Classical and Non-Classical Uses of SAT in Model-Checking

CP meets CAV
Master Class

Jean-François Raskin
Université Libre de Bruxelles
Objectives

• Give representative examples of the use of SAT solvers in verification algorithms for finite state systems

• Disclaimer I: not my work

• Disclaimer II: by no means a full review of the literature (examples only)
Plan

- Bounded model-checking
- Unbounded model-checking
- Inductive invariant generation
Preliminaries
Transition Systems

The basic model for CAV of reactive systems = Transition systems = Directed graphs with labels

Vertices = System/Prg states

Edges = transitions from states to states

Labels = basic properties of states
The basic model for CAV of reactive systems

= Transition systems

= Directed graphs with

Vertices = System/Prg states

Transitions from states to states

Labels = basic properties of states

Usually far too large to be represented explicitly
Symbolic transition systems

- Let $\mathcal{B}(X)$ denotes the set of Boolean formulas over $X$, the variables of the system (or abstractions of them).

- For $F \in \mathcal{B}(X)$, we note $\llbracket F \rrbracket = \{ v : X \to \{0,1\} | v \models F \}$.

- A Symbolic Transition System (STS) $S=(X,I,T)$ where:
  - $X$ is a set of boolean variables.
  - $I \in \mathcal{B}(X)$ defines the initial states.
  - $T \in \mathcal{B}(X \cup X')$ defines the transition relation.
Symbolic transition systems

- We associate to STS=(X,I,T) an explicit, so exponentially larger, transition system TS=(S,S₀,E):
  - \( S = \{ v \mid v : X \to \{0,1\} \} \)
  - \( S₀ = \{ v \in S \mid v \vDash I \} = \llbracket S₀ \rrbracket \)
  - \( E = \{ (v,v') \mid (v,v') \vDash T \} = \llbracket T \rrbracket \)
Typical verification questions

- **Safety**: do all the executions of the system avoid a given set of bad states?
Typical verification questions

- **Reachability:** is there an execution of the system that reaches **good** states? (dual of safety)
Typical verification questions

- **Liveness**: are all the executions of the system doing eventually/repeatedly something good?
Circuit Example

Model:

\[ C = \{ g = a \land b, \quad p = g \lor c, \quad c' = p \} \]

Can we reach a state of the circuit in which \( c \land \neg p \) holds?
Bounded model-checking
[BCC+99]
Bounded model-checking

- Falsifying safety properties

Let \( STS=(X,I,T) \) and \( \text{Bad} \in \mathcal{B}(X) \)

- Is there a \( \llbracket T \rrbracket \)-path from \( \llbracket I \rrbracket \) to \( \llbracket \text{Bad} \rrbracket \)?

- **Bound:**

  Is there a \( \llbracket T \rrbracket \)-path from \( \llbracket I \rrbracket \) to \( \llbracket \text{Bad} \rrbracket \) of length at most \( k \)?
System unfolding

Models

Transition system described by a set of constraints

Each circuit element is a constraint

Note: $a = a_t$ and $a' = a_{t+1}$

Model:

$C = \{ g = a \land b, \ p = g \lor c, \ c' = p \}$

$k$ unfolding

Can the circuit reach a state where $c$ is true in at most $k$ steps?

Bad
Unfolding of $T$

- **Unfolding of $T$ $k$ times:**

  \[ T(X_0, X_1) \land T(X_1, X_2) \land \ldots \land T(X_{k-2}, X_{k-1}) \]

- **Use SAT solver to check satisfiability of**

  \[ I(X_0) \land T(X_0, X_1) \land T(X_1, X_2) \land \ldots \land T(X_{k-2}, X_{k-1}) \land \lor_{i=0..k-1} \text{Bad}(X_i) \]

- **A satisfying assignment corresponds to a path of length at most $k$ from $⟦I⟧$ to $⟦\text{Bad}⟧$, i.e. a **counter-example** to the safety property**

- **Formulas above can easily be expressed as sets of clauses and so can be readily analyzed by a **Boolean SAT solver**
Completeness threshold

