Introduction to Bounded Model Checking

Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University
Linz, Austria

FATS Seminar

ETH Zürich, Switzerland
Wednesday, October 28, 2009
A Personal History of Model Checking

Burch, Clarke, McMillan, Dill, Hwang '90: Symbolic Model Checking

Davis, Putnam '60: DP

Coudert, Madre '89: Symbolic Reachability

Davis, Logemann, Loveland '62: DPLL

McMillan '93: SMV

Bryant '86: BDDs

Een, Sorenson '03: MiniSAT

Pnueli '77: Temporal Logic

Boyer '86: BDDs

Biere, Artho, Schuppan '01: Liveness2Safety

Quielle, Sifakis '82: Model Checking

Biere, Cimatti, Clarke, Zhu '99: Bounded Model Checking

Kurshan '93: Localization

Een, Biere '05: SatELite

McMillan '93: SMV

Sheeran, Singh, Stalmarck '00: \(k\)-Induction

Clarke, Emerson, Sifakis:
Turing Award 2007

Holzmann '91: SPIN

Ball, Rajamani '01: SLAM

Clarke, Grumberg, Jah, Lu, Veith '03: CEGAR

Holzmann '81: On-The-Fly Reachability

Graf, Saidi '97: Predicate Abstraction

Peled '94: Partial-Order-Reduction

Introduction to Bounded Model Checking – FATS Seminar ETH 2009

Armin Biere – FMV – JKU Linz
What is Model Checking?

- mechanically check properties of models

- models:
  - finite automata, labelled transition systems
  - often requires automatic/manual abstraction techniques

- properties:
  - mostly interested in *partial properties*
  - specified in temporal logic: CTL, LTL, etc.
  - safety: something bad should not happen
  - liveness: something good should happen

- automatic generation of counterexamples
Reachability

- set of states $S$, initial states $I$, transition relation $T$

- bad states $B$ reachable from $I$ via $T$?

- symbolic representation of $T$ (circuit, program, parallel product)
  - avoid explicit matrix representations, because of the
  - state space explosion problem, e.g. $n$-bit counter: $|T| = O(n)$, $|S| = O(2^n)$
  - makes reachability PSPACE complete [Savitch’70]

- on-the-fly [Holzmann’81’] for protocols
  - restrict search to reachable states
  - simulate and hash reached concrete states
Forward Fixpoint: Initial and Bad States
Forward Fixpoint: Step 1
Forward Fixpoint: Step 2
Forward Fixpoint: Step 3
Forward Fixpoint: Bad State Reached
Forward Fixpoint: Termination, No Bad State Reachable
initial states $I$, transition relation $T$, bad states $B$

\[
\text{model-check}^\mu_{\text{forward}} (I, T, B)
\]

\[
S_C = \emptyset; \quad S_N = I;
\]

\[
\textbf{while } S_C \neq S_N \textbf{ do}
\]

\[
\textbf{if } B \cap S_N \neq \emptyset \textbf{ then}
\]

\[
\quad \text{return} \quad \text{“found error trace to bad states”};
\]

\[
S_C = S_N;
\]

\[
S_N = S_C \cup \text{Img}(S_C);
\]

\[
\textbf{done};;
\]

\[
\text{return} \quad \text{“no bad state reachable”};
\]
Symbolic Model Checking

- work with symbolic representations of states
  - symbolic representations are potentially exponentially more succinct
  - favors BFS: next frontier set of states in BFS is calculated symbolically

- originally “symbolic” meant model checking with BDDs
  [CoudertMadre’89/’90,BurchClarkeMcMillanDillHwang’90,McMillan’93]

- Binary Decision Diagrams [Bryant’86]
  - canonical representation for boolean functions
  - BDDs have fast operations (but image computation is expensive)
  - often blow up in space
  - restricted to hundreds of variables
### Linear Size BDD for Bit-Vector Comparison

**Boolean function/expression:**

\[
\bigwedge_{i=0}^{n-1} x_i = y_i
\]

