Symbolic Model Checking

Sergey Tverdyshev
deru@wjpserver.cs.uni-sb.de

12.03.2002

Abstract

Finite state models of concurrent systems grow exponentially as the number of components of the system increases. This is known as state explosion problem. To avoid this problem E.M. Clarke and K.L. McMillan proposed a method called *symbolic model checking*. This method avoids building a state graph explicitly by using Boolean formulas to represent sets and relations. Because Boolean formulas represent sets we need to operate on entire sets rather than on individual states and transitions. For this purpose, we use a fixed point representation of temporal operators. We describe a model checking algorithm for μ-Calculus formulas and show a symbolic representation for Boolean formulas.

Introduction

Over the last decade, it has become apparent that finite-state system can often be verified automatically by examining state-graph models of system behaviour. There are a number of different methods, but many of them use explicit representations of the state-space. Because the number of states in the model may grow exponentially with the number concurrently executing components, explicit representation may lead to the state explosion problem. The technique to avoid this problem, proposed by Clarke and McMillan, is to represent the state space symbolically instead of explicitly.
In the first section we define least and greatest fixed point by Tarski [1] and computation algorithms for them. We will show the fixed point representations of CTL operators. In the second section we introduce Quantified Boolean Formulas and symbolic Kripke structures. In the third section we show symbolic model checking algorithm. In the last section we present Binary Decision Diagram, by Bryant, and discuss their efficiency.

1 Fixed Points

Let $M = (S, S_0, R, L)$ be an arbitrary finite Kripke structure, where

- S denotes the set of states

- $S_0 \subseteq S$ denotes the initial states
- $R \subseteq S \times S$ denotes the transition relation
- L denotes the labelling function. $L : S \to 2^{AP}$ where AP is the set of atomic propositions.

The power set of S denoted 2^S is partially ordered under set inclusion $\subseteq$. Each set $S' \subseteq S$ can be thought as a predicate on S, where the predicate is viewed as being true for exactly the states in S'. The least element is the empty set we denote as $False$. The greatest element is the full set S which we denote as $True$. A function that maps 2^S to 2^S is called a predicate transformer. Let τ be such function $\tau : 2^S \to 2^S$. A set $S' \subseteq S$ is a fixed point of τ if $\tau(S') = S'$. We also define the following properties:

Definition 1

- *τ is called monotonic: $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$*
- *τ is $\cup - continuous$: $P_1 \subseteq P_2 \subseteq ...$ implies $\tau(\bigcup_i P_i) = \bigcup_i \tau(P_i)$*
- *τ is $\cap - continuous$: $P_1 \supseteq P_2 \supseteq ...$ implies $\tau(\bigcap_i P_i) = \bigcap_i \tau(P_i)$*

It can be shown that a monotonic predicate transformer on a finite set S is both $\cup$-continuous and $\cap$-continuous. A least fixed point, denoted $\mu Z.\tau(Z)$ is a fixed point such that for all fixed point x it is a subset of x i.e. $\mu Z.\tau(Z) \subseteq x$. A greatest fixed point, denoted $\nu Z.\tau(Z)$ is a fixed point such that for all fixed point x it is a superset of x i.e. $x \subseteq \nu Z.\tau(Z)$. A monotonic predicate transformer always has a least fixed point and a greatest fixed point. Since the predicate transformer is monotonic its least/greatest fixed point can be computed as the intersection/union of all the fixed points, and they are unique [1]. In the following lemma we use the notation $\tau^i(P)$ to denote the i-th application of τ to P.

Lemma 1
Whenever τ is monotonic and $\cup - continuous$: $\mu Z.\tau(Z) = \bigcup_i \tau^i(False)$ and whenever τ is monotonic and $\cap - continuous$: $\nu Z.\tau(Z) = \bigcap_i \tau^i(True)$

Proof
We will only prove the first part of the claim, the proof of the second part is similar. We need to show (a) the sequence $(False = \tau^0(False), \tau^1(False), \tau^2(False), \cdots)$ has a limit, (b) the limit of this sequence is a fixed point and (c) the least fixed point.

(a) We prove the claim by induction on i:

> The base is straightforward since $\tau^0(False) = False$ and $False \subseteq \tau(False)$. From the induction hypothesis $\tau^{i-1}(False) \subseteq \tau^i(False)$ and the monotonicity of τ we can claim that $\tau(\tau^{i-1}(False)) \subseteq \tau(\tau^i(False))$ that is equal to $\tau^i(False) \subseteq \tau^{i+1}(False)$. From the last we can conclude that there exists $i_0 \in \mathbb{N}$ such that for all $i > i_0$ we have $\tau^i(False) = \tau^{i_0}(False)$, because S is finite.

