Bounded Model Checking and Boolean Satisfiability

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

Evropský sociální fond Praha & EU: Investujeme do vaší budoucnosti

Testing, Bounded Model Checking

Question: Does transition system fulfill property?

Test on finite path t, necessary condition for correctness:

- $t \models \mathbf{G}\ p$: property p holds on every state of t.
- $t \models \mathbf{F}\ p$: if t contains a cycle
  then it reaches a state where p holds.
- $t \models p$ iff p is a state property and
  t has length 0 or $t(0) \models p$.

$Test(\phi, T) \Leftrightarrow$ for all finite paths $t \in T$, $t \models \phi$.

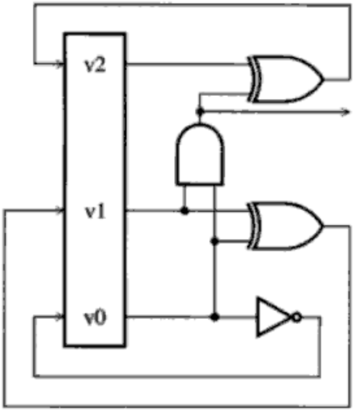
$\mathrm{BMC}(\phi, n) \Leftrightarrow$ for all finite paths t of a certain length n, $t \models \phi$

Problem: While possible for finite systems
explicitely checking all paths usually too inefficient in practice.

Example: 10 states, $n = 10$, $10^{10} = 10000000000$ paths to check!

Efficient Checking of BMC(ϕ, n)

Concrete example: Modulo 8 counter: State: $S \doteq \mathbb{B}^3$



$$v_0' = \neg v_0$$
$$v_1' = v_0 \oplus v_1$$
$$v_2' = (v_0 \wedge v_1) \oplus v_2$$

Example: Digital Circuit

Transition relation:

$$R \doteq \left\{ ((v_0, v_1, v_2), (v_0', v_1', v_2')) \mid \begin{array}{l} (v_0' \Leftrightarrow \neg v_0) \wedge \\ (v_1' \Leftrightarrow v_0 \oplus v_1) \wedge \\ (v_2' \Leftrightarrow (v_0 \wedge v_1) \oplus v_2) \end{array} \right\}$$

Formula in propositional logic (*Boolean formula*)!

Bad news: Checking satisfiability of Boolean formulas is NP-hard

Good news:

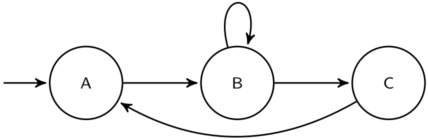
- In practice, algorithms can solve huge problems
- Especially, for satisfiable problems,
  can often find satisfying assignment extremely quickly!

Further Plan:

- Encoding BMC to SAT
- Solving SAT

Boolean Encoding of BMC

Example: transition system



$$(\{A, B, C\}, \{A\}, \left\{ \begin{array}{l} (A, B), \\ (B, B), \\ (B, C), \\ (C, A) \end{array} \right\})$$

Encoding of states:

A	$(\top, \top)$
B	$(\top, \bot)$
C	$(\bot, \top)$
	$(\bot, \bot)$

Equivalent transition system: $(\mathbb{B}^2, \{(\top, \top)\}, \left\{ \begin{array}{l} ((\top, \top), (\top, \bot)), \\ ((\top, \bot), (\top, \bot)), \\ ((\top, \bot), (\bot, \top)), \\ ((\bot, \top), (\top, \top)) \end{array} \right\})$

