Symbolic Model Checking with Sets of States Represented as Formulas

Per Bjesse
Chalmers University of Technology, Sweden
bjesse@cs.chalmers.se

Abstract. We analyse traditional symbolic model checking [BCL$^+$94] to isolate the operations that a representation for sets of states need to support and present a different view, where sets of states are represented by propositional logic formulas. High level algorithms for symbolic model checking based on Stålmarck's method are then presented, which results in a model checking procedure closely related to bounded model checking [BCC$^+$]. Our method lifts the requirement on finding bounds and allows intermediate minimisations of state set representations. We also sketch how to generalise the approach to stronger logics.

1 Introduction

Propositional temporal logic theorem proving is PSPACE-complete, and often infeasible for industrial-size systems. Model checking is a reduction of this complex problem to a possibly exponential number of subproblems that hopefully have better characteristics.

Different model checking approaches achieve success with different classes of systems; the suitability of a particular method is governed by the tradeoff between space complexity and the time complexity of the operations. One of the most important determining factors for the time/space ratio is the choice of set representation.

We will present a new view of states and sets of states where the verification problem is reduced to formula minimisation and tautology checking on a sequence of propositional logic formulas. This approach has the advantage that the representation can be made very compact; some of the operations are on the other hand complex since tautology checking is coNP-complete. The time/space ratio will therefore be very different from that of many traditional model checking methods.

The logical view of set representation has the benefit of allowing generalisations to richer logics, which for instance makes direct model checking of infinite state spaces possible. Our purpose in this paper is to present a different angle on model checking, introduce the new family of representations, and demonstrate some of the possibilities.

The presentation is organised as follows: In section 3 and 4 we analyse CTL model checking and extract requirements on representations for sets of states,

section 5 introduces the idea of encoding sets of states as propositional logic formulas and relates it to BDDs. Section 6 presents algorithms that use Stålmarck's method to implement the ideas in practice, and relates the approach to bounded model checking. Section 7 finally sketches how to extend the ideas to two other logics and the paper is concluded by some remarks about time/space tradeoffs.

2 Conventions

Infinite composition of a function f is denoted f^ω, where the value of $f^\omega(x)$ is equal to $f^k(x)$ if there is a k such that $f^k(x) = f^{k+1}(x)$ and otherwise is undefined. The notation $g : x \longmapsto e(x)$ means that g is a function that maps every x to $e(x)$.

We presume basic knowledge of first order logic (FOL) and propositional logic (PROP). The propositional constants (true,false) are denoted $(\top, \bot)$ and substitution of e for every occurrence of p in ϕ is written as $\phi[e/p]$. The size of a formula ϕ, $|\phi|$, is the sum of the number of variable occurrences and the number of connectives in ϕ ($\top$ and $\bot$ therefore have size 0).

We also make some use of an extension of propositional logic called quantified boolean formulas, QBF. This logic is propositional logic enriched with quantifiers that range over propositions, with the semantics of quantifiers defined by

$$\exists p.\phi \equiv \phi[\top/p] \vee \phi[\bot/p]$$
$$\forall p.\phi \equiv \phi[\top/p] \wedge \phi[\bot/p]$$

These identities make it possible to map any QBF formula to an equivalent propositional logic formula.

3 CTL Model checking

Model checking is the process of determining the set of states in a Kripke structure that satisfies a temporal logic formula. For simplicity, our investigations are based on computational tree logic (CTL) model checking by fixpoint computations.

3.1 Syntax and semantics

The language of CTL ($\mathfrak{L}_{CTL}$) is freely generated from a a finite set of atomic state propositions, AP, according to the following rules:

$x \in \mathfrak{L}_{CTL}$	if $x \in AP$
$\neg x \in \mathfrak{L}_{CTL}$	if x $\in \mathfrak{L}_{CTL}$
$x \# y \in \mathfrak{L}_{CTL}$	if # is one of the connectives $\vee$, $\wedge$ or $\rightarrow$ and $x, y \in \mathfrak{L}_{CTL}$
$\Pi\, x \in \mathfrak{L}_{CTL}$	if Π is one of the operators AX, EX, AG or EG and $x \in \mathfrak{L}_{CTL}$
$\Pi[x, y] \in \mathfrak{L}_{CTL}$	if Π is one of the operators AU or EU and $x, y \in \mathfrak{L}_{CTL}$

The semantics of temporal formulas are defined with respect to a Kripke structure that contains a set of states $State$, a mapping $Label$ that takes a state to the set of atomic propositions that label the state and a transition relation $Trans \subseteq State \times State$.