**Interleaved variable order:**

\[
x_3 > y_3 > x_2 > y_2 > x_1 > y_1 > x_0 > y_0
\]

**Comparison of two \( n \)-bit-vectors needs \( 3 \cdot n \) inner nodes for the interleaved variable order**
missing edges lead to 0

Exponential BDD for Bit-Vector Comparison

Model Checking 13
Unrolling of Forward Least Fixpoint Algorithm

0: continue? \( S^0_C \neq S^0_N \) \( \exists s_0[I(s_0)] \)

0: terminate? \( S^0_C = S^0_N \) \( \forall s_0[\neg I(s_0)] \)

0: bad state? \( B \cap S^0_N \neq \emptyset \) \( \exists s_0[I(s_0) \land B(s_0)] \)

1: continue? \( S^1_C \neq S^1_N \) \( \exists s_0, s_1[I(s_0) \land T(s_0, s_1) \land \neg I(s_1)] \)

1: terminate? \( S^1_C = S^1_N \) \( \forall s_0, s_1[I(s_0) \land T(s_0, s_1) \rightarrow I(s_1)] \)

1: bad state? \( B \cap S^1_N \neq \emptyset \) \( \exists s_0, s_1[I(s_0) \land T(s_0, s_1) \land B(s_1)] \)

2: continue? \( S^2_C \neq S^2_N \) \( \exists s_0, s_1, s_2[I(s_0) \land T(s_0, s_1) \land T(s_1, s_2) \land \neg (I(s_2) \lor \exists t_0[I(t_0) \land T(t_0, s_2)])] \)

2: terminate? \( S^2_C = S^2_N \) \( \forall s_0, s_1, s_2[I(s_0) \land T(s_0, s_1) \land T(s_1, s_2) \rightarrow I(s_2) \lor \exists t_0[I(t_0) \land T(t_0, s_2)] \)

2: bad state? \( B \cap S^1_N \neq \emptyset \) \( \exists s_0, s_1, s_2[I(s_0) \land T(s_0, s_1) \land T(s_1, s_2) \land B(s_2)] \)
Falsification Part of Fixpoint Algorithm

0: continue? \( S_C^0 \neq S_N^0 \) \( \exists s_0[I(s_0)] \)

0: terminate? \( S_C^0 = S_N^0 \) \( \forall s_0 [\neg I(s_0)] \)

0: bad state? \( B \cap S_N^0 \neq \emptyset \) \( \exists s_0[I(s_0) \land B(s_0)] \)

1: continue? \( S_C^1 \neq S_N^1 \) \( \exists s_0,s_1[I(s_0) \land T(s_0,s_1) \land \neg I(s_1)] \)

1: terminate? \( S_C^1 = S_N^1 \) \( \forall s_0,s_1[I(s_0) \land T(s_0,s_1) \rightarrow I(s_1)] \)

1: bad state? \( B \cap S_N^1 \neq \emptyset \) \( \exists s_0,s_1[I(s_0) \land T(s_0,s_1) \land B(s_1)] \)

2: continue? \( S_C^2 \neq S_N^2 \) \( \exists s_0,s_1,s_2[I(s_0) \land T(s_0,s_1) \land T(s_1,s_2) \land \neg (I(s_2) \lor \exists t_0[I(t_0) \land T(t_0,s_2)])] \)

2: terminate? \( S_C^2 = S_N^2 \) \( \forall s_0,s_1,s_2[I(s_0) \land T(s_0,s_1) \land T(s_1,s_2) \rightarrow I(s_2) \lor \exists t_0[I(t_0) \land T(t_0,s_2)]] \)

2: bad state? \( B \cap S_N^1 \neq \emptyset \) \( \exists s_0,s_1,s_2[I(s_0) \land T(s_0,s_1) \land T(s_1,s_2) \land B(s_2)] \)
• look only for counter example made of $k$ states (the bound)

BMC($k$): $I(s_0) \land T(s_0, s_1) \land \ldots \land T(s_{k-1}, s_k) \land \bigvee_{i=0}^{k} B(s_i)$

