the result is a formula \( \Theta \) defined as

\[
\Theta := \neg p \lor \neg q.
\]

There may of course be multiple such existential theories \( \Theta \), which have to be satisfied at level 1, not necessarily at the same state. The conjunction \( \Phi \land \Theta \) is passed to the propositional procedure; in this case, the conjunctive formula that is passed on is

\[
(p \lor q) \land (\neg p \lor \neg q).
\]

The formula is translated into a new CSP and its consistency is checked; this results into two possible assignments, hence the procedure halts returning that the original formula is satisfiable.

9.4.2 Mapping Modal Formulas into CSPs

As the example in Subsection 9.4.1 illustrates, proposition letters and modally quantified formulas are both treated as propositional variables; namely variables with only two possible values, 0 or 1.

Consider a unimodal language \( \mathcal{ML} := \mathcal{ML}(P) \), and an \( \mathcal{ML} \) conjunction \( \phi \) of \( n \) disjunctions of the form

\[
\bigvee_{i}^{r_{i}} \Box \phi_{i}^{j} \lor \bigvee_{j}^{r_{i}'} \neg \Box \theta_{j}^{i} \lor \bigvee_{l}^{m_{i}} L_{i}^{l}
\]

for \( i = 1, \ldots, n \), where each \( L_{i}^{l} \) is either a proposition letter or its negation. The disjunction is then encoded as a constraint by treating all formulas of the form \( \Box \phi_{i}^{j} \) or \( \neg \Box \theta_{j}^{i} \) as propositional literals, and applying the encoding \( T^{\text{CSP}}_{\text{SAT}} \) as in Definition 9.3.2. Formally, we have the following definition.


**Definition 9.4.1.** Consider a modal formula $\phi$ in CNF. Suppose that $\phi$ is a conjunction of $n$ formulas such as (9.2). Let $L(\phi)$ be the set of distinct propositions that occur in the set of literals $L_i$ in $\phi$. Thus consider the propositional language whose set of proposition variables is

$$Prop := L(\phi) \cup \{ \Box \phi_i, \Diamond \theta_j : i = 1, \ldots, n \},$$

and consider $\phi$ as a proposition in this language; call it $\phi^{Prop}$. Then the CSP translation of the modal formula $\psi$ into CSP form is

$$CSP(\phi) := T^{CSP}_{SAT}(\phi^{Prop}).$$

Using Fact 9.3.3, we obtain an analogue of Theorem 9.2.5.

**Corollary 9.4.2.** Consider a modal language $\mathcal{ML}$ and a formula $\psi$ in CNF. The formula $\psi$ is satisfiable in an $\mathcal{ML}$ model iff there exists a truth-value assignment $\mu$ that satisfies $CSP(\psi)$, and such that the propositional formula $P(\mu)$ generated by $\mu$ is $\mathcal{ML}$ satisfiable.

**Proof.** Assume that $\phi$ is a conjunction of disjunctions (9.2), and that all tautological formulas such as $p \lor \neg p$ have been removed from $\phi$. Let $L(\phi)$ be the set of distinct propositions that occur in the set of literals $L_i$ in $\phi$. Denote by $\mathcal{P}$ the propositional language whose letters are the $L(\phi)$ propositions and all the $\phi$ disjuncts $\lor \psi^i_j$ and $\lor \theta^i_j$.

Suppose that $\mu$ is an assignment that satisfies $CSP(\phi)$. Then it satisfies $\phi$ as a $\mathcal{P}$ formula, by Fact 9.3.3. Assume that $P(\mu)$ is satisfiable in an $\mathcal{ML}$ model $\mathcal{M}$ and world $w$. This implies that, if $\mu$ assigns 1 to a $\mathcal{P}$ letter in $\phi$, then this holds true at $w$ in $\mathcal{M}$; else it holds false. If $\mu$ does not assign any value to an $\mathcal{ML}$ proposition letter $p$, then we extend $\mu$ to $p$ by means of the valuation in $\mathcal{M}$. Thus $\mathcal{M}$ satisfies $\phi$ at $w$.

Vice versa, suppose that $\mathcal{M}, w \models \phi$. Then define the $\mathcal{P}$ assignment $\mu$ as

$$\mu(\phi^i) = 1 \text{ iff } \mathcal{M}, w \models \phi^i,$$

for each proposition letter $\phi^i$ of $\mathcal{P}$. It is not difficult to prove by structural induction that $\mu$ makes $\phi$ true as a $\mathcal{P}$ formula and that $\mathcal{M}, w \models P(\mu)$. The result now follows from Fact 9.3.3. $\square$


**9.4.3 Mapping Modal Inferences into CSP Inferences**

In Algorithm 9.4.1 below, FC returns an assignment for the input CSP. During search, an empty assignment (false) is generated iff the current CSP is detected to
be inconsistent; i.e., the corresponding formula is unsatisfiable. Then backtracking takes place and another value for the last instantiated variable is checked etc. If no assignment can be found, the FC algorithm concludes that the input CSP is unsatisfiable. Thus the propositional search space is explored by FC, interleaving backtracking with hyper-arc consistency, in a depth-first manner.