That a state s_0 in a Kripke structure M satisfies the temporal formula γ is henceforth written as $M, s_0 \models \gamma$. An atomic proposition p is true in a state exactly when it is in the labelling set of the state; the semantics of compound formulas are defined recursively:

$M, s_0 \models p$	iff $p \in AP$ and $p \in Label(s_0)$
$M, s_0 \models \neg\phi$	iff not $M, s_0 \models \phi$
$M, s_0 \models \phi \wedge \psi$	iff $M, s_0 \models \phi$ and $M, s_0 \models \psi$
$M, s_0 \models AX\,\phi$	iff for all states t such that $Trans(s_0, t)$ $M, t \models \phi$
$M, s_0 \models EX\,\phi$	iff for some state t such that $Trans(s_0, t)$ $M, t \models \phi$
$M, s_0 \models AU[\phi, \psi]$	iff for all paths $(s_0, s_1, ...)$,
	$\exists i \geq 0.\ M, s_i \models \psi\ \wedge\ \forall j.(0 \leq j < i \rightarrow\ M, s_i \models \phi)$
$M, s_0 \models EU[\phi, \psi]$	iff for some path $(s_0, s_1, ...)$,
	$\exists i \geq 0.\ M, s_i \models \psi\ \wedge\ \forall j.(0 \leq j < i \rightarrow\ M, s_i \models \phi)$

Disjunction, implication and the remaining temporal operators are regarded as abbreviations:

$\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$	
$\phi \rightarrow \psi \equiv \neg(\phi \wedge \neg\psi)$	
$AF\,\phi \equiv AU[\top, \phi]$	"ϕ is true in the future along all paths from s_0"
$EF\,\phi \equiv EU[\top, \phi]$	"ϕ is true in the future along some path from s_0"
$AG\,\phi \equiv \neg EF(\neg\phi)$	"ϕ holds in all future states along all paths from s_0"
$EG\,\phi \equiv \neg AF(\neg\phi)$	"ϕ holds in all future states along some path from s_0"

3.2 Computing models of CTL formulas

We are interested in determining the set of models $\overline{\phi}$ of a temporal formula ϕ relative to a particular Kripke structure M:

$$\overline{\phi} = \{s\ : M, s \models \phi\}$$

The set of models for an atomic proposition is directly computable; the connectives correspond to set operations

$$\overline{\neg\phi} = State \setminus \overline{\phi}$$
$$\overline{\phi \vee \psi} = \overline{\phi} \cup \overline{\psi}$$
$$\overline{\phi \wedge \psi} = \overline{\phi} \cap \overline{\psi}$$
$$\overline{\phi \rightarrow \psi} = (State \setminus \overline{\phi}) \cup \overline{\psi}$$

Given the models for a formula ϕ, the set of models for $AX\,\phi$ and $EX\,\phi$ can be constructed by inspecting neighbouring states

$$\overline{EX\,\phi} = \{x : \exists s.Trans(x,s) \wedge s \in \overline{\phi}\}$$
$$\overline{AX\,\phi} = \{x : \forall s.Trans(x,s) \rightarrow s \in \overline{\phi}\}$$

All of the remaining temporal operators have well-known fixpoint characterisations [EC80]:

$$\overline{AF\,\gamma} = \mu Y.\gamma \vee AX\ Y$$
$$\overline{EF\,\gamma} = \mu Y.\gamma \vee EX\ Y$$
$$\overline{AG\,\gamma} = \nu Y.\gamma \wedge AX\ Y$$
$$\overline{EG\,\gamma} = \nu Y.\gamma \wedge EX\ Y$$
$$\overline{AU[\gamma_1\,,\gamma_2]} = \mu Y.\gamma_2 \vee (\gamma_1 \wedge AX\ Y)$$
$$\overline{EU[\gamma_1\,,\gamma_2]} = \mu Y.\gamma_2 \vee (\gamma_1 \wedge EX\ Y)$$

The fixpoint operators $\mu Y.\rho(Y)$ and $\nu Y.\rho(Y)$ takes a *temporal transition formula* $\rho(Y)$ to a set of states. A temporal transition formula can be thought of as a temporal formula schema, i.e. a formula that is parameterised with a variable ranging over other formulas. The denotation of a temporal transition formula is hence a set transformer: If Y is the only free variable in $\rho(Y)$, then

$$\overline{\rho} : \overline{\phi} \longmapsto \overline{\rho[\phi/Y]}$$

The fixpoint operators themselves are implicitly defined by

$\mu Y.\rho(Y) = \overline{\phi}$ iff ϕ is the least set satisfying the equation $\overline{\rho}(\overline{\phi}) = \overline{\phi}$

$\nu Y.\rho(Y) = \overline{\phi}$ iff ϕ is the largest set satisfying the equation $\overline{\rho}(\overline{\phi}) = \overline{\phi}$

All temporal transition formulas in the fixpoint characterisations denote monotone set transformers [EC80]; the equations are therefore well-defined.