• simple for safety properties: bad state $B$ is reachable

• harder for liveness properties: cycle with no progress states $N$ reachable

$I(s_0) \land T(s_0, s_1) \land \ldots \land T(s_{k-1}, s_k) \land \bigwedge_{i=0}^{k} N(s_i) \land \exists \ell T(s_k, s_\ell)$

• can also encode liveness into safety [BiereArthoSchuppan’01]
• look only for counter example made of $k$ states (the bound)

\[
\begin{array}{ccccccc}
  s_0 & \rightarrow & s_1 & \rightarrow & s_l & \rightarrow & s_{l+1} & \rightarrow & s_k \\
  B & \lor & B & \lor & B & \lor & B
\end{array}
\quad \text{or} \quad
\begin{array}{ccccccc}
  s_0 & \rightarrow & s_1 & \rightarrow & s_l & \rightarrow & s_{l+1} & \rightarrow & s_k \\
  N & \land & N & \land & N & \land & N & \land & N
\end{array}
\]

• simple for safety properties: bad state $B$ is reachable

\[
\text{BMC}(k) : \quad I(s_0) \land T(s_0, s_1) \land \cdots \land T(s_{k-1}, s_k) \land \bigvee_{i=0}^{k} B(s_i)
\]

• harder for liveness properties cycle with no progress states $N$ reachable

\[
I(s_0) \land T(s_0, s_1) \land \cdots \land T(s_{k-1}, s_k) \land \bigwedge_{i=0}^{k} N(s_i) \land \bigvee_{l=0}^{k} T(s_k, s_l)
\]

• can also encode liveness into safety [BiereArthoSchuppan'01]
Bounded Model Checking State of the Art

- increase in efficiency of SAT solvers [Grasp, zChaff, MiniSAT, SatELite,…]

- SAT more robust than BDDs in bug finding
  
  (shallow bugs are easily reached by explicit model checking or testing)

- better unbounded but still SAT based model checking algorithms
  
  - $k$-induction [SinghSheeranStalmarck’00]
  
  - interpolation [McMillan’03]

- 4th Intl. Workshop on Bounded Model Checking (BMC’06)

- other logics, better encodings, e.g. [LatvalaBiereHeljankoJuntilla-FMCAD’04]

- other models, e.g. C/C++/Verilog [Kröning…], hybrid automata [Audemard…-BMC’04]
Induction with SAT

[SinghSheeranStalmarck’00]

- more specifically, **k-induction**

  - does there exist $k$ such that the following formula is **unsatisfiable**

$$B(s_0) \land \cdots \land B(s_{k-1}) \land T(s_0, s_1) \land \cdots \land T(s_{k-1}, s_k) \land B(s_k) \land \bigwedge_{0 \leq i < j \leq k} s_i \neq s_j$$

  - if **unsatisfiable** and BMC($k$) **unsatisfiable** then **bad state unreachable**

- bound on $k$: length of **longest cycle free path** = reoccurrence diameter

- $k = 0$ check whether $\neg B$ tautological (propositionally)

- $k = 1$ check whether $\neg B$ inductive for $T$
Interpolation in Model Checking

[McMillan’03]

• SAT based technique to overapproximate frontiers $\text{Img}(S_C)$
  – currently most effective technique to show that bad states are unreachable
  – better than BDDs and $k$-induction in many cases [HWMCC’08]

• starts from a **resolution proof** refutation of a BMC problem with bound $k + 1$

$$
S_C(s_0) \land T(s_0, s_1) \land T(s_1, s_2) \land \cdots \land T(s_k, s_{k+1}) \land B(s_{k+1})
$$

  – result is a characteristic function $f(s_1)$ over variables of the second state $s_1$
  – these states do not reach the bad state $s_{k+1}$ in $k$ steps
  – any state reachable from $S_C$ satisfies $f$: $S_C(s_0) \land T(s_0, s_1) \Rightarrow f(s_1)$

• $k$ is bounded by the diameter (exponentially smaller than longest cycle free path)
Generating Propositional Interpolants from Resolution