Boolean Encoding of BMC

$$(\mathbb{B}^2, \{(\top,\top)\}, \left\{\begin{array}{l} ((\top,\top),(\top,\bot)), \\ ((\top,\bot),(\top,\bot)), \\ ((\top,\bot),(\bot,\top)), \\ ((\bot,\top),(\top,\top)) \end{array}\right\})$$

$$\left(\mathbb{B}^2, \left\{(v_0, v_1) \mid v_0 \wedge v_1\right\}, \left\{((v_0, v_1), (v_0', v_1')) \mid \begin{array}{c} [v_0 \wedge v_1 \wedge v_0' \wedge \neg v_1'] \\ \vee \\ [v_0 \wedge \neg v_1 \wedge v_0' \wedge \neg v_1'] \\ \vee \\ [v_0 \wedge \neg v_1 \wedge \neg v_0' \wedge v_1'] \\ \vee \\ [\neg v_0 \wedge v_1 \wedge v_0' \wedge v_1'] \end{array}\right\}\right)$$

Simplified Transition System

$$\left(\mathbb{B}^2, \left\{(v_0, v_1) \mid v_0 \wedge v_1\right\}, \left\{((v_0, v_1), (v_0', v_1')) \mid \begin{array}{c} [v_0 \wedge v_1 \wedge v_0' \wedge \neg v_1'] \\ \vee \\ [v_0 \wedge \neg v_1 \wedge v_0' \wedge \neg v_1'] \\ \vee \\ [v_0 \wedge \neg v_1 \wedge \neg v_0' \wedge v_1'] \\ \vee \\ [\neg v_0 \wedge v_1 \wedge v_0' \wedge v_1'] \end{array} \right\}\right)$$

$$\left(\mathbb{B}^2, \left\{(v_0, v_1) \mid v_0 \wedge v_1\right\}, \left\{((v_0, v_1), (v_0', v_1')) \mid \begin{array}{c} [v_0 \wedge v_0' \wedge \neg v_1'] \\ \vee \\ [v_0 \wedge \neg v_1 \wedge \neg v_0' \wedge v_1'] \\ \vee \\ [\neg v_0 \wedge v_1 \wedge v_0' \wedge v_1'] \end{array} \right\}\right)$$

Arbitary Finite Transition Systems

Every finite transition sytems (S, S_0, R)
can be encoded as a Boolean one:

- Binary encoding of state (needs $\lceil \log_2 |S| \rceil$ variables)
- Set of initial, states, transition relation:
  representation as a Boolean formula

Boolean Satisfiability (SAT)

- Input: Boolean formula
- Output:
  - satisfying assignment, if it exists,
  - `unsat`, if the formula is not satisfiable.

$$\left(\mathbb{B}^2, \left\{(v_0, v_1) \mid v_0 \wedge v_1\right\}, \left\{((v_0, v_1), (v_0', v_1')) \mid \begin{array}{c} [v_0 \wedge v_0' \wedge \neg v_1'] \vee \\ [v_0 \wedge \neg v_1 \wedge \neg v_0' \wedge v_1'] \vee \\ [\neg v_0 \wedge v_1 \wedge v_0' \wedge v_1'] \end{array} \right\}\right)$$

Examples:

- Input: $v_0 \wedge v_1$ (does our system have an initial state?)
- Output: $\{v_0 \mapsto \top, v_1 \mapsto \top\}$ (yes, state A)

- Input: $v_0 \wedge v_1 \wedge \left[\begin{array}{c} [v_0 \wedge v_0' \wedge \neg v_1'] \vee \\ [v_0 \wedge \neg v_1 \wedge \neg v_0' \wedge v_1'] \vee \\ [\neg v_0 \wedge v_1 \wedge v_0' \wedge v_1'] \end{array}\right] \wedge \neg v_0' \wedge v_1'$
  (may our system after one step be in C?)
- Output: `unsat` (no)

Bounded Model Checking via SAT

SAT:

- Input: Boolean formula
- Output:
  - satisfying assignment, if it exists,
  - `unsat`, if the formula is not satisfiable.