The subprocedure $KFC$ has to handle modal satisfiability: more precisely, $KFC$ determines modal satisfiability by calling constraint satisfaction procedures over Boolean CSPs.

We only sketch a proof of the correctness and completeness of Algorithm 9.4.1.

**Theorem 9.4.3.** The $KCSP$ procedure returns a non-empty assignment iff the input modal formula is satisfiable.

**Proof.** The procedure $BoolCSP$ transforms the given formula into as CSP as in Definition 9.4.1. This is proved consistent by FC iff the formula is satisfiable as a propositional formula, see Theorem 9.3.4 and Fact 9.3.3. Our theorem now follows from Corollary 9.4.2. □

### Algorithm 9.4.1: KCSP

```plaintext
procedure BoolCSP(ψ)  
return BoolFC(CSP(ψ));

procedure BoolFC(CSP(ψ))  
μ := FC(CSP(ψ));  
if μ = false then return false; % backtrack  
else return KFC(μ);      

procedure KFC(μ)  
Φ := \{ϕ : □ϕ is assigned 1 in μ\};  
for each □θ in μ that is assigned 0 do  
θ := CNF(¬θ);  
if not BoolCSP(θ ∧ Φ) then return false; % backtrack  
return true;
```

Notice that the $KCSP$ algorithm interleaves steps in which modal information is "hidden" — within the scope of modal operators — so as to get a propositional problem, with steps in which modal information is "unpacked" — i.e., the $KFC$ subprocedure is called on the modally quantified formulas.
Chapter 9. Diamonds and Constraints

9.5 Experimental Assessment

This section contains a brief experimental discussion of KCSP. In order to test our preliminary procedure, we considered two test sets:

- manually coded formulas: these were devised following the criteria proposed in [HS96] for the creation of benchmark formulas;
- several formulas from the problem sets proposed by Heuerding and Schwendimann in [HS96], which were used in, for example, Tableaux'98 — see also p. 155, where this set is used to compare the output of the layered translation to the one of the standard translation.

The KCSP algorithm was implemented in ECL’PS*, version 5.4, by Sebastian Brand; the implementation is called mc.pl. A translator from the [HS96] format into the format of mc.pl was provided by Juan Heguiabehere. We ran our experiments on an AMD Athlon Processor (1.1 GHz), with 512 MB RAM, under Red Hat Linux 7.1.

The program mc.pl returns a full search tree for the input formula. This is not an efficient choice, and in future experiments we shall also take this feature into account. Yet, this choice gave us a better idea of the behaviour of the algorithm, and this is our major concern at this stage of the work. Several formulas used in the experiments are available at http://www.cwi.nl/~gennari/thesis/kcsp.html.

By running the tests in [HS96], we noticed a clear difference between the behaviour of mc.pl on unsatisfiable and satisfiable formulas. So our discussion is divided in two subsections as below: results with unsatisfiable formulas; experiments with satisfiable formulas.

The current implementation of KCSP is still a prototype, and it cannot compete with highly optimised theorem provers for modal logics as those discussed in Section 8.1. In the final Subsection 9.5.3, we elaborate on this issue and some possible improvements to the basic procedure KCSP, that are triggered by the experimental work presented as below.

9.5.1 The Unsatisfiable Case

Our experiments with manually coded formulas were rather promising: we passed to mc.pl a series of modal formulas with at most 18 distinct proposition letters, modal depth between 0 and 3, and at most 18 disjuncts; no redundant propositional tautologies occur in those formulas. On each instance, the program answered correctly within 0.36 seconds; a number of these formulas, and the related search trees explored by mc.pl are on the aforementioned web page.

Once we were confident that our algorithm consistently displayed a good behaviour on manually coded formulas, we considered a number of K theorems, as provided in [HS96]: there, theorems are partitioned into sets, and formulas
in a set usually differ in the number of propositional letters (V) and the modal depth (D). To run our experiments, first those K theorems were negated, then the obtained unsatisfiable formulas were transformed in CNF — see Definition 8.2.1.

Figure 9.1 below displays some of the tests that we ran: each label on the horizontal axis corresponds to a different test set; thus the results for the first two formulas from the considered test set are compared in terms of CPU seconds, as displayed along the vertical axis.

![Test formulas from [HS96]](image)

Figure 9.1: Comparison of the first two formulas from test sets in [HS96].