A ∧ B unsatisfiable then f is an interpolant iff

(I1) \( A \Rightarrow f \) and (I2) \( B \land f \Rightarrow \bot \)

an interpolating quadruple \( (A, B) c [f] \) is well formed if

(W1) \( V(c) \subseteq V(A) \cup V(B) \) and (W2) \( V(f) \subseteq G \cup V(c) \) with \( G = V(A) \cap V(B) \)

an interpolating quadruple \( (A, B) c [f] \) is valid if

(V1) \( A \Rightarrow f \) and (V2) \( B \land f \Rightarrow c \)

proof rules which produce well formed and valid interpolating quadruples:

\[
\text{(R1)} \quad \frac{\text{c } \in \text{A}}{(A, B) \ c \ [c]} \quad \frac{(A, B) \ c \lor \ l \ [f]}{(A, B) \ c \lor \ d \ [f \land g]} \quad |l| \ \in \ V(B) \quad \text{(R3)}
\]

\[
\text{(R2)} \quad \frac{\text{c } \in \text{B}}{(A, B) \ c \ [\top]} \quad \frac{(A, B) \ c \lor \ l \ [f]}{(A, B) \ c \lor \ d \ [f \land g]} \quad |l| \ \notin \ V(B) \quad \text{(R4)}
\]
Mining Inductive Invariants

- through abstract interpretation resp. static analysis, or alternatively ...

- randomly simulate model and extract potential invariants
  - signals / predicates which always hold
  - implications of signals / predicates that occur in the simulation / tests
  - equivalent signals (works well in sequential equivalence checking)

- prove them to be $k$-inductive
  - quite natural in sequential equivalence checking for circuits
  - synthesis algorithms also only see finitely many time steps

- how to obtain environment model / constraints / contracts?
Using Inductive Invariants

• inductive invariants help to speed-up both $k$-induction (and interpolation)

• let $P$ be inductive: $I(s) \Rightarrow P(s)$ and $T(s, s') \land P(s) \Rightarrow P(s')$

• we want to prove that a bad state can not reached

• if BMC($k$) is unsatisfiable it is enough to prove unsatisfiability of

$$P(s_0) \land P(s_1) \land \cdots \land P(s_k) \land$$

$$\overline{B(s_0)} \land \cdots \land \overline{B(s_{k-1})} \land$$

$$T(s_0, s_1) \land \cdots \land T(s_{k-1}, s_k) \land B(s_k) \land \bigwedge_{0 \leq i < j \leq k} s_i \neq s_j$$

• this formula can become unsatisfiable much earlier, i.e. for smaller $k$, than

$$\overline{B(s_0)} \land \cdots \land \overline{B(s_{k-1})} \land$$

$$T(s_0, s_1) \land \cdots \land T(s_{k-1}, s_k) \land B(s_k) \land \bigwedge_{0 \leq i < j \leq k} s_i \neq s_j$$
Simple Path Constraints

- bounded model checking: \cite{BiereCimattiClarkeZhu'99}

\[
I(s_1) \land T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \land \bigvee_{0 \leq i \leq k} B(s_i) \quad \text{satisfiable?}
\]

- reoccurrence diameter checking: \cite{BiereCimattiClarkeZhu'99}

\[
T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \land \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j \quad \text{unsatisfiable?}
\]

- \( k \)-induction base case: \cite{SheeranSinghStålmarck'00}

\[
I(s_1) \land T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \land B(s_k) \land \bigwedge_{0 \leq i < k} \neg B(s_i) \quad \text{satisfiable?}
\]

- \( k \)-induction induction step: \cite{SheeranSinghStålmarck'00}

\[
T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \land B(s_k) \land \bigwedge_{0 \leq i < k} \neg B(s_i) \land \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j \quad \text{unsatisfiable?}
\]
All Different Constraints (ADC)

- classical concept in constraint programming:
  - $k$ variables over a domain of size $m$ supposed to have different values
  - for instance $k$-queen problem