- **Diameter** of a system = length of the longest simple path in the transition system

- Bounded model-checking for safety property with a bound $k = \text{diameter of the system}$ ensures completeness

- Unfortunately, computing the diameter of a symbolic transition system is hard. Indeed deciding if the diameter of a symbolic transition system is equal to $k$ is $\text{PSPACE-C}$ (so as hard as the verification problem itself)
Beyond safety

- Let $\text{Good} \in \mathcal{B}(x)$

- Given an infinite path $\rho$ in TS, we note $\text{Inf}(\rho)$ the set of states that appear infinitely many times along $\rho$

- An infinite path in TS is **good** if $\text{Inf}(\rho) \cap [\text{Good}] \neq \emptyset$

- **Liveness**: check that all paths in TS are **good**

- Counter-examples are **lasso-path** such that the cycle does not contain any good states

- **Bound**: find a lasso-path of length at most $k$ that does not cross $[\text{Good}]$ in the lasso part
Beyond safety

- Encoding in SAT:

\[
\begin{align*}
&I(X_0) \\
&\wedge T(X_0, X_1) \wedge \ldots \wedge T(X_{k-2}, X_{k-1}) \\
&\wedge \bigvee_{m=0..k-1} T(X_{k-1}, X_m) \\
&\wedge j=m..k-1 \neg \text{Good}(X_j)
\end{align*}
\]

Liveness is violated
Beyond safety

- Encoding in SAT:

\[ I(X_0) \land T(X_0, X_1) \land \ldots \land T(X_{k-2}, X_{k-1}) \land \bigvee_{m=0..k-1} T(X_{k-1}, X_m) \land \bigwedge_{j=m..k-1} \neg Good(X_j) \]

All this can be extended to linear temporal logic specifications (LTL)

Lasso
Unbounded Model-Checking
Four examples of unbounded SAT based MC

• Symbolic Reachability Analysis based on SAT Solvers [ABE00]
• Unbounded Sat-based model-checking with abstractions [CCKSVW02] + McMillan variant
• Interpolation and unbounded SAT-based model-checking [McMillan03]
• Discovering inductive invariants in subset constructions
Symbolic Reachability Analysis based on SAT Solvers [ABE00]
Symbolic Forward/Backward Reachability

• Let $\text{STS}=(X,I,T)$ and let $\text{Bad} \in \mathcal{B}(X)$

• $\text{ReachFwd}(I)$ is the least set of states $R$ such that $R = [I] \cup \text{Post}[T](R)$
Forward exploration
Post

Diagram:
- Green: Success states
- Light green: Potential exploration states
- Dark green: Current exploration state
- Red: Bad state

...
Symbolic Forward/Backward Reachability

- Let $\text{STS} = (X,I,T)$ and let $\text{Bad} \in \mathcal{B}(X)$

- $\text{ReachFwd}(I)$ is the least set of states $R$ such that $R = \llbracket I \rrbracket \cup \text{Post}[T](R)$

- $\text{ReachBack}(\text{Bad})$ is the least set of states $B$ such that $B = \llbracket \text{Bad} \rrbracket \cup \text{Pre}[T](B)$
Backward exploration
Pre
Symbolic Forward/Backward Reachability

- Let $STS= (X, I, T)$ and let $Bad \in \mathcal{B}(X)$

- $\text{ReachFwd}(I)$ is the least set of states $R$ such that $R = \langle I \rangle \cup \text{Post}[T](R)$

- $\text{ReachBack}(Bad)$ is the least set of states $B$ such that $B = \langle Bad \rangle \cup \text{Pre}[T](B)$

- Symbolic MC: fixpoints + data structures for manipulating sets symbolically
BDDs

share suffixes

remove unnecessary tests
BDDs - Canonicity and Succinctness

• BDDs are canonical representation for Boolean functions

• Make very easy to check fixed-point

• Fact: some Boolean functions have provably large BDD representations, e.g. binary multiplication