BMC($\mathbf{G}\,\mathrm{ok}, n$):

for all finite paths t of length n, $t \models \mathbf{G}\,\mathrm{ok}$

$\neg$BMC($\mathbf{G}\,\mathrm{ok}, n$):

there is a finite path t of length n, $t \not\models \mathbf{G}\,\mathrm{ok}$

there is a finite path t s.t.

not for all k s.t. $0 \leq k \leq n-1$, $t(k) \models \mathrm{ok}$

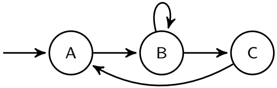
there is a finite path t s.t.

there is a k s.t. $0 \leq k \leq n-1$, $t(k) \not\models \mathrm{ok}$

there is a finite path t s.t.

$t(0) \not\models \mathrm{ok}$ or $\ldots$ or $t(n-1) \not\models \mathrm{ok}$

For above example, $\mathcal{I}(ok) = \{A, B\}$, $n = 3$



there is a finite path t s.t.

$t(0) \not\models$ ok or ... or $t(n-1) \not\models$ ok

there is a finite path t s.t.

$t(0) \notin \{A, B\}$ or ... or $t(n-1) \notin \{A, B\}$

there is a finite path t s.t.

$t(0) \notin \{A, B\}$ or $t(1) \notin \{A, B\}$ or $t(2) \notin \{A, B\}$

Finite path: $((v_0^0, v_1^0), (v_0^1, v_1^1), (v_0^2, v_1^2))$

$$v_0^0 \wedge v_1^0 \wedge \begin{bmatrix} [v_0^0 \wedge v_0^1 \wedge \neg v_1^1] \\ \vee \\ [v_0^0 \wedge \neg v_1^0 \wedge \neg v_0^1 \wedge v_1^1] \\ \vee \\ [\neg v_0^0 \wedge v_1^0 \wedge v_0^1 \wedge v_1^1] \end{bmatrix} \wedge \begin{bmatrix} [v_0^1 \wedge v_0^2 \wedge \neg v_1^2] \\ \vee \\ [v_0^1 \wedge \neg v_1^1 \wedge \neg v_0^2 \wedge v_1^2] \\ \vee \\ [\neg v_0^1 \wedge v_1^1 \wedge v_0^2 \wedge v_1^2] \end{bmatrix} \wedge$$

$$\neg v_0^0 \vee \neg v_0^1 \vee \neg v_0^2$$

Bounded Model Checking via SAT

SAT:

- Input: Boolean formula
- Output:
  - satisfying assignment, if it exists,
  - `unsat`, if the formula is not satisfiable.

$\text{BMC}(\phi, n)$:

for all finite paths t of length n, $t \models \phi$

$\neg\text{BMC}(\phi, n)$:

not for all finite paths t of length n, $t \models \phi$

there is a finite path t of length n, $t \not\models \phi$

Intuition: Work with $\neg\text{BMC}(\phi, n)$

- satisfying assignment $\leftrightarrow$ counter-example (e.g., path to unsafe state)
- `unsat` $\leftrightarrow$ $\text{BMC}(\phi, n)$ holds (no path to unsafe state of length n exists)

Boolean Encoding of Bounded Model Checking

Assumption: Transition system $(\mathbb{B}^k, \{\vec{v} \mid S_o\}, \{(\vec{v}, \vec{v}') \mid T\})$, so

- S_0, T, and ok are Boolean formulas encoding the corresponding sets.
- $\vec{v}$, $\vec{v}'$ are Boolean variables representing the states of the transition system

$\neg BMC(\mathbf{G}\text{ok}, n)$:
there is finite path t s.t.
$t(0) \not\models \text{ok}$ or $\ldots$ or $t(n-1) \not\models \text{ok}$

Representation of finite path t of length n:
k-tuple of Boolean vectors $(\vec{v}^0, \ldots, \vec{v}^{n-1})$

Representation of $\neg BMC(\mathbf{G}\text{ok}, n)$:

$$S_0[\vec{v} \leftarrow \vec{v}^0] \wedge \bigwedge_{i=0}^{n-2} T[\vec{v} \leftarrow \vec{v}^i, \vec{v}' \leftarrow \vec{v}^{i+1}] \wedge \bigvee_{i \in \{0,\ldots,n-1\}} \neg\text{ok}[\vec{v} \leftarrow \vec{v}^i]$$

Symbolic Model Checking

- Use symbols (in our case logical formulas) for representing big sets.
- In the case of infinite state space the only possibility.