- propagation algorithms to establish arc-consistency
  - explicit propagators: [Régin’94]
    - $O(k \cdot m)$ space
    - $O(k^2 \cdot m^2)$ time
  - symbolic propagators: [GentNightingale’04] also [MarquesSilvaLynce’07]
    - one-hot CNF encoding with $\Omega(k \cdot m)$ boolean variables

- in model checking $k \ll m$ typically $k < 1000 \quad m = 2^n > 2^{100}$ n latches
Symbolic ADCs for Large Domains

- encoding bit-vector inequalities directly:
  - let \( u, v \) be two \( n \)-bit vectors, \( d_0, \ldots, d_{n-1} \) fresh boolean variables
    \[
    u \neq v \quad \text{is equisatisfiable to} \quad (d_0 \lor \cdots \lor d_{n-1}) \land \bigwedge_{j=0}^{n-1} (u_j \lor v_j \lor \overline{d_j}) \land (u_j \lor \overline{v_j} \lor \overline{d_j})
    \]
  - can be extended to encode Ackermann Constraints + McCarthy Axioms
  - either **eagerly** encode all \( s_i \neq s_j \) quadratic in \( k \)
  - or **refine** adding bit-vector inequalities on demand \( [\text{EénSörensson-BMC’03}] \)

- **natively handle ADCs within SAT solver:** main contribution in FMCAD’08
  - similar to theory consistency checking in lazy SMT vs. “lemmas on demand”
  - can be extended to also perform theory propagation

- sorting networks ineffective in our experience \( [\text{KröningStrichman’03,JussilaBiere’06}] \)
Lemmas on Demand for ADCs

Abstract

SAT? NO YES

Spurious? NO YES

Refine

just leave out ADCs
do not encode them

add violated ADC(s) as Lemma on Demand

call SAT solver incrementally

check solution:
violates any original ADC?
Lemmas on Demand for Satisfiability Modulo Theory (SMT)

Abstract

- Call SAT solver
- SAT? NO
- SAT? YES
- Spurious? NO
- Spurious? YES

Refine

- Only keep boolean skeleton of original first-order formula
- Add instance of violated axiom as lemma on demand
- Check solution: violates any theory axiom
- NO
- NO
- YES
Localization / Counter Example Guided Abstraction Refinement

Localization [Kurshan’93], Predicate Abstraction [GrafSaidi’97], SLAM [BallRajamani’01], CEGAR [ClarkeGrumbergJhaLuVeith’03]

Abstract

Refine

add more logic of model as lemma on demand

cut connections in model locally around property

call model checker incrementally

SAT?

Spurious?

check solution:
impossible to extend to full model

NO

YES

NO

YES

NO
Lemmas on Demand for Extensional Arrays

Abstract

- Call SAT solver
- SAT?
  - NO
  - YES
- Spurious?
  - NO
  - YES
- Refine
  - YES
  - NO

- Add instance of violated axiom as lemma on demand
- Replace array reads and equality checks by fresh variables
- Call SAT solver incrementally
- Check solution: violates any array axiom
Early Unsat Termination

Encode input to CNF

Add under-approx. clauses C

Refine under-approx.

Call SAT solver

SAT? NO YES

C used? YES NO

NO YES

Formula is satisfiable

Formula is unsatisfiable
Combining Over- and Under-Approximation

Array formula

Over-approximate

Encode to CNF

Add under-approx. clauses C

Refine under-approx.

Refine over-approx.

Formula is unsatisfiable

Call SAT solver

SAT?

C used?

Formula is satisfiable

Call SAT solver

spurious?

Add lemma

spurious?