• Idea: use potentially more compact representations... at the expense of canonicity and (maybe) some algorithmic efficiency
Boolean circuits

\[
\begin{array}{c}
\text{reduce}
\end{array}
\]
Boolean circuits

- As BDDs, **Boolean circuits** represent sets of valuations (=states)
- There is no (useful) canonical form
- There are often more compact than BDDs
- Algorithms exists for Boolean op. (obviously) and for computing PRE and POST images
- Satisfiability is **NP-Complete**
Boolean circuits

• As BDDs, **Boolean circuits** represent sets of valuations (=states)

• There is no (useful) canonical form

• There are often more compact than BDDs

• Algorithms exists for Boolean op. (obviously) and for computing PRE and POST images

• Satisfiability is **NP-Complete**

  use SAT
Checking satisfiability of Boolean circuits with SAT

\[
\begin{align*}
(i_0 &\iff \neg i_1 \land i_2) \\
\land (i_1 &\iff i_3 \iff i_4) \\
\land (i_2 &\iff i_3 \land i_4) \\
\land (i_3 &\iff x \land z) \\
\land (i_4 &\iff z \land y) \\
\land \neg i_0
\end{align*}
\]

Not equivalent but satisfiability is maintained
SMC algorithm using BC and SAT

$R_0 = [I]$ as a BC

$R_{i+1} = \text{Post}[T](R_i)$ as a BC + $\exists$ elim.

Check $R_{i+1} \iff R_i$ using SAT

Check unsat $R_{i+1} \land \text{Bad}$ using SAT

Unsat? Yes

KO No

Unsat? Yes

OK No
Unbounded SAT-based model-checking with abstractions [CCKSVW02]
Abstractions

• Symbolic model-checking sensitive to the number of Boolean variables (symbolic state explosion problem)

• But (coarse) abstractions are often sufficient to prove correctness

• Try to lower the number of variables using abstraction
Predicates on program/circuit state space

States satisfying the same predicates are (considered) equivalent

Merged into one abstract state
State-space partitioning

Abstract transition relation

\[ T^\alpha(A_1, A_2) \]

iff

\[ \exists s_1 \in A_1 \cdot \exists s_2 \in A_2 \cdot T(s_1, s_2) \]
**State-space partitioning**

**Abstract** transition relation

\[ T^\alpha(A_1, A_2) \]

iff

\[ \exists s_1 \in A_1 \cdot \exists s_2 \in A_2 \cdot T(s_1, s_2) \]
State-space partitioning

Analyze the abstract graph

Overapproximation:
Safe $\Rightarrow$ System Safe

No false positives

Problem
Spurious counterexamples
Counterex.-Guided Refinement

[Kurshan et al 93] [Clarke et al 00][Ball-Rajamani 01]

Solution
Use spurious counterexamples to refine abstraction!
Counterex.-Guided Refinement

[Kurshan et al93] [Clarke et al 00][Ball-Rajamani 01]

Solution

Use spurious counterexamples to refine abstraction

1. **Add predicates** to distinguish states across cut
2. Build **refined** abstraction

Imprecision due to merge
Iterative Abstraction-Refinement

1. Add predicates to distinguish states across cut
2. Build refined abstraction - eliminates counterexample
3. Repeat search till real counterexample or system proved safe

Solution
Use spurious counterexamples to refine abstraction
Abstraction refinement

Choose initial \( C' \subseteq C \)

\( \text{M.C.\,Abstr}(C') \)

Cex valid in \( C \) ?