(b) This follows from (a): $\tau^{i_0}(False)$ is a fixed point since $\tau(\tau^{i_0}(False)) = \tau^{i_0+1}(False) = \tau^{i_0}(False)$

function Lfp (least fixed point)	function Gfp (greatest fixed point)
$Q = False$	$Q = True$
$Q' = \tau(False)$	$Q' = \tau(True)$
while$(Q \neq Q')$	**while**$(Q \neq Q^{'})$
{	{
$\quad Q = Q'$	$\quad Q = Q'$
$\quad Q' = \tau(False)$	$\quad Q' = \tau(False)$
}	}
return Q	**return** Q

Figure 1: Least and greatest fixed point algorithms

(c) We need to show that $\tau^{i_0}(False)$ is the least fixed point

Let X be any fixed point. We show by induction that $\tau^i(False) \subseteq X$ for all $i \in \{0, ..., i_0\}$
For the base case we need to show that $False \subseteq X$. This is obviously true. For the induction step consider that the induction hypothesis holds for all $i < i_0$, so $\tau^i(False) \subseteq X$. From the induction hypothesis and the monotonicity of τ follows that $\tau(\tau^i(False)) \subseteq X$ that is equal to $\tau^{i+1}(False) \subseteq X$. Applying claim *(a)* yields $\tau^{i_0}(False) \subseteq X$ that is true for any X fixed point. The last implies $\bigcup_i \tau^i(False) = \tau^{i_0}(False) = \mu Z.\tau(Z)$. □

As a consequence of the preceding lemma, if τ is monotonic, its least/greatest fixed point can be computed by the algorithms in figure 1: it is obvious that at the beginning i-th iteration of the loop, $Q = \tau^{i-1}(False)$ and $Q' = \tau^i(False)$. For the Lfp algorithm lemma 1 implies $False \subseteq \tau(False) \subseteq \tau^2(False) \subseteq \cdots$ and that the loop terminates. The number of iterations is bound by the number of states in the set S.

1.1 Fixed point representation

We identify each CTL formula f with the set of states in which the formula holds: $s(f) = \{s | \quad M, s \models f\}$. For all $Z \subseteq S$ we identify CTL operator EX as follows: $EX(Z) = \{s | \exists s' \quad (s, s') \in R \quad \text{and} \quad s' \in Z\}$.
Now we can define the CTL operators EG, EU in terms of $s(f)$, and fixed points of an appropriate predicate transformer. The other operators can be expressed in terms of the operators above i.e. EX, EG, EU:

Theorem 1

1. $EG(f) = \nu Z.(s(f) \cap EX(Z))$
2. $EU(f_1, f_2) = \mu Z.(s(f_2) \cup (s(f_1) \cap EX(Z)))$

Proof
We give a proof of the first part of the theorem. The proof of the second part is similar. To prove it we need three auxiliary lemmas:

Lemma 2
$\tau(Z) = (s(f) \cap EX(Z))$ is monotonic.

Lemma 3
$EG(F)$ is a fixed point of the predicate transformer τ.

Lemma 4
Let $\tau^{i_0}(True)$ be the limit of the sequence $(True, \tau(True), \tau^2(True), \cdots)$, then

- *for all s in S if s in $\tau^{i_0}(True)$, then s satisfies f.*
- *$\exists s'$ such that $(s, s') \in R$ and $s' \in \tau^{i_0}(True)$*

The proofs of the lemmas are technical; therefore we will omit them. From lemma 1 and the definition of the EG operator we observe: since τ is monotonic it is also $\cap - continuous$. This implies:

$EG(f) = \nu Z.(s(f) \cap EX(Z))$ if and only if $EG(f) = \bigcap_i \tau^i(True)$.

So now we break down our proof in two steps.

1. $EG(f) \subseteq \bigcap_i \tau^i(True)$
2. $EG(f) \supseteq \bigcap_i \tau^i(True)$

Proof of the first claim.
We will prove it by induction on i. The base case ($i = 0$) is straightforward since $\tau^0(True) = True$ and $EG(f) \subseteq True$ is obviously true. For the induction step assume that induction hypothesis $EG(f) \subseteq \tau^i(True)$ holds. Then because of monotonicity of τ the following : $\tau(EG(f)) \subseteq \tau^{i+1}(True)$. Application of the lemma 3, $\tau(EG(f)) = EG(f)$ to the last claim yields $EG(f) \subseteq \bigcap_i \tau^i(True)$.