Symbolic model checking The basic algorithms for finding models for CTL formulas are uncomplicated; the tricky part is to make them scale up to industrial size systems. A major obstacle is that the algorithms will require space linear in the number of states if sets are represented by a straightforward enumeration. This means that systems with several hundreds of variables will have a slight chance of being verifiable. Symbolic model checking tries to get around this problem by instead representing sets by a necessary and sufficient criterion for being a member of the set. The choice of criterion and how it is encoded has great impact on the class of verifiable systems.

4 Operations and requirements on a representation for sets of states

Primitive model checking operations A representation for sets of states must support all the operations that are needed to compute the set of models of

arbitrary CTL formulas. The description of CTL model checking indicates that the following operations, that we refer to as *primitive model checking operations*, are necessary:

1. Atomic proposition model-finding. $m_{atom}(p) = \overline{p}$ for $p \in AP$
2. Compound formula model construction
   - Unary connective. $m_\neg : \overline{\phi} \longmapsto \overline{\neg\phi}$
   - Binary connectives. $m_\# : (\overline{\phi}, \overline{\psi}) \longmapsto \overline{\phi \# \psi}$ for $\#$ arbitrary propositional connective
3. Transitions operations
   - $m_{EX} : \overline{\phi} \longmapsto \overline{EX\ \phi}$
   - $m_{AX} : \overline{\phi} \longmapsto \overline{AX\ \phi}$
4. Fixpoint operations. m_μ, m_ν

Example 1 *Given that* msg *and* safe *are atomic propositions, the fixpoint characterisation of AG implies that the set of models of the temporal formula* $\text{msg} \rightarrow AG(\text{safe})$ *is computed by*

$$m_\rightarrow(m_{\text{atom}}(\text{msg}), m_\nu(\lambda x.m_\wedge(m_{\text{atom}}(\text{safe}), m_{AX}(x))))$$

Efficiency requirements A representation for sets of states must also satisfy a number of other requirements in order to make model checking computationally feasible.

Let us use the term "suitable" to mean "has a representation that should not be too expensive to compute nor be too large for the sets of states that arise". Assuming $p \in AP$, it is clear that a representation for sets of states must have the following characteristics relative to a particular verification problem:

1. $m_{atom}(p)$ is suitable
2. $m_\neg, m_\#, m_{AX}$ and m_{EX} preserve suitability
3. Set equality can be decided efficiently to allow fixpoint stabilisation checking.
4. Set membership can be decided efficiently.

Different representations therefore have different suitability for each problem.

5 Encodings states and sets of states in propositional logic

As the Kripke model M is only labelled with finitely many propositions, we can enumerate AP as $\{p_i : i < |AP|\}$ and in each state define

$$val_n^s = \begin{cases} p_n & \text{if } M, s \models p_n \\ \neg p_n & \text{otherwise} \end{cases}$$

We presume that each state is uniquely labelled (it is simple to modify the encodings to suit structures where this is not the case). This conveniently allows

us to identify each state s with the set of literals $\{val_n^s : n < |AP|\}$; sets of states S are represented by a propositional logic formula ϕ that has AP as propositions and fulfils that

$$s \models \phi \quad \text{iff} \quad s \in S \tag{1}$$

Set equality and set membership testing corresponds to tautology checking.

We assume a syntactic set transforming operator $next(S)$ that takes a set of propositional logic formulas and "primes" every propositional variable. Binary relations on states $R \subseteq State \times State$ are defined by propositional logic formulas σ for which it holds that their propositions come from $AP \cup next(AP)$ and that

$$s_1 \cup next(s_2) \models \sigma \quad \text{iff} \quad R(s_1, s_2) \tag{2}$$

When we say that a set S or relation R is "defined by ρ", we mean that ρ is a formula that fulfils equation 1 or 2.

Given that the Kripke model M has a transition relation defined by τ, the primitive model checking operations for a propositional logic set representation have very simple definitions:

$$\begin{gathered} m_{atom}(p) = p \\ m_{\neg}(\phi) = \neg\phi \\ m_{\#}(\phi, \psi) = \phi \# \psi \\ m_{EX}(\phi) = \exists t.\tau(s,t) \wedge \phi(t) \\ m_{AX}(\phi) = \forall t.\tau(s,t) \rightarrow \phi(t) \\ m_{\mu}(f) = f^{\omega}(\bot) \\ m_{\nu}(f) = f^{\omega}(\top) \end{gathered}$$

The next state operations m_{AX} and m_{EX} need to compute propositional logic formulas Θ that is the result of quantifying over a state s in a defining formula. This translates to nested boolean quantifications over the atomic propositions $p_0, p_1 \ldots$ in AP

$$\begin{aligned} \exists s.\phi(s) &\equiv \exists p_0.\exists p_1 \ldots \exists p_n.\phi \\ \forall s.\phi(s) &\equiv \forall p_0.\forall p_1 \ldots \forall p_n.\phi \end{aligned}$$

These QBF formulas can in turn be mapped to pure propositional logic formulas as described in section 2. A quantification of a state variable is therefore equivalent to $|AP|$ boolean quantifier eliminations on the defining formulas.