Add constr. to \( C' \)

C' = subset of the constraints that define the system

OK

KO

No
Abstraction refinement

1. Choose initial $C' \subseteq C$
2. M.C. Abstr($C'$)
3. Cex valid in $C$?
   - Yes: OK
   - No: Add constr. to $C'$
4. Use BDDs
5. Use SAT
Abstract Cex - Safety

- Abstract variables $Y = \text{Support}(C', l, \text{Bad})$

- Abstract system is model-checked using BDD-based symbolic MC with variables in $Y$ only and $|Y| \ll |X|$

- Abstract counter-example is a truth assignment to $\{ y_t \mid y \in Y \land 0 \leq t \leq k \}$ where $k$ is the number of steps in the counter-example
Concretization of Cex

• The abstract Cex $A^\alpha$ satisfies:
  $$A^\alpha(Y) = I(Y_0) \land T_{0..k-1}(Y_0,...,Y_{k-1}) \land \bigvee_{i=0..k-1} Bad(Y_i)$$

• Search for a concrete $A$ consistent with $A^\alpha$:
  $$A^\alpha(Y) \land I(X_0) \land T_{0..k-1}(X_0,...,X_{k-1}) \land \bigvee_{i=0..k-1} Bad(X_i)$$
  =BMC but **guided** by the abstract Cex

• If **unsat** Cex cannot be made concrete and it is **spurious**
Refinement

- Refinement: **add** constraints to C’
- Goal: to **eliminate** the Cex in the next abstract model
- There are many technics for that
- One based on SAT machinery: use **resolution based refutation** of the unsat formula that defines the concretization of the abstract counter-example
Resolution based refinement

- \( A^\alpha(Y) \land I(X_0) \land T_{0..k-1}(X_0,\ldots,X_{k-1}) \land \lor_{i=0..k-1} \text{Bad}(X_i) \) is unsatisfiable

- SAT solver returns unsatisfiable and produces an UNSAT CORE

- \( A^\alpha \) cannot be extended to a concrete Cex: CORE is sufficient to prove it

- Add CORE to \( C' \)
Abstraction refinement

Choose initial $C' \subseteq C$

M.C. Abstr($C'$)

Cex valid in $C$?

Add CORE to $C'$

OK

KO

Cex

No
Variation [McMillan03]

1. **BMC at depth k using SAT solver**
   - Cex? Yes → KO
   - Cex? No → Use refutation to define abstraction

2. **Use refutation to define abstraction**
   - True? Yes → OK
   - False? → Yes

3. **MC Abstraction**
   - Cex? No → k++

**Abstraction is not necessary a refinement of previous one**

Conclude when k is large enough.
Interpolation based unbounded Sat-based model-checking [McMillan03]
Interpolant

• An interpolant \( I \) for an unsatisfiable formula \( A \land B \) is a formula such that
  
  • \( A \Rightarrow I \) \( % \) \( I \) overapproximates \( A \)
  
  • \( I \land B \) is unsatisfiable
  
  • \( I \) only refers to the common variables of \( A \) and \( B \)
  
  • Ex: \( A \equiv p \land q, \ B \equiv \neg q \land r, \ I \equiv q \)
  
  • Intuitively, \( I \) is the set of facts that the SAT solver considers relevant to prove \( A \land B \) unsatisfiable
Interpolation and SAT-MC

- First, call \textsc{BMC}(ST,p,k) \quad p=\text{invariant}?
- Decompose \textsc{BMC}(ST,p,k) into \text{Pref}(ST,p,k) \land \text{Suff}(ST,p,k), where
  - \text{Pref}(ST,p,k) = \text{init} + \text{first transition}
  - \text{Suff}(ST,p,k) = k-1 \text{ last transitions} + \neg p
- if formula is SAT, we have Cex
- Otherwise, compute \( I \) for \text{Pref}(ST,p,k) \land \text{Suff}(ST,p,k)

\begin{align*}
\text{Pref}(ST,p,k) & \quad \text{Suff}(ST,p,k) \\
\downarrow & \quad \downarrow \\
\text{I} & \quad \text{Post(I)} & \quad k-1 \text{ steps}
\end{align*}
Interpolation and SAT-MC

**Fact:** the interpolant $\mathcal{I}$ overapproximates the set of initial states and those accessible in one step and that do not lead to bad states within $k$ steps (quality of the overapproximation)