YES

YES

NO

YES

NO

NO

NO

YES
Lazy SMT

- Lemmas on Demand are as lazy as it gets
  - SAT solver enumerates full models of propositional skeleton
  - abstracted lemmas are added / learned on demand
  - theory solver checks consistency of conjunction of theory literals

- on-the-fly consistency checking
  - additionally theory solver checks consistency of partial model as well

- theory propagation
  - theory solver even deduces and notifies SAT solver about implied values of literals

- generic framework: DPLL(T) [NieuwenhuisOliverasTinelli-JACM’06]
All Different Objects (ADOs)

Symbolic All-Different Constraints

[BiereBrummayer-FMCAD’08]

Introduction to Bounded Model Checking – FATS Seminar ETH 2009
Armin Biere – FMV – JKU Linz
All Different Objects (ADOs)

Symbolic All-Different Constraints [BiereBrummayer-FMCAD’08]

ADO for \( v \)

ADO for \( w \)

ADO for \( u \)

hash
All Different Objects (ADOs)

Symbolic All-Different Constraints

[Brummayer-FMCAD'08]

0 \ u_1 \ 1

ADO for \ u \\

v_2 \ v_1 \ v_0

ADO for \ v \\

w_2 \ w_1 \ w_0

ADO for \ w \\

hash
All Different Objects (ADOs)

0  $\mathit{u_1}$  1

ADIO for $\mathit{u}$

$\mathit{v_2}$  $\mathit{v_1}$  $\mathit{v_0}$

ADIO for $\mathit{v}$

$\mathit{w_2}$  $\mathit{w_1}$  $\mathit{w_0}$

ADIO for $\mathit{w}$

hash

move
All Different Objects (ADOs)

Symbolic All-Different Constraints
[BiereBrummayer-FMCAD’08]

ADO for $u$

ADO for $v$

ADO for $w$

hash
All Different Objects (ADOs)

- ADO for $u_0 1$
- ADO for $v_0$
- ADO for $w_2 1$

Hash
All Different Objects (ADOs)

\[ \begin{align*}
\text{ADO for } u & \quad \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \\
\text{ADO for } v & \quad \begin{bmatrix} 0 & 1 & v_0 \end{bmatrix} \\
\text{ADO for } w & \quad \begin{bmatrix} w_2 & 1 & 1 \end{bmatrix}
\end{align*} \]

complete

hash
All Different Objects (ADOs)

- Symbolic All-Different Constraints

- Insertions into an ADO for $u$, $v$, and $w$
All Different Objects (ADOs)

Symbolic All-Different Constraints
[BiereBrummayer-FMCAD'08]

Introduction to Bounded Model Checking – FATS Seminar ETH 2009

Armin Biere – FMV – JKU Linz
All Different Objects (ADOs)

ADO for $u$

ADO for $v$

ADO for $w$

conflict $u = v$

hash
• ADO key is calculated from concrete bit-vector
  – by for instance XOR’ing bits word by word

• ADOs watched by variables (not literals)
  – during backtracking all inserted ADOs need to be removed from hash table
  – save whether variable assignment forced ADO to be inserted
  – stack like insert/remove operations on hash table allow open addressing

• conflict analysis
  – all bits of the bit-vectors in conflict are followed
  – can be implemented by temporarily generating a pseudo clause

\[(u_2 \lor \overline{u}_1 \lor \overline{u}_0 \lor v_2 \lor \overline{v}_1 \lor \overline{v}_0)\]
Mixed Approach versus Refine Only

Symbolic All-Different Constraints

[BiereBrummayer-FMCAD'08]
Conclusion on Symbolic All-Different Constraints

- symbolic consistency checker for ADCs over bit-vectors
  - successfully applied to simple path constraints in model checking
  - similar to theory consistency checking in lazy SMT solvers
  - combination with eager refinement approach

- future work: ADC based BCP for bit-vectors
  - aka theory propagation in lazy SMT solvers
  - extensions to handle Ackermann constraints or even McCarthy axioms
  - one-way to get away from pure bit-blasting in BV
Summary

• bounded model checking
  – routinely used in HW industry for falsification
  – need to improve word-level techniques for SW and HW verification / falsification

• SAT (and SMT) has seen tremendous improvements in recent years
  – was key enabler to make bounded model checking successful
  – many applications through the whole field of computer science

• still lots of opportunities:
  – parallel Model Checking / parallel SMT and SAT solving
  – portfolio and preprocessing (PrecoSAT was our first attempt)
  – make quantified boolean formula (QBF) reasoning work (QBF is PSPACE compl.)