Both the fixpoint operations m_μ and m_ν take a formula transformer f as an argument, and use it to compute the fixpoint $f^\omega(\rho)$ of a sequence of formulas ϕ_n with $\phi_0 = \rho$ and $\phi_{n+1} = f(\phi_n)$. The equality under consideration here is logical equivalence, which means that

$$f^{\omega}(\rho) = \phi_k \quad \text{iff} \quad k \text{ is the smallest number such that } \models \phi_{k+1} \leftrightarrow \phi_k$$

All sets that are generated during the model checking can therefore be encoded as formulas. There are two essentially different ways of representing sets symbolically, both bearing simple relations to the defining formulas. Each of these representations have special strengths and weaknesses.

Semantic representation A simple representation for the defined sets is to encode the truth table of the propositional logic formulas. The encoding must be clever as the size of a truth table is exponential in the number of atomic propositions; representation size can therefore be lowered significantly by opting for a decision graph rather than a table.

Binary decision diagrams(BDDs) [Bry86] are examples of such an encoding that also demand canonicity in order to turn equivalence checking of represented formulas into an $O(1)$ operation. The canonicity requirement consists of fixing an ordering of the variables; which particular ordering that is chosen has a large impact on the representation size.

BDDs have many good characteristics, and often allow model checking of state spaces that are surprisingly large. A design with 100 boolean variables (2^{100} states) has a good chance of being within the reach of the technology. However, it is not hard to find problems where the BDD representations grow too large even for relatively small circuits; for example, multipliers of greater width than 16 bits seem impossible to verify monolithically with BDDs.

Direct formula representation Canonicity makes equivalence checking cheap, but can cause representations to explode. For certain systems non-canonical representations are therefore preferable.

An obvious, non-canonical representation for the sets defined by propositional logic formulas are the formulas themselves. Basing the representation on the actual formulas means that we choose a syntactic theorem-proving inspired approach to set encoding. There are many possible benefits to this view:

- Many sets with enormous BDD representations for any variable ordering have tractable formula representations
- Variable ordering is unnecessary
- The number of variables is less critical than for BDDs as tautology checkers routinely handle formulas with thousands of variables
- We can draw from the experience of the propositional logic theorem proving community

The naive implementations of the primitive model checking operations will however be space inefficient as

$$|m_{\#}(\phi, \psi)| \approx |\phi| + |\psi|$$
$$|m_{AX,EX}(\phi)| \approx |\phi| \cdot 2^{|AP|}$$

Special algorithms that select sufficiently small formula representatives are clearly needed.

It is also necessary to be able to decide set equality efficiently; any set can be represented by infinitely many equivalent formulas which means that set equality checking must be done by propositional logic theorem proving. This is a coNP complete problem, so it is unlikely that equality can be efficiently decided for any two sets. However, the choice of theorem prover and the particular formulas will govern the tractability of each case.

To summarise, we will at a bare minimum need the following operations for the implementation of practical propositional-logic based model checking:

- A tautology checker that is efficient for many of the formulas that arise in practice.
- A mechanism for minimising formulas.

6 Stålmarck's method for symbolic model checking

Stålmarck's method [Stå89] is a proof procedure for propositional logic, that has been industrially successful. It is remarkably proficient at tautology checking for many very large formulas that arise in practice.

We will in this section present how Stålmarck's method can be used to implement the necessary operations on the formula representation efficiently. Only the parts of the proof system that are necessary for understanding the algorithms will be presented; readers interested in further information are referred to Sheeran and Stålmarck's tutorial [SS98].

6.1 Formula representation

Stålmarck's method represents a propositional logic formula as a set of *triples* and a set of *literal identifiers*. Each triple $i_n : r_1 \# r_2$ is formed from an identifier, a connective and two references; each literal identifier $i_n : l$ is formed from an identifier and a negated or unnegated formula variable. References are negated or unnegated identifiers.

The triple set that represents a given formula can be seen as the abstract syntax tree (AST) of the formula, with an identifier allocated to each internal node and formula literal; each subformula is hence denoted by a unique identifier. The AST root i_{top} is called the *top identifier*.

Given that we denote the set of triples that correspond to a given formula $Triple$ and the set of literals Lit, the set of entities under considerations in the proof is $Entity = Triple \cup Lit \cup Bool$ (special use is made of the two propositional constants $\bot, \top$). Each entity corresponds to a truth constant or a subformula. The size ordering on subformulas induce an ordering on entities; this allows definition of $min(e_1, e_2)$ and $max(e_1, e_2)$.

6.2 Equivalence checking

At any stage of a proof, each $e \in \mathit{Entity}$ is in a unique equivalence class $[e]$ which can be considered to be the set of entities that at the moment are assumed to have the same truth value as e. We refer to the equivalence classes and the set of entities under consideration as the *system*. A singular system is a system where $[e] = \{e\}$ for all entities e; the representing system for ϕ, $\mathit{repr}(\phi)$, is the singular system that results from triplifying ϕ according to some reasonable strategy.