**Idea:** iterate from a new set of initial states : $\mathcal{I}$
procedure interpolation \((M, p)\)
1. initialize \(k\)
2. while true do
3. if \(BMC(M, p, k)\) is SAT then return counterexample
4. \(R = I\)
5. while true do
6. \(M' = (S, R, T, L)\)
7. let \(C = \text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
8. if \(C\) is SAT then break (goto line 15)
9. /* \(C\) is UNSAT */
10. compute interpolant \(I\) of \(\text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
11. \(R' = I\) is an over-approximation of states reachable from \(R\) in one step.
12. if \(R \Rightarrow R'\) then return verified
13. \(R = R \lor R'\)
14. end while
15. increase \(k\)
16. end while
end
Interpolation procedure

procedure interpolation \((M, p)\)
1. initialize \(k\)
2. while true do
3. if \(BMC(M, p, k)\) is SAT then return \textit{counterexample}\)
4. \(R = I\)
5. while true do
6. \(M' = (S, R, T, L)\)
7. let \(C = \text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
8. if \(C\) is SAT then break (goto line 15)
9. /* \(C\) is UNSAT */
10. compute interpolant \(\mathcal{I}\) of \(\text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
11. \(R' = \mathcal{I}\) is an over-approximation of states reachable from \(R\) in one step.
12. if \(R \Rightarrow R'\) then return \textit{verified}\)
13. \(R = R \lor R'\)
14. end while
15. increase \(k\)
16. end while
end
Interpolation procedure

procedure interpolation \((M, p)\)
1. initialize \(k\)
2. while \text{true} do
3. if \(BMC(M, p, k)\) is SAT then return \textit{counterexample}
4. \[ R = I \]
5. while true do
6. \[ M' = (S, R, T, L) \]
7. let \( C = \text{Pref}(M', p, k) \land \text{Suff}(M', p, k) \)
8. if \( C \) is SAT then break (goto line 15)
9. /* \( C \) is UNSAT */
10. compute interpolant \( I \) of \( \text{Pref}(M', p, k) \land \text{Suff}(M', p, k) \)
11. \[ R' = I \] is an over-approximation of states reachable from \( R \) in one step.
12. if \( R \Rightarrow R' \) then return \textit{verified}
13. \[ R = R \lor R' \]
14. end while
15. increase \( k \)
16. end while
end

Potentially spurious counter-example due to over-approximation
Interpolation procedure

procedure interpolation \((M,p)\)
1. initialize \(k\)
2. while true do
3. if \(BMC(M,p,k)\) is SAT then return counterexample
4. \(R = I\)
5. while true do
6. \(M' = (S, R, T, L)\)
7. let \(C = \text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
8. if \(C\) is SAT then break (goto line 15)
9. /* \(C\) is UNSAT */
10. compute interpolant \(I\) of \(\text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
11. \(R' = I\) is an over-approximation of states reachable from \(R\) in one step.
12. if \(R \Rightarrow R'\) then return verified
13. \(R = R \lor R'\)
14. end while
15. increase \(k\)
16. end while
end
Procedure Interpolation

procedure interpolation \((M, p)\)
1. initialize \(k\)
2. while true do
3. \(\text{if } BMC(M, p, k) \text{ is SAT then return counterexample}\)
4. \(R = I\)
5. while true do
6. \(M' = (S, R, T, L)\)
7. \(\text{let } C = \text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
8. \(\text{if } C \text{ is SAT then break (goto line 15)}\)
9. \(\text{/* } C \text{ is UNSAT */}\)
10. \(\text{compute interpolant } \mathcal{I} \text{ of } \text{Pref}(M', p, k) \land \text{Suff}(M', p, k)\)
11. \(R' = \mathcal{I} \text{ is an over-approximation of states reachable from } R \text{ in one step.}\)
12. \(\text{if } R \Rightarrow R' \text{ then return verified}\)
13. \(R = R \lor R'\)
14. end while
15. increase \(k\)
16. end while
end

2.9 Quantification-based Model Checking
There are many approaches \cite{25} to doing quantifier elimination which is a key step in reachability analysis. The purely SAT-based quantifier elimination procedure introduced in \cite{20} works by enumeration of all the satisfying assignments. The SAT solver is modified to generate all the satisfying assignments by adding blocking clauses to the problem each time an assignment is found. The SAT solving process is continued until no new solutions are found. A blocking clause, which refers only to the state variables, represents the negation of a state cube. This quantification procedure yields a purely SAT-based method for computing the preimage in backward symbolic model checking.