Proof of the second claim.
Consider the arbitrary state s in $\bigcap_i \tau^i(True)$. For all i we have $s \in \tau^i(True) \Rightarrow s \in \tau^{i_0}(True)$. By lemma 4, the state s is the start of an infinite sequence of states in which each state is related to the previous one by the next-state relation. Furthermore, each state in the sequence satisfies f. Thus $s \models EGf$ i.e. $s \in EG(f) \Rightarrow \bigcap_i \tau^i(True) \subseteq EG(f)$ $\square$

Example 1
We want to compute $EG(f)$ in the Kripke structure depicted in figure 2.
Define $Z = \{s| \quad s \models f\}$, where f is an atomic proposition. The states marked "T" are in the current τ^i. Since we want to compute EG we start from $True$ i.e. all states are marked. Application of the Gfp-algorithm yields the greatest fixed point after the third iteration.

2 Quantified Boolean Formulas

Quantified **B**oolean **F**ormulas (QBF) are an extension of propositional logic allowing quantifiers over propositional variables. Given a set $\mathbf{V}$ of propositional variables, the set of Quantified Boolean Formulas over $\mathbf{V}$, denoted by QBF($\mathbf{V}$), is the least set containing:

1. $true$ and $false$ in $\mathbf{V}$ are QBFs,

$\tau^0(True) = True$

$\tau^1(True) = Z \cap EX(True) = Z.$

$\tau^2(True) = Z \cap EX(Z).$

$\tau^3(True) = Z \cap EX(Z \cap EX(Z)).$

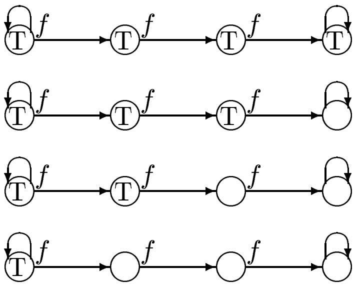

Figure 2: Kripke structure(Example 1)

2. every variable in $\mathbf{V}$ is a QBF,
3. if p and q QBFs, then so are $p \wedge q$ and $\neg p$,
4. if p is a QBF and v is in $\mathbf{V}$, then $\exists v.p$ is a QBF.

We introduce a truth assignment. A truth assignment a is a function $a : \mathbf{V} \rightarrow \{true, false\}$. Let v be in $\mathbf{V}$ and b be in $\{true, false\}$, then we also define the truth assignment $a(v \leftarrow b)$ as
$a(v \leftarrow b)(v') = \begin{cases} v' & \text{if } v' \neq v \\ b & \text{otherwise} \end{cases}$
Also we identify formulas with sets of truth assignments:

1. $v \in \mathbf{V}$ represents the set of truth assignments $\{a| \quad a(v) = true\}$,
2. $a \in (p \vee q)$ if and only if $a \in p$ and $a \in q$,
3. $a \in (\neg p)$ if and only if $a \notin p$,

   It is useful to define an operator for QBFs that substitutes a quantified Boolean formula for a variable.

   Definition 2
   If p and q are quantified Boolean formulas, and v is a variable, then:
   $a \in p(v \leftarrow q)$ *if only if* $a(v \leftarrow c) \in p$*, where c is true whenever* $a \in p$ *and* $false$ *otherwise.*

4. $a \in (\exists v.p)$ if and only if $a(v \leftarrow true) \in p$ or $a(v \leftarrow false) \in p$.

We define quantification in terms of substitutions:
$\exists v.p = p(v \leftarrow false) \vee p(v \leftarrow true)$
Quantification and substitution can also be defined for vectors of variables. If $W = (w_1, \ldots, w_k)$ is a vector of propositional variables and $Q = (q_1, \ldots, q_k)$ is a vector of formulas, then

1. $a \in \exists W.p$ if and only if for some $b : W \rightarrow \{true, false\}, \quad a(w_i \leftarrow b(w_i)) \in p$
2. $a \in p(W \leftarrow Q)$ if and only if $a(w_i \leftarrow (a \in q)) \in p$

2.1 Symbolic Kripke Structure

The state of a concurrent system is generally modelled as a vector where each component of the system represents the state of one variable. Without loss of generality we assume that all of the state components are boolean valued. A state of concurrent system therefore can be viewed as a truth assignment over the variables. Let $V = \{v_1, \ldots, v_n\}$ and $V' = \{v'_1, \ldots, v'_n\}$ be two ordered set of variables. Our goal is to represent any $Q \subseteq S$ by QBF $\mathbf{Q}$ over V, and represent R by QBF $\mathbf{R}$ over $V \cup V'$. Each QBF formula represents the set of truth assignments, which satisfy the formula; therefore we can view each QBF as set of state. For example, if we have two state variables a and b, then the formula $a \vee b$ represents all the states in which a is true or b is true. To represent R symbolically we make the next definition:

Definition 3
Let $(x, y) \in S$ and $a : V \cup V' \to \{true, false\}$, with $a(v_i) := x(v_i)$ and $a(v'_i) := y(v_i)$ then:

$$(x, y) \in R \textit{ if and only if } a \in \mathbf{R}$$

This can also be written in terms of substitutions:

$$(x, y) \in R \textit{ if and only if } x(v'_i \to y(x_i)) \in \mathbf{R}$$

Now we define the symbolic Kripke structure $M = (\mathbf{S}, S_0, \mathbf{R}, L)$ with:

- $\mathbf{S} = \{a : V \to \{true, false\}\}$
- $S_0 \subseteq S$ is the set of initial states.
- $\mathbf{R}$ is the transition relation.
- $L : s \to 2^V, \quad L(s) = \{v_i| \quad s(v_i) = 1\}$

We solve the model checking problem (whenever a given state in a symbolically represented Kripke structure satisfy a formula f) only by manipulation of Boolean formulas. The key point is that the Kripke structure itself is never built, it is symbolically represented by QBF formulas.

Example 2
We want to check whether or not in the Kripke structure from figure 3 the state s_1 satisfies CTL formula $EX(\neg b)$. We have got $V = \{b\}, \quad V' = \{b'\}$, and $\mathbf{R} = b \vee b'$. In the Kripke structure above $s = \{s_1, s_2\}$, $L(s_0) = \{\}$ and $L(s_1) = \{b\}$. The set of states that satisfies CTL-formula $(EX(\neg b))$ in state s_1 is:
$s(EX \neg b) = \exists b'.(\mathbf{R} \wedge (\neg b)(b \leftarrow b')) = \exists.((b \vee b') \wedge (\neg b')) = \exists b'(b \vee \neg b')$
$= (b \wedge \neg false) \vee (b \wedge \neg true) = b.$
The answer is no, because $s_1 \notin \mathbf{b}$. Note that here $\mathbf{b}$ and b have the same meaning, because of the definition of QBFs.

3 Symbolic Model Checking Algorithm

Figure 4 depicts the complete procedure for symbolic model checking. This function is recursively defined. It takes as input an QBF and returns another QBF. The returned QBF presents the set of states satisfies to the given CTL formula in the given Kripke structure.

Note that to determine when a limit is reached in the fixed point series (in the procedures *CheckEU* and *CheckEG*), it is necessary to determine whether two symbolic representations of sets are equal; therefore it is very useful to use a canonical

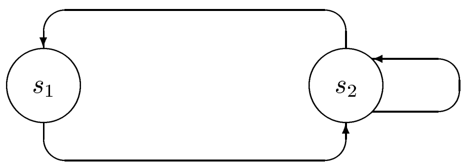

Figure 3: Kripke structure (Example 2)

$Check(f)$:
- f an atomic proposition: *return* f
- $f = \neg p :$ *return* $\neg Check(f)$
- $f = p \vee q :$ *return* $Check(p) \vee Check(q)$
- $f = EXp :$ *return* $CheckEX(Check(p))$
- $f = EGp :$ *return* $CheckEG(Check(p), true)$
- $f = EU(p,q) :$ *return* $CheckEU(Check(p), Check(q), false)$

$CheckEX(\mathbf{p}) = \exists V'.(\mathbf{R} \wedge \mathbf{p}(V \leftarrow V'))$

$CheckEG(\mathbf{p}, \mathbf{y})$
- $\mathbf{y'} = \mathbf{p} \wedge CheckEX(\mathbf{y})$
- *if* $\mathbf{y'} = \mathbf{y}$ *then return* $\mathbf{y}$
- *else return* $CheckEG(\mathbf{p}, \mathbf{y'})$

$CheckEU(\mathbf{p}, \mathbf{q}, \mathbf{y})$
- $\mathbf{y'} = \mathbf{q} \vee (\mathbf{p} \wedge CheckEX(\mathbf{y}))$
- *if* $\mathbf{y'} = \mathbf{y}$ *then return* $\mathbf{y}$
- *else return* $CheckEU(\mathbf{p}, \mathbf{q}, \mathbf{y'})$

Figure 4: Symbolic CTL model checking algorithm

form of representation (see section 4). Since the model has 2^n states, the maximal number of iterations needed to reach a fixed point is $2^n + 1$. In practice, however, the number of iterations can be quite small.