All proofs proceed by refutation; the proof procedure tries to derive a contradiction from an representing system which has been augmented by merging $[i_{top}]$ and $[\bot]$.

Example 1 *The system that is used to prove the formula* $\neg a \wedge b \rightarrow b \vee c$ *contains the set of entities*

$$\{i_1 : \neg a,\ i_2 : b,\ i_3 : c,\ i_4 : i_1 \wedge i_2,\ i_5 : i_2 \vee i_3,\ i_6 : i_4 \rightarrow i_5,\ \top,\ \bot\}$$

and the following equivalence classes:

$$\{\{i_1\},\ \{i_2\},\ \{i_3\},\ \{i_4\},\ \{i_5\},\ \{i_6, \bot\},\ \{\top\}\}$$

Inference rules are applied to the given system in order to propagate semantic information; every successful rule application merges at least two equivalence classes. A proof has been reached when $[\bot] = [\top]$; a system that satisfies this condition is called *explicitly contradictory*.

The inference rules comes in two varieties:

- Simple rules, that uses information on the truth value of i_i, i_j or i_k for a triple $i_i : i_j \ \# \ i_k$ to derive new information according to the semantics of the connective $\#$. As an example, if $[i_j] = [\top]$ and $\#$ is the disjunction operator, one of the simple rules will merge $[i_i]$ and $[\top]$.
- The Dilemma rule, that case splits over the truth value of a triple and adds the information that was derivable under both the assumption that the triple was false and under the assumption that the triple was true to the original system.

Exhaustive application of simple rules to a set of triples until no new equivalence classes can be merged is called *0-saturation*. This operation is linear in the number of entities, and is therefore very cheap.

1-saturation is the process of in turn applying the Dilemma rule to each entity in a set with an internal 0-saturation in each branch. Only one assumption has been made at all times; hence the name 1-saturation. Analogously (n+1)-saturation can be defined in terms of n-saturation.

Every tautology can be shown to have a degree of hardness, a minimum saturation level that is needed to arrive at an explicitly contradictory system from the augmented representing system. The complexity of n-saturation is exponential in n. However, a very large class of formulas that arise in formal verification turn out to have a hardness degree of 0 or 1. The power of Stålmarck's method is that the proof search is done in such a way that a proof for a formula with a low degree of hardness is found quickly.

6.3 Formula minimisation

So, Stålmarck's method is in practice very efficient when it comes to finding proofs for large formulas. But can the proof procedure also do formula minimisation?

A completed k-saturation without any initial assumptions on the truth value of i_{top} results in a system where the equivalence classes contain entities that must have the same truth value in all interpretations that satisfies the formula. This information can under certain conditions be used to simplify the system by removing entities.

We define two entities e_1 and e_2 in a system sys to be k-discoverable as equivalent, denoted $equiv(k, sys, e_1, e_2)$, if k-saturation without assumptions results in a system that not is explicitly contradictory and that also fulfils that $\{e1, e2\} \subseteq eq$ for some equivalence class eq. We also assume the existence of an operation $subst(sys, e_1, e_2)$ that overwrites all the systems references to $max(e_1, e_2)$ with references to $min(e_1, e_2)$, removes unconnected triples and literals and returns the corresponding singular system.

Theorem 1 *Given that sys is a representing system for a formula ϕ and that the entities e_1 and e_2 correspond to subformulas s_1 and s_2, the system $subst(sys, e_1, e_2)$ is equivalent to sys if one of the following conditions hold:*

- *e_1 and e_2 are* propagation equivalent*, which means that $equiv(0, sys, e_1, e_2)$.*
- *e_1 and e_2 are* k-taut equivalent*, which means that both $equiv(k, repr(\phi), e_1, e_2)$ and $equiv(k, repr(\neg\phi), e_1, e_2)$.*
- *e_1 and e_2 are* k-context equivalent*, which means that if s_1 and s_2 are the subformulas corresponding to e_1 and e_2, it holds that $equiv(k, sys, e_1, e_2)$ and $\models subst(sys, e_1, e_2) \rightarrow (s_1 \leftrightarrow s_2)$.*

Proof. Assume that $M \models sys$ for some interpretation M, and that e_1 and e_2 are k-discoverable for some k. As entity equivalences discovered without any assumptions hold in all interpretations that satisfies sys, e_1 and e_2 have the same truth value in M. The system that results from overwriting $max(e_1, e_2)$ with $min(e_1, e_2)$ is therefore also satisfied by M. But neither resetting of equivalence classes (sys is singular) nor removal of unconnected triples can affect satisfaction. Therefore $\models sys \rightarrow subst(sys, e_1, e_2)$.