A recent quantification-based method \cite{10} uses a circuit representation of the blocking constraints and use a hybrid solver that works directly on this representation. This enables circuit cofactoring with respect to the input assignments to simplify the circuit graph in each enumeration step. This results in more solutions in each enumeration step and thus far fewer enumerations steps. It is reported in \cite{10} that this method outperforms the technique described in Interpolation procedure when \(k=\text{diameter}\), the abstract algorithm concludes ! But most often it concludes much earlier ! This is a complete framework !
Discovering inductive invariants in subset constructions
Inductive invariants
Inductive invariants

I

Inv

Bad
Verifying inductive invariants

- Let $\text{STS}=(X,I,T)$ be a symbolic transition system
- $\text{Inv} \in \mathcal{B}(X)$ is an inductive invariant
  - iff
  - $\text{Inv}(X) \land T(X,X') \implies \text{Inv}(X')$
  - iff
  - $\neg (\text{Inv}(X) \land T(X,X') \implies \text{Inv}(X'))$ is UNSAT
How to discover inductive invariants?
Universality of NFA

• Nond. finite automata $A = (Q, \Sigma, q_0, \delta, F)$

• $L(A) \neq \Sigma^*$ iff there exists a word $w$ such that all runs on $w$ end up in $Q \setminus F$.

• Special case for $L(A) \subseteq ? L(B)$, PSpace-C.
Universality of NFA

- Can be solved through reachability in STS (subset construction)
- **Hard** because one Boolean variable per state of the automaton - BDDs do not scale
- But special class of STS: monotonicity
- There are practical alternative algorithms to BDDs, based on antichains for example
“Closed” subset construction

Transition relation can be “closed” without changing the language.

Init: sets containing initial states of A
Final: sets containing no accepting states of A

$A^c$: 

$\{1, 2, 3\}$ $\rightarrow$ $\{3, 4, 5, 6\}$ $\rightarrow$ $\{3, 4, 5, 6, 7\}$ $\rightarrow$ $\{3, 4, 5, 6, 8\}$

those sets can be added safely

Final:

$\{1, 2, 3\}$
$\{1, 3\}$
$\{1, 2\} \{2, 3\}$
$\{1\} \{2\} \{3\}$

Init:

$\{1\}$
$\{1, 2\} \{1, 3\}$
$\{1, 4\}$
$\{1, 2, 3\}$
$\{1, 2, 4\}$
Forward analysis

\[ \uparrow \{ q_0 \} \]

\[ U_1 = U_0 \cup \text{Post}(U_0) \]

\[ \cdots \]

\[ U_{i+1} = U_i \cup \text{Post}(U_i) \]

\[ \cdots \]

\[ U^* = U^* \cup \text{Post}(U^*) \]

\[ \downarrow F \]

\[ \cap \]

\[ \neq ? \emptyset \]
Forward analysis

\[ U_i = U_0 \cup \text{Post}(U_0) \]

\[ U_{i+1} = U_i \cup \text{Post}(U_i) \]

\[ U^* = U^* \cup \text{Post}(U^*) \]

\[ \downarrow F \]

\[ \uparrow \{q_0\} \]

\[ \cap \neq \emptyset \]
Forward analysis with antichains

\[ U_1 = U_0 \cup \text{Post}(U_0) \]

\[ U_{i+1} = U_i \cup \text{Post}(U_i) \]

\[ U^* = U^* \cup \text{Post}(U^*) \]

\[ \subseteq \text{-Upward-closed sets are canonically represented by their } \subseteq \text{-minimal elements} \]

\[ \cap \neq \emptyset \]

orders of magnitude faster than BDDs
Discover post-fixpoint using SAT

- A set of sets $\mathcal{S} \subseteq 2^Q$ is a post-fixpoint of $\text{Post}[A]$ if:
  - $\{q_0\} \in \mathcal{S}$
  - $\text{Post}[A](\mathcal{S}) \subseteq \mathcal{S}$
- Problem: find $\mathcal{S}$ such that $\mathcal{S} \cap \mathcal{F} = \emptyset$
- Rely on the *antichain representation* of $\mathcal{S}$
Using SAT to synthesize $S$

