

COMP6463: Temporal Logic and Model Checking

5. Topics in Symbolic Model Checking

Michael Norrish

Canberra Research Lab., NICTA

Semester 1, 2009



NICTA

Outline

- 1 Introduction
- 2 Generating Counter-examples
- 3 Important Model Checking Optimisations
 - BDD Optimisations
 - Model Optimisations
- 4 Bounded Model Checking
- 5 Summary

Symbolic Model Checking in Summary

- The fundamental model checking problem is the **state space explosion**
- **BDDs** provide efficient representations of sets of boolean variables
- BDDs can be used to **check CTL formulas**
 - states (and transition relations) are no longer explicit, but represented **symbolically** with BDDs

Counter-Examples

(The MC advocate's view of) **Interactive Theorem Proving**:

- 1 State desired goal
- 2 Spend **long, arduous** period attempting to prove goal through application of appropriate tactics *etc*
- 3 If successful, stop.
- 4 Otherwise, carefully mull over failed goal state until you realise what was wrong with original goal. Return to 1.

Counter-Examples

(The MC advocate's view of) **Interactive Theorem Proving**:

- 1 State desired goal
- 2 Spend **long, arduous** period attempting to prove goal through application of appropriate tactics *etc*
- 3 If successful, stop.
- 4 Otherwise, carefully mull over failed goal state until you realise what was wrong with original goal. Return to 1.

Model Checking:

- 1 State desired goal
- 2 Push button
- 3 If successful, stop.
- 4 Otherwise, system returns **counter-example** demonstrating why goal is false. Correct restatement of goal is easy!

Counter-Examples and Witnesses are Dual

If $AG \phi$ is false at w , a path starting at w and leading to a state where ϕ is false is a **counter-example**.

In other words, the counter-example is a **witness** to the truth of $EF(\neg\phi)$.

Properties to be checked are often **universal**:

- generating witnesses to existential goals is an important tool.
- And if the user's goal is existential, they may find witnesses reassuring.

Counter-Example Requirements

Given starting world w , need to be able to generate witnesses to:

$EX \phi$: a neighbouring world w' where ϕ is true (easy)

$E[\phi_1 \cup \phi_2]$: a finite path to a world where ϕ_2 is true, keeping ϕ_1 true in the meantime (handles EF)

$EG \phi$: an infinite path with ϕ true the whole time.

Ideally, it would be nice to have shortest possible counter-examples.

Witnesses for $E[\phi_1 \cup \phi_2]$

Assume we know which states satisfy $E[\phi_1 \cup \phi_2]$

- (our initial world w is in this set)

Begin a breadth-first search at w :

- only consider neighbours where $E[\phi_1 \cup \phi_2]$ is true
- stop when you find a world satisfying ϕ_2

Using a breadth-first search ensures the shortest path is found.

Witnesses for $\text{EG } \phi$

Need an infinite path.

This means a loop.

Again, starting at w

- breadth first search;
- only move to neighbours where $\text{EG } \phi$ is true;
- stop when a loop is found.

Counter-Examples and Fairness Constraints

Things are trickier if your infinite path also has to hit **fairness constraints**.

Details on this (and symbolic model checking with fairness) are in Clarke's book.

Outline

- 1 Introduction
- 2 Generating Counter-examples
- 3 Important Model Checking Optimisations**
 - BDD Optimisations
 - Model Optimisations
- 4 Bounded Model Checking
- 5 Summary

BDD Optimisations

If R is the parallel, asynchronous composition of n components, R will be something like:

$$\bigvee_{i < n} \left(R_i(v_i, v'_i) \wedge \bigwedge_{j \neq i} (v'_j = v_j) \right)$$

(plus fairness constraints!)

Turning this into one BDD may be impractical.

The critical BDD operation is to calculate the various **image** sets.

Optimising Image Set Computations

Imagine a 3-component system on (x, y, z) .

So R is $R((x, y, z), (x', y', z'))$ (a BDD on 6 variables).

Instead of

$$\begin{aligned} Reach^{i+1}(v) = \\ Reach^i(v) \vee \exists v_0. Reach^i[v := v_0] \wedge R[v := v_0, v' := v] \end{aligned}$$

Optimising Image Set Computations

Imagine a 3-component system on (x, y, z) .

So R is $R((x, y, z), (x', y', z'))$ (a BDD on 6 variables).

Have

$$\begin{aligned} Reach^{i+1}(x, y, z) = \\ Reach^i(x, y, z) \vee \\ \exists x_0 y_0 z_0. Reach^i(x_0, y_0, z_0) \wedge R((x_0, y_0, z_0), (x, y, z)) \end{aligned}$$

This formula includes the potentially huge R which we know is really “either x goes to x' , or y goes to y' , or z goes to z' ”.