It is clear that all propagation equivalences are k-taut equivalences as sys is a representing (singular) system for ϕ and we have made no assumptions on truth values of subformulas; 0-saturation therefore only propagates information from propositional constant leaves. This entails that the resulting equivalences are tautological. Furthermore, k-taut equivalence trivially implies k-context equivalence. Now assume that $M \models subst(sys, e_1, e_2)$ and that both e_1 and e_2 are k-context equivalent. We must show that $M \models sys$. We know that s_1 and s_2 are assigned the same truth value by M as $\models subst(sys, e_1, e_2) \rightarrow (s_1 \leftrightarrow s_2)$. The valuation tree corresponding to $subst(sys, e_1, e_2)$ can therefore be converted to

a valuation tree for sys by replacing $min(s_1, s_2)$ with $max(s_1, s_2)$ in appropriate places. Consequently $\models subst(sys, e_1, e_2) \rightarrow sys$.

□

We refer to the following minimisation algorithm as *m-reduce*[1]:

1. Find all pairs of m-equivalent entities in the system.
2. If the system was discovered to be explicitly contradictory during the search for m-equivalent entities, the system is equivalent to $\bot$. Stop.
3. Otherwise use the $subst$ operator to iteratively reduce the system using the m-equivalent entity pairs (e_i, e_j) in an order that respects decreasing absolute value of $|e_i| - |e_j|$. Stop.

The resulting equivalent system will be smaller if at least one of the m-equivalent pairs contained entities of differing size. It is furthermore clear that the propositions of the resulting formula are a subset of the propositions of the original formula.

By choosing an appropriate notion of equivalence in step 1, we choose how much work we want to spend on compacting the representation. 0-saturation is a linear algorithm, which makes propagation equivalences easy to find. When the resulting compaction not is enough, the algorithms that use higher degrees of saturation can be applied; they have higher complexity but can find more removable subformulas. 1-saturation is for example guaranteed to find all shared subformulas [Har96].

6.4 The connective operators $m_{\#}$, $m_{\neg}$

It is sufficient to consider the binary connectives ($\neg\phi \equiv \phi \rightarrow \bot$ and $\bot$ can be encoded as a system only containing $\bot$).

We refer to the following algorithm that takes two given systems S_1 and S_2 and builds a composite system under an operator # as the *m-fuse* algorithm:

1. m-reduce S_1 and S_2
2. Build a new system S with the following characteristics
   - Lit contains the literals that occur in the formulas represented by S_1 and S_2.
   - $Triple$ is the union of the sets of triples in S_1 and S_2 after triple identifiers have been changed to preserve uniqueness and literal references have been adjusted to be consistent with Lit.
   - The top triple of S is $i_{top} : i_1 \# \ i_2$, where i_1 and i_2 are the triples that correspond to the old top triples of S_1 and S_2.
   - $[e] = \{e\}$ for all entities e in S.

[1] m is either "propagation", "k-taut" or "k-context"

6.5 Boolean quantification for m_{AX},m_{EX}

We implement boolean quantification by using the identity $\Pi p.\phi \equiv \phi[\top/p] \mathbin{@} \phi[\bot/p]$ for Π quantifier and @ either the disjunction and conjunction operator (see section 2). The substitution operations is simple to express using *subst*; the operator application is implemented by m-fusing.

We refer to the following algorithm as *m-quantify*:

1. Make two copies, sys_1 and sys_2, of the current formula system.
2. Construct $res_1 = subst(sys_1, \top, p)$ and $res_2 = subst(sys_2, \bot, p)$
3. m-fuse res_1 and res_2 under @

Our main concern is to stop the representation from growing too much as each boolean quantification without minimisation potentially doubles the formula size. The use of m-fuse will guarantee m-minimisation, and the systems will be in good position for reductions as at least two formula leaves are propositional constants.

6.6 Discussion

Efficiency The presented algorithms will work well when the representation size can be kept within bounds using low degrees of saturation, and the formulas that arise during a verification are easy so that fixpoint termination can be determined.

However, if it is too hard to detect that a fixpoint has been reached, the model checker can just continue iterating and building new formulas (logically equivalent to the old); paradoxically the hardness degree of a formula can sometimes be reduced by building another, more redundant representation. Hardness of a formula can also be decreased by adding special subformulas that preserves the semantics [SB98]. Which formulas to add seems to be hard to deduce in general, but some heuristic could be possible as we are interested in formulas that arise in a particular way.

One indication that formula hardness not should be a general problem is the fact that experiments with bounded model checking using Stålmarck's method (see next subsection) have generated formulas that are tractable even though the investigated systems have been very hard to verify with traditional methods.

The algorithms that have been presented in this section are primitive and can be refined and optimised. Minimisations can for example be done more often than the algorithms suggest (we have here minimised at least once in each primitive operation), or only when strictly needed. Algorithm complexity will however always be dominated by the cost of k-saturating, an algorithm that is $O(n^{2k+1})$ for n formula size; the remaining parts of the presented algorithms are linear.