- Fix $k$ the number of sets in the antichain
- $X = \{ (q,i) \mid q \in Q \land 1 \leq i \leq k \}$
- any $v : X \rightarrow \{0,1\}$ represent an antichain

\{ q \mid v(q,i)=1 \}  

set nr. $i$ of the antichain
Boolean encoding

• \( S \) is a post-fixpoint of \( \text{Post}[A] \) and \( S \) does not intersect with \( \downarrow F \)

\[
\cap_{i=1}^{k} \cap_{\sigma \in \Sigma} \vee_{j=1}^{k} \cap_{(q,i) \in X} (q,i) \rightarrow \cap_{(q,j) | q \in \delta(q,\sigma)} (q,j)
\]

• \((q_0, 1)\)

\[
\cap_{i=1}^{k} \vee_{q \in F} \neg(q, i)
\]

Check that it is a post fixpoint for \( \text{POST} \)
Boolean encoding

• \( S \) is a post-fixpoint of \( \text{Post}[A] \) and \( S \) does not intersect with \( \downarrow F \)

\[
\bigwedge_{i=1}^{k} \bigwedge_{\sigma \in \Sigma} \bigvee_{j=1}^{k} \bigwedge_{(q,i) \in X} (q, i) \rightarrow \bigwedge_{(q,j) \mid q \in \delta(q,\sigma)} (q, j)
\]

• \((q_0, 1)\)

\[
\bigwedge_{i=1}^{k} \bigvee_{q \in F} \neg(q, i)
\]

Check that initial state of automaton is contained
**Boolean encoding**

- $S$ is a post-fixpoint of $\text{Post}[A]$ and $S$ does not intersect with $\downarrow F$

\[ \bigwedge_{i=1}^{i=k} \bigwedge_{\sigma \in \Sigma} \bigvee_{j=1}^{j=k} \bigwedge_{(q,i) \in X} (q,i) \rightarrow \bigwedge_{(q,j) \mid q \in \delta(q,\sigma)} (q,j) \]

- $(q_0, 1)$

\[ \bigwedge_{i=1}^{i=k} \bigvee_{q \in F} \neg (q, i) \]

Check universality
Boolean encoding

- $\mathcal{S}$ is a post-fixpoint of $\text{Post}[\mathcal{A}]$ and $\mathcal{S}$ does not intersect with $\downarrow F$

  $$\bigwedge_{i=1}^{k} \bigwedge_{\sigma \in \Sigma} \bigvee_{j=1}^{k} \bigwedge_{(q,i) \in X} (q, i) \rightarrow \bigwedge_{(q,j)}$$

- $(q_0, 1)$

- $\bigwedge_{i=k}^{i}$

Similar to template based inductive invariant generation using SMT solvers
Conclusion

- There are several uses of SAT solvers beyond Bounded MC
- SAT can be used to help SMC
- UNSAT Core are important and rich objects, useful for abstraction refinements
- Interpolation pushes the idea further (no more BDDs)
- Direct construction of inductive invariants can be useful too
Pointers to bibliography

• Kenneth L. McMillan: Interpolation and SAT-Based Model Checking. CAV 2003.


• Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu: Symbolic Model Checking without BDDs. TACAS 1999.

• Parosh Aziz Abdulla, Per Bjesse, Niklas Eén: Symbolic Reachability Analysis Based on SAT-Solvers. TACAS 2000.


• Handbook of Model-Checking. Springer. 2015.