Optimising Image Set Computations

Imagine a 3-component system on (x, y, z) .

So R is $R((x, y, z), (x', y', z'))$ (a BDD on 6 variables).

So, we really have

$$\begin{aligned} Reach^{i+1}(x, y, z) = \\ Reach^i(x, y, z) \vee \\ \exists x_0 y_0 z_0. Reach^i(x_0, y_0, z_0) \wedge \\ \left(\begin{array}{c} R_x(x_0, x) \wedge (y_0 = y) \wedge (z_0 = z) \\ \vee \\ (x_0 = x) \wedge R_y(y_0, y) \wedge (z_0 = z) \\ \vee \\ (x_0 = x) \wedge (y_0 = y) \wedge R_z(z_0, z) \end{array} \right) \end{aligned}$$

and if we use $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r) \dots$

Optimising Image Set Computations

Imagine a 3-component system on (x, y, z) .

So R is $R((x, y, z), (x', y', z'))$ (a BDD on 6 variables).

$$\begin{aligned} Reach^{i+1}(x, y, z) = \\ Reach^i(x, y, z) \vee \\ \exists x_0 y_0 z_0. \\ \left(\begin{array}{c} Reach^i(x_0, y_0, z_0) \wedge R_x(x_0, x) \wedge (y_0 = y) \wedge (z_0 = z) \\ \vee \\ Reach^i(x_0, y_0, z_0) \wedge (x_0 = x) \wedge R_y(y_0, y) \wedge (z_0 = z) \\ \vee \\ Reach^i(x_0, y_0, z_0) \wedge (x_0 = x) \wedge (y_0 = y) \wedge R_z(z_0, z) \end{array} \right) \end{aligned}$$

And now, we use $(\exists x.P(x) \vee Q(x)) \equiv (\exists x.P(x)) \vee (\exists x.Q(x))$

Optimising Image Set Computations

Imagine a 3-component system on (x, y, z) .

So R is $R((x, y, z), (x', y', z'))$ (a BDD on 6 variables).

$$\begin{aligned} Reach^{i+1}(x, y, z) = \\ Reach^i(x, y, z) \vee \\ \left(\begin{array}{c} \exists x_0. Reach^i(x_0, y, z) \wedge R_x(x_0, x) \\ \vee \\ \exists y_0. Reach^i(x, y_0, z) \wedge R_y(y_0, y) \\ \vee \\ \exists z_0. Reach^i(x, y, z_0) \wedge R_z(z_0, z) \end{array} \right) \end{aligned}$$

and the expensive existentials need only be done over the component descriptions, not all of R .

Making Models Smaller

The state-space explosion is the problem that arises if the models become too large.

So rather than being smart with BDDs (“compiler optimisations”), why not be smart at the high level?

Cone of Influence Reduction

Input 1: Huge, unwieldy description of whole system

Input 2: Property stating that two specific flags are never both high together.

Cone of Influence Reduction

Input 1: Huge, unwieldy description of whole system

Input 2: Property stating that two specific flags are never both high together.

Important Realisation: Only part of Input 1 is relevant!

Reduce work by throwing away all parts of model that do not affect the variables appearing in the specification.

Abstraction

Assume there is a **high-level description** of the model.

- Don't compile this into BDD directly.

Instead **abstract** by merging similar states.

For example, abstract an 8-bit register x (with 256 values) into one of

$$x^0, x^+ \text{ or } x^-$$

Resulting Kripke structure will be **much** smaller

- but propositions that may be true at each state are more limited too

Abstraction

Theorem

If an ACTL formula (CTL with only A -path quantifiers) is true of an abstracted system, then it is true of the full system too.

(Recall that the ACTL formulas possible over the abstracted system will not be as expressive as those possible over the full system.)

Symmetry Reductions

Another form of abstraction.

If a system is full of repetitions and/or similar units, abstract, then

- find a symmetry map on sets of states (e.g., flip components 1 & 2)
- **quotient**, reducing number of states
- model-check smaller system

Provisoes

- Quotiented system must be bisimilar to original
- Propositions must be invariant under symmetry map

Outline

- 1 Introduction
- 2 Generating Counter-examples
- 3 Important Model Checking Optimisations
 - BDD Optimisations
 - Model Optimisations
- 4 Bounded Model Checking**
- 5 Summary

Bounded Model Checking

- Principally a **bug finding** method
- **Completeness** may be lost
- **Performance** can be very good
- Finds errors/faults up to a particular **bound**
 - is there an error in paths of length **k** or less?
- Only checks simple properties (not full CTL)

BDDs are Great But. . .