Table 9.1, displayed below, reports more formulas than those compared in Figure 9.1; the parameter C stands for the maximum number of distinct atoms (i.e., variables) in clauses (i.e., constraints) of the tested CNF formula. As in Section 8.5, we gave the theorem prover a time out of an hour; thus the abbreviation NA means that we did not obtain an answer within this time bound. As it is clear from Figure 9.1 and Table 9.1, tests become harder for the KCSP algorithm when C ≥ 3, and D ≥ 4 or V ≥ 4; in fact, such values can result in more calls to the KCSP procedure, and more backtracking points.

### 9.5.2 The Satisfiable Case

The best performances of the KCSP algorithm was achieved in the unsatisfiable case, and the algorithm does not seem to be efficient in the satisfiable case yet. We believe that this behaviour of the algorithm does not depend on its current implementation, but on the design of the algorithm itself. For instance, the KCSP algorithm does not fully exploit the fact that the input formulas are disjunctions;
Chapter 9. Diamonds and Constraints

<table>
<thead>
<tr>
<th>Test formula</th>
<th>V</th>
<th>D</th>
<th>C</th>
<th>CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_branch_p1</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>0.03</td>
</tr>
<tr>
<td>k_branch_p2</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>32.15</td>
</tr>
<tr>
<td>k_d4_p1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>k_d4_p2</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1.03</td>
</tr>
<tr>
<td>k_dum_p1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>8.55</td>
</tr>
<tr>
<td>k_dum_p2</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>NA</td>
</tr>
<tr>
<td>k_lin_p1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0.11</td>
</tr>
<tr>
<td>k_lin_p2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>0.47</td>
</tr>
<tr>
<td>k_path_p1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>k_path_p2</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>0.47</td>
</tr>
<tr>
<td>k_ph_p1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>k_ph_p2</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>NA</td>
</tr>
<tr>
<td>k_poly_p1</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>0.01</td>
</tr>
<tr>
<td>k_poly_p2</td>
<td>15</td>
<td>6</td>
<td>2</td>
<td>0.02</td>
</tr>
<tr>
<td>k_t4_p1</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>15.54</td>
</tr>
<tr>
<td>k_t4_p2</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>165.69</td>
</tr>
</tbody>
</table>

Table 9.1: Test $K$ theorems from [HS96]

i.e., it tries to return a total assignment, even when a partial assignment to its variables could be sufficient for determining satisfiability. As for this, let us consider a conjunction of $2n \geq 2$ formulas of the form

\[ p \lor q_1^i \lor \cdots \lor q_n^i, \]

\[ p \lor \neg q_1^i \lor \cdots \lor \neg q_n^i, \]

with $1 \leq i \leq n$. Such a formula is clearly satisfied by assigning 1 to $p$, and any other value to the remaining propositional letters. Still the Boolean forward checking procedure in $KCSP$ will search for a consistent total assignment, and uselessly enforce hyper-arc consistency. The situation becomes even worse when each $q_j^i$ is substituted by a modally quantified formula; in fact this results in useless calls to the $KFC$ modal procedure.

9.5.3 Finale

As remarked in Subsection 9.5.2 above, the $KCSP$ algorithm still explores fruitless branches of the search tree. Such useless explorations depend on the basic $FC$ algorithm schema: this is a complete solver that returns a total assignment to the input problem. In contrast, the $DP$ procedure does not suffer from this drawback, thanks to unit substitution: unit substitution removes, from the search tree, those disjunctions in which the current variable that is assigned the value either true or false occurs positively or negatively, respectively. Therefore, a natural
optimisation of the basic \( KCSP \) algorithm will consist in embedding some form of unit substitution into \( KCSP \); then it will be meaningful and interesting to have both a theoretical and an experimental comparison of such a variation of the basic \( KCSP \) procedure and the \( K\)-\textsc{sat} algorithm.

In the following section, we propose other possible variations of the basic \( KCSP \) module: there, either hyper-arc consistency (see Subsection 9.6.1) or forward checking (see Subsection 9.6.2) are replaced by different constraint based algorithms.

### 9.6 Variations of \( KCSP \)

In this section, we briefly propose two other possible approaches to determining modal satisfiability via constraint propagation and solving algorithms.

#### 9.6.1 Boolean CSPs and Constraint Propagation

There is at least one other main reformulation of a SAT problem as a CSP. This makes use of so-called Boolean constraints. These are represented implicitly as Boolean formulas and equality constraints of the following forms, where \( x, y \) and \( z \) stand in for generic CSP variables:

\[
\begin{align*}
  x &= y \quad \text{is an } EQ \text{ constraint;} \\
  \neg x &= y \quad \text{is a } NOT \text{ constraint;} \\
  x \land y &= z \quad \text{is an } AND \text{ constraint;} \\
  x \lor y &= z \quad \text{is an } OR \text{ constraint.}
\end{align*}
\]

The equality symbol in the constraints above is interpreted as \( \leftrightarrow \), the logical connective of bi-implication. We can now provide a formal definition for Boolean CSPs with the above types of constraints.