4 Binary Decision Diagram

To make symbolic model checking practical, we need an efficient algorithm for manipulating Boolean formulas. In particular, we must be able to reduce formulas including existential quantifiers to a suitable representation, so that we can test formulas for equality.
Reduced ordered binary decision diagrams (ROBBDs) are a canonical form of representation for Boolean formulas. They are often smaller than traditional normal forms. As an illustration of reduction to canonical form we first consider Ordered Binary Decision Tree. A binary decision tree is a rooted, directed tree that consists of two types of nodes, terminal and nonterminal nodes. Each nonterminal node v is labelled by a variable $var(v)$ and has two successors: $low(v)$ corresponding to the case where the variable v is assigned 0, and $high(v)$ corresponding to the case

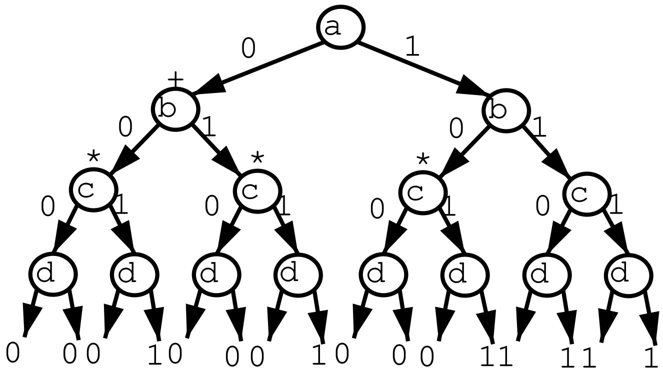

Figure 5: Ordered Decision Tree

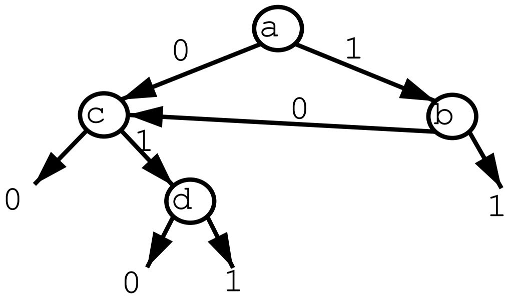

Figure 6: Ordered Binary Decision Diagram

where v is assigned 1. Each terminal vertex v is labelled by $value(v)$ which is either 1 or 0. Figure 5 depicts the ordered decision tree for formula $a \wedge b \vee c \wedge d$. Formulas are evaluated by following a path through the tree and returning the value at the leaf.

Binary decision trees do not provide concise representation for Boolean formulas. However, there is a lot of redundancy in such trees. For example, in the figure 5 we can see that the nodes marked " * " are the roots of isomorphic subtrees. Thus they can be combined into a single subtree. From the node marked " + " we arrive in the same tree; therefore this node does not affect the value of the function and may be eliminated. After applying the reduce operations to the tree we obtain OBDD, that is depicted in figure 6. For all evaluations the OBDD it gives the same result as the original tree.

OBDDs have been shown to be compact for a wide variety of functions. The size of an OBDD, however, is strongly dependent on the ordering of variables. Also OBDD is a unique representation of Boolean formula. A few important examples do not have a compact OBDD representations e.g. multipliers, barrel shifter. Bryant [2] has shown, efficient algorithms for Boolean operations (can be computed in quadratic time in the size of the input trees), and quantification.

Summary

Fixed point computation and fixed point representation of CTL formulas allow us to compute the set of states that a CTL formula satisfies. Symbolic Kripke structure make this method practical; model checking can be done only by manipulations of Boolean formulas. Binary Decision Diagram is a way to do these manipulations efficiently.

References

[1] A. Tarski A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5.

[2] R.E. Bryant Graph based algorithms for Boolean function manipulations. IEEE Transactions on Computers, C-35(8) 1986

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill Symbolic Model Checking: 10^{20} States and Beyond 1992.

[4] Kenneth L. McMillan Symbolic Model Checking. *An approach to the state explosion problem.* 1992

[5] Edmund M. Clarke, Orna Grumberg and Doron A. Peled Model Checking. MIT Press, Cambridge, Massachusetts; London, England, 1999

[6] Henry Reef Andersen An Introduction to Binary Decision Diagram. *Lecture notes for Efficient Algorithms and Programs.* The IT University of Copenhagen, 1999