BDDs can be used to check **big** systems

- but real systems are even bigger

BDDs are very sensitive to **variable order**

- e.g., wrong order for **comparator** results in **exponential** size
- a good variable order for a circuit counts as an **industrial secret** (Intel)

BDDs can **blow up** in terrible and unexpected ways

- using them becomes the domain of the expert
- even if the final BDD is small, intermediate BDDs may be very large

And In the Blue Corner: SAT

SAT solvers are usually very **space efficient**

- contrast potentially exponentially-sized BDDs
- (can still take exponential time of course)

SAT solvers **don't** require careful tuning or variable orders

- Dynamic heuristics for variable choice and clause learning work well without user intervention

A great deal of research continues to go into making SAT solvers better and better.

(You already know all this from earlier in this course!)

SAT the Universal Acid

SAT, the quintessential hard problem (Cook, 1971)

- SAT is hard, so reduce SAT to your problem

SAT, the universal constraint solver (various people, 1990s)

- SAT is easy, so reduce your problem to SAT

(Slide stolen from Daniel Jackson @ MIT)

SAT the Universal Acid

SAT, the quintessential hard problem (Cook, 1971)

- SAT is hard, so reduce SAT to your problem

SAT, the universal constraint solver (various people, 1990s)

- SAT is easy, so reduce your problem to SAT

(Slide stolen from Daniel Jackson @ MIT)

So how do we use SAT to solve Model Checking problems?

SAT Applied to Model Checking

The transition relation for a Kripke structure can be represented with a propositional formula over v and v' (vectors of variables).

BDD-based model checking uses

- existential quantification,
- substitution, and
- equality tests

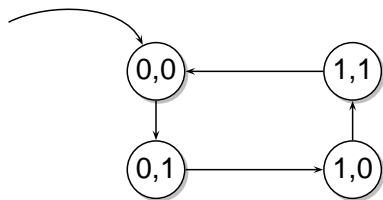
to calculate images and fix-points.

Bounded model checking keeps everything as a propositional formula that is checked for satisfiability.

- As bound k increases, so too does the size of the formula

Calculating R^k

Say R is the relation for a two bit (a and b) counter:

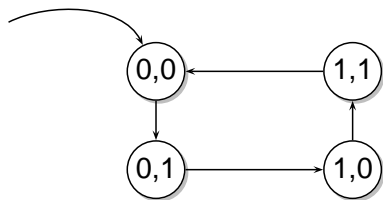


$$R \equiv b' = \neg b \wedge a' = a \oplus b$$

$$I \equiv \neg a_0 \wedge \neg b_0$$

Calculating R^k

Say R is the relation for a two bit (a and b) counter:



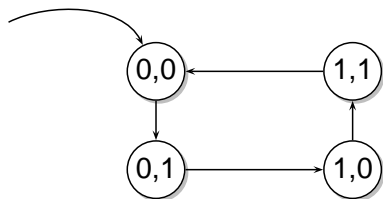
$$R \equiv b' = \neg b \wedge a' = a \oplus b$$

$$I \equiv \neg a_0 \wedge \neg b_0$$

$$R^1 \equiv (a_1 = a_0 \oplus b_0) \wedge (b_1 = \neg b_0)$$

Calculating R^k

Say R is the relation for a two bit (a and b) counter:



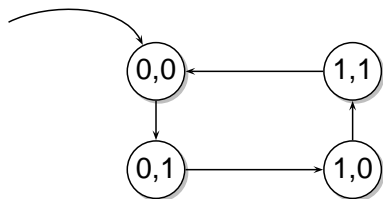
$$R \equiv b' = \neg b \wedge a' = a \oplus b$$

$$I \equiv \neg a_0 \wedge \neg b_0$$

$$R^2 \equiv (a_1 = a_0 \oplus b_0) \wedge (b_1 = \neg b_0) \wedge (a_2 = a_1 \oplus b_1) \wedge (b_2 = \neg b_1)$$

Calculating R^k

Say R is the relation for a two bit (a and b) counter:



$$R \equiv b' = \neg b \wedge a' = a \oplus b$$

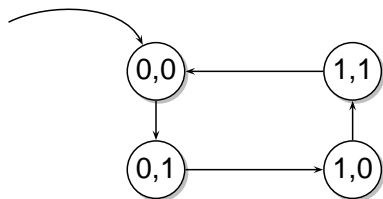
$$I \equiv \neg a_0 \wedge \neg b_0$$

$$R^3 \equiv (a_1 = a_0 \oplus b_0) \wedge (b_1 = \neg b_0) \wedge \\ (a_2 = a_1 \oplus b_1) \wedge (b_2 = \neg b_1) \wedge \\ (a_3 = a_2 \oplus b_2) \wedge (b_3 = \neg b_2)$$

Etc, etc...