Bounded model checking Embedding model checking in propositional logic is not a novel idea; a very promising approach to linear time temporal logic

(LTL) model checking is that taken by Ed Clarke and his colleagues [BCC+] in their work on bounded model checking.

The main idea of bounded model checking (BMC) is to generate a single propositional logic formula that defines the set of length k paths that satisfies the temporal formula according to a suitably modified semantics. A test for PROP satisfiability is therefore enough to prove temporal satisfaction for bounded length paths. It turns out that there is always a finite k such that satisfiability for length k paths is necessary and sufficient to infer that there exists an infinite path that satisfies the temporal formula. A troublesome aspect of the bounded model checking approach is that a bound on k must be determined to guarantee that bounded satisfiability is equivalent to general satisfiability.

We, on the other hand, generate a sequence of formulas ϕ_k that analogously defines the set of states that satisfies a CTL temporal formula for k steps forward. The logical fixpoint of this sequence defines the set of states that satisfies the temporal property with respect to the standard semantics. Rather than producing a single, monolithic formula, however, we have the opportunity of minimising and using domain specific knowledge to guide the process inbetween each step. We can also avoid the issue of finding bounds on k since theorem proving is used to detect that a fixpoint has been reached.

Bounded model checking is closely related to our approach. We have however arrived at a similar solution by considering how sets rather than paths can be represented using logical formulas, which among other things means that we could avoid having to prove that a bound k really exists (we have only changed the representation of sets).

The experimental data presented for bounded model checking is encouraging, and suggests that embeddings into propositional logic can be very good for systems that are hard for BDDs. We believe that the fact that our approach can avoid reasoning about bounds, and that we have the possibility to do internal minimisation when the formulas grow too much could be very useful.

7 Other logics

Up to now, we have been considering a representation that encodes a set as a propositional logic formula. But we are not forced to use propositional logic; any other logic of equal or greater strength is a possible candidate.

For stronger logics it is actually often possible to embed a model checking problem as a single formula that directly expresses the temporal property according to the normal formula semantics. We maintain that there are many benefits to solving a number of smaller subproblems generated by fixpoint calculations rather than approaching the problem monolithically and relinquishing all control to the theorem prover. If a fixpoint approach is taken, the model checking process is divided up into many steps; inbetween each step the process can be guided and the formulas transformed.

We will briefly illustrate the benefits of some other logics by discussing how efficiency could be improved by using QBF, and by indicating how FOL could allow model checking of more complex state spaces.

7.1 Quantified boolean formulas

Set representation in QBF might at first not seem any different from set representation in PROP as there exists a simple translation between the two logics. The translation is however superpolynomial, so we can gain efficiency by allowing boolean quantifiers at the formula level throughout the process: For example, we can sidestep having to immediately fold out the quantifiers, which in some cases avoids generation of huge formulas.

Consider a formula $\phi \leftrightarrow \phi$, where ϕ contains a large number of boolean quantifiers. Syntactic comparison on the QBF level alone is enough to deduce that the formula is a tautology. However, if we are first obliged to transform the QBF formula into an equivalent PROP formula, a large amount of simplification would be necessary to keep the formula size under control.

QBF-level reasoning can also be used to move quantifiers around before a mapping to PROP is done. One example of this would be to push in quantifiers as far as possible by an anti-prenex transformation, or to reduce quantifiers in a different order than inside out. In this way existing PROP provers can be used as back-ends for QBF theorem proving.

We do not know of any dedicated QBF theorem provers, but any FOL prover that has special support for quantifications over finite domains is also capable of QBF level reasoning: Given that Π is a quantifier and that x not is free in ϕ, the QBF formula $\Pi p.\phi$ can be mapped to an equivalent finite domain FOL formula $\Pi x \in \{0,1\}.\phi[(x=1)/p]$.

7.2 A simple use of first order logic

As opposed to propositional logic, first order logic is expressible enough to define any recursively enumerable set. Given that $k \in \mathbb{Z}$ and i_x is an element of a finite set of integer variables IVar, this makes its possible to model check Kripke structures labelled with both ordinary atomic propositions and statements of the form $i_x = k$. We define an an extension to CTL, called iCTL, based on such extended Kripke structures:

In each state s, each $p \in AP$ is true or false and each $i \in$ IVar has an integer value. The atoms of iCTL, AP', are

$$AP \ \cup \ \{i_n = k : n < |\text{IVar}|, k \in \mathbb{Z}\} \ \cup \ \{i_n = i_m : n, m < |\text{IVar}|\}$$

Compound iCTL formula semantics are defined by a straight forward extension of the CTL semantics.