**Definition 9.6.1.** A Boolean CSP \( P := \langle X, D, C \rangle \) is a \textsc{bool} CSP if its constraints are of the form (EQ), (NOT), (AND) or (OR) as above, that we call \textsc{bool} constraints.

To reason about such constraints, rules such as the following one are employed:

\[
x \lor y = z, \ x = 0, \ z = 1 \quad \vdash \quad y = 1,
\]

the intuitive reading of which is: if \( x \) or \( y \) have the same truth values as \( z \), \( x \) is false and \( z \) is true, then \( y \) must be true. A set of independent rules, called \textsc{bool}, is provided in [Apt00b]. As the author proves, applying the set of rules \textsc{bool} is “equivalent” to performing unit propagation. To formulate this equivalence, first
a mapping of $BOOL$ constraints into clause sets is provided (by interpreting the equality symbol $=$ in the $BOOL$ constraints as the logical connective $\leftrightarrow$); hence by providing the opposite translation. Thus the author can prove, through those translations, that unit propagation and the $BOOL$ rule system can simulate each others in constant time, but “the simulation of the unit propagation by means of the Boolean constraint propagation [i.e., the $BOOL$ rules] leads to a generation of redundant constraints”, due to the introduction of redundant variables. However, what is more interesting to us is the following equivalence.

**Theorem 9.6.2 ([Apt00b]).** A CSP, without empty domains, is hyper-arc consistent iff it is closed under the applications of the rules of the $BOOL$ system.

Hence, the $BOOL$ system could be used instead of hyper-arc consistency in $KCSP$ to enforce modal consistency.

### 9.6.2 When a Bit of Cross-eye Helps

A third interesting approach is that of incorporating look-back enhancements of BT, like *Conflict Direct Backjumping* (CDB), into the basic DP procedure; see [BS97]. Look-back algorithms go in the opposite direction with respect to forward checking: more precisely stated, look-back algorithms exploit information about search that has already taken place, i.e., the set of the assigned variables $A$ in the terminology of FC. The advantage of an algorithm like DP with CDB with respect to FC is that, like DP, the first does not need to return a total assignment. In fact, unit propagation is present in this improved version of DP; it is implemented by maintaining a pointer to the constraint, in the input CSP, that causes the exclusion of a specific assignment. We refer the reader to [BS97] for an accurate description of it and the related experimental work.

Given that also this improvement of DP is complete and correct, it could be interesting to see how it could improve the basic $K$-SAT algorithm for modal logics. Not only could this be worth more exploration, but also a combination of look-ahead techniques, like FC, and look-back techniques, like CDB, already developed in the CSP community: in fact, as the experimental work in [BS97] highlights, “the dramatic performance improvements resulting from the incorporation of look-back is in fact due to a synergy between the look-ahead techniques and look-back techniques applied”.

### 9.7 Conclusions

#### 9.7.1 Synopsis

In Chapter 8 we partially encode the tree model property in a translation from modal to first-order logics; as shown in that chapter, this does pay off, and it
allows us to reuse existing, sophisticated theorem provers for modal logics, without modifying them.

In the present chapter, the tree model property is still behind a series of procedures for modal logics. In fact, all these “reason” propositionally, layer by layer, on the input modal formula. They differentiate according to the basic propositional solver adopted.

Again, we try to reuse the “existing technology” for CSPs, and make it work for modal reasoning: this time, the encoding is not into first-order formulas but into sequences of CSPs, so to speak. These are passed to a propositional solver, that is slightly modified so as to “open” boxes and diamonds, and enforce satisfiability within them too, in a top-down manner.

9.7.2 Discussion

As shown in the first part of this thesis, a number of constraint techniques have been developed in the literature for tackling constraint satisfiability and, in particular, propositional satisfiability in an efficient manner. This chapter constitutes a first attempt to use constraint satisfaction algorithms for propositional satisfiability in the field of automated theorem proving for modal logics. Besides, we spend an entire section, namely Section 9.6 above, on possible variations of the solving schema proposed in this chapter, and possible optimisations to the latter.

It would also be interesting to study whether soft constraint propagation and solving algorithms could be used for testing a limited form of modal satisfiability. In some cases, we would like to set preferences on properties of the system: so that, even when not all properties can be satisfied, an optimal solution, with respect to our preferences, could still be returned. This could also constitute an interesting test-bed for soft constraint solving and propagation algorithms.