Calculating R^k

Say R is the relation for a two bit (a and b) counter:



$$R \equiv b' = \neg b \wedge a' = a \oplus b$$
$$I \equiv \neg a_0 \wedge \neg b_0$$

Can check $AG \phi$ property (ϕ propositional only) to k steps by checking

$$I \wedge R^k \wedge \bigvee_{0 \leq i \leq k} \neg \phi[a := a_i, b := b_i]$$

Formula should be unsatisfiable.

Iterative Deepening

Check $AG \phi$ property (ϕ propositional only) over state s by checking

$$I \wedge R^k \wedge \bigvee_{0 \leq i \leq k} \neg \phi[s := s_i]$$

Repeatedly check with increasing k until

- formula is satisfiable (assignment gives counterexample);
- you get bored;
- SAT problems become too big to decide; or
- you know that k is big enough that you have checked all possible paths (can be difficult)

BMC for Liveness Properties

Imagine checking $AF \phi$ (or similar $A[\phi_0 \cup \phi]$) property.

- *i.e.*, on every path some desirable state satisfying ϕ is eventually reached.

What does a counter-example look like?

BMC for Liveness Properties

Imagine checking $AF \phi$ (or similar $A[\phi_0 \cup \phi]$) property.

- *i.e.*, on every path some desirable state satisfying ϕ is eventually reached.

What does a counter-example look like?

- Because system is finite-state: a counter-example is a path with a loop where none of the states in the path satisfy ϕ .
(In until case: a finite path with ϕ_0 becoming false and no sign of ϕ is also a counter-example.)

BMC for Liveness Properties

$\text{AF } \phi$: a counter-example is a path with a loop where none of the states in the path satisfy ϕ .

For each k , test satisfiability of:

$$I \wedge R^k \wedge \left(\bigwedge_i \neg \phi(s_i) \right) \wedge \left(\bigvee_{i < k} s_i = s_k \right)$$

A satisfying assignment demonstrates the existence of a loop disproving the liveness of ϕ .

BMC for Inductive Invariants

Wish to show that (again, propositional) ϕ is true for all reachable states. (i.e., $I \Rightarrow AG \phi$)

Suffices to show

$$I \Rightarrow \phi(s_0)$$

$$R(s, s') \wedge \phi(s) \Rightarrow \phi(s')$$

This is a very fast method for showing certain properties true of a system.

- Finding a ϕ that implies the desired property might be arbitrarily hard

Parameterising Induction Proofs with a Bound

If we show

$$I \Rightarrow \phi(s_0)$$

$$R(s, s') \wedge \phi(s) \Rightarrow \phi(s')$$

call this induction with “depth 0”.

May get better coverage by doing induction to “depth k ”

$$I \wedge R^k \Rightarrow \bigwedge_{i \leq k} \phi(s_i) \quad (\text{base cases})$$

$$\left(\bigwedge_{i \leq k} R(s_i, s_{i+1}) \wedge \phi(s_i) \right) \Rightarrow \phi(s_{i+1}) \quad (\text{step case})$$

When doing the induction, allowed to assume that ϕ holds on k previous states.

Parameterising Induction Proofs with a Bound

If we show

$$I \Rightarrow \phi(s_0)$$

$$R(s, s') \wedge \phi(s) \Rightarrow \phi(s')$$

call this induction with “depth 0”.

May get better coverage by doing induction to “depth k ”

$$I \wedge R^k \Rightarrow \bigwedge_{i \leq k} \phi(s_i) \quad (\text{base cases})$$

$$\left(\bigwedge_{i \leq k} R(s_i, s_{i+1}) \wedge \phi(s_i) \right) \Rightarrow \phi(s_{i+1}) \quad (\text{step case})$$

When doing the induction, allowed to assume that ϕ holds on k previous states.

- Can also add constraint that the $s_i \dots s_{i+k}$ are all distinct (quadratic in size though)

Summary—Making Model-Checking Usable

- Make sure false specifications result in useful counter-examples
- Optimise, Optimise, Optimise!
 - be smart about application of BDD operations
 - spot opportunities to make high-level models smaller
- Exploit SAT technology, drop BDDs, and find bugs