Given an extended Kripke structure M, we define in each state s

$$ival_n^s = k \ \text{ iff } \ M, s \models i_n = k$$

For convenience we again assume that the states are uniquely labelled, and identify the state s with the set

$$\{val_n^s : n < |\text{Var}|\} \cup \{ival_n^s : n < |\text{IVar}|\}$$

Under the assumption that $\Gamma = \{\neg(i = j) : i, j \in \mathbb{Z}, i \neq j\}$, sets of states S are now representable by first order logic formulas ϕ that have propositions from AP', variables from IVar and for which it holds that

$$s \in S \quad \text{iff} \quad s, \Gamma \models \phi$$

The set Γ serves as an axiomatisation of equality over integer constants. Binary relations can analogously be represented with formulas, just as in section 5.

No new operations are needed for iCTL model checking; m_{atom}, $m_{\neg}$, $m_{\#}$, m_{AX}, m_{EX}, m_{μ} and m_{ν} suffice. All of the operations, with the exception of the transition operators, can be defined in the same spirit as for CTL.

The problem with the transition operators is now that quantification over a state will result in quantifiers that both range over propositions and variables. Boolean quantifiers can be handled as before but the quantifiers that range over integer variables are not expandable as their domains are infinite. However, as we will use a first order logic theorem prover, these quantifiers can be left in to be removed by unification.

To implement iCTL in practice, both a minimiser and theorem prover for FOL is needed. One candidate for such a system would be the FOL extension to Stålmarck's method [Lun99], with the algorithms of section 6 lifted to first order logic.

7.3 And more....

There is no need to stop at simple equalities as we did in the case of iCTL. Once the step has been taken to first order logic for set representation, any operations that are needed such as addition or complex conjugation can be axiomatised.

Special logics such as monadic first order logic, Presburger arithmetic and PROP + finite domain arithmetic can also be the foundations for model checking tools with different characteristics, applicable to yet other kinds of systems. The efficiency of the underlying theorem prover and minimiser is really the only limiting factor.

8 Conclusions

Many different trade-offs can be made between representation size and the complexity of primitive model checking operations. Table 1 contains a number of approaches to verification of temporal logic specifications together with the complexity of operations relative to the size of the generated problem(s). Each of the verification methods has its own time/space complexity characteristic, and is therefore preferable for certain systems. For example, enumeration-based model

Verification approach	Complexity of operations	Remarks
Enumeration	Polynomial	
BDD fixpoint calculations	Variable ordering NP-hard	
BMC	Single NP/coNP-complete problem	
PROP fixpoint calculations	coNP-complete subproblems	
QBF fixpoint calculations	PSPACE-complete subproblems	
FOL fixpoint calculations	Possibly undecidable subproblems	Infinite state space possible
Temporal logic theorem proving	Single PSPACE-complete problem	

Table 1. Some approaches to verification of temporal logic specifications

checking has very simple primitive model checking operations but requires a linear amount of memory in the size of the represented sets. BDD based symbolic model checking require operations of medium complexity, but can represent many sets compactly.

The aim of this paper has been to present a new family of model checking algorithms with different tradeoffs, and to demonstrate some benefits of a theorem proving approach to an efficient method of temporal logic theorem proving.

The next step will be to implement the algorithms we have described inside Prover, a commercial tool based on Stålmarcks method, and investigate what systems the approach is suitable for. We are eager to compare results with bounded model checking and investigate how the generalisations to other logics work out in practice.

References

[BCC+] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking Using SAT Procedures Instead of BDDs. Submitted to TACAS'99.

[BCL+94] Jerry Burch, Edmund Clarke, David Long, Kenneth McMillan, and David Dill. Symbolic Model Checking for Sequential Circuit Verification. *Transactions on Computer-Aided Design of Iintegrated Circuit and Systems*, 13(4):401–423, April 1994.

[Bry86] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35, August 1986.

[EC80] E. A. Emerson and E.M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181, July 1980.

[Har96] John Harrison. The Stålmarck method as a HOL derived rule. In *Theorem Proving in Higher Order Logics*, number 1125 in Lecture Notes In Computer Science, pages 221–234. Springer Verlag, 1996.

[Lun99] Lars Lundgren. Stålmarck's method in first order logic. Master's thesis, Chalmers technical university, Dept. of Computing Science, 1999.

[SB98] Mary Sheeran and Arne Borälv. How to Prove Properties of Recursively Defined Circuits using Stålmarck's Method. In Mary Sheeran and Bernhard Möller, editors, *Workshop on Formal Techniques for Hardware and*

Hardware-like Systems (associated with Intl. Conf. of Mathematics of Program Construction), June 1998.

[SS98] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's Proof Procedure for Propositional Logic. In *Formal Methods in Computer-Aided Design*, number 1522 in LNCS, pages 82–100. Springer Verlag, November 1998.

[Stå89] Gunnar Stålmarck. A System for Determining Propositional Logic Theorems by Applying Values and Rules to Triplets that are Generated from a Formula, 1989. Swedish Patent No. 467 076 (approved 1992), U.S. Patent No. 5 276 897 (1994), European Patent No. 0403 454 (1995).