Further examples of symbolic representations:

- Representation of convex polyhedra using system of linear inequalities
- Representation of regular languages using regular expressions

SAT Solving

Input: Boolean formula in conjunctive normal form (CNF)

Output:

- satisfying assignment, if it exists,
- `unsat`, if the formula is not satisfiable.

special transformations to CNF, e.g., Tseitin encoding [Prestwich, 2009]

When I studied:

There is no point in trying to solve NP-complete problems,
unless somebody proves P=NP.

Today ...

Formula-SAT

There is a race for more and more efficient SAT solvers
and it is completely common to solve huge SAT problems.

Formula-SAT

Today's solvers only have a few hundred lines of code.

Often open-source

For example: MiniSAT `http://minisat.se`

Yearly SAT competition `http://www.satcompetition.org`

But: Solver still exponential in worst case,
question $P = NP$ still open!

How Do SAT Solvers Work?

Example: $[\neg P \vee Q \vee R] \wedge [\neg Q \vee R] \wedge [\neg Q \vee \neg R] \wedge [P \vee Q]$

Terminology:

- *Literal*: Boolean variable or its negation
- *Clause*: Disjunction of literals (we do not distinguish clauses with different order of same literals)

We all know a very simple algorithm!

Truth Table

P	Q	R	$\neg P \vee Q \vee R$	$\neg Q \vee R$	$\neg Q \vee \neg R$	$P \vee Q$	
$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\bot$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\bot$

In general: $2^{|V|}$ assignments

In general, we do not have a chance with this

Improvement?

The table has some structure! All rows in the first half ...

Structured Truth Table

P	Q	R	$\neg P \vee Q \vee R$	$\neg Q \vee R$	$\neg Q \vee \neg R$	$P \vee Q$	
$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$
		$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$
	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
		$\top$	$\top$	$\top$	$\bot$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$
		$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
		$\top$	$\top$	$\top$	$\bot$	$\top$	$\bot$

Simplification after assignment of individual variables.

After Assignment to First Variable

<table>
<tr><th>P</th><th>Q</th><th>R</th><th>$\neg P \vee Q \vee R$</th><th>$\neg Q \vee R$</th><th>$\neg Q \vee \neg R$</th><th>$P \vee Q$</th><th></th></tr>
<tr><td rowspan="4">$\bot$</td><td rowspan="2">$\bot$</td><td>$\bot$</td><td rowspan="4">$\top$</td><td rowspan="8">$\neg Q \vee R$</td><td rowspan="8">$\neg Q \vee \neg R$</td><td rowspan="4">Q</td><td rowspan="8"></td></tr>
<tr><td>$\top$</td></tr>
<tr><td rowspan="2">$\top$</td><td>$\bot$</td></tr>
<tr><td>$\top$</td></tr>
<tr><td rowspan="4">$\top$</td><td rowspan="2">$\bot$</td><td>$\bot$</td><td rowspan="4">$Q \vee R$</td><td rowspan="4">$\top$</td></tr>
<tr><td>$\top$</td></tr>
<tr><td rowspan="2">$\top$</td><td>$\bot$</td></tr>
<tr><td>$\top$</td></tr>
</table>

After Assignment to Second Variable

P	Q	R	$\neg P \vee Q \vee R$	$\neg Q \vee R$	$\neg Q \vee \neg R$	$P \vee Q$	
$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	
		$\top$					
	$\top$	$\bot$		R	$\neg R$	$\top$	
		$\top$					
$\top$	$\bot$	$\bot$	R	$\top$	$\top$	$\top$	
		$\top$					
	$\top$	$\bot$	$\top$	R	$\neg R$		
		$\top$					

After Assignment to Third Variable

<table>
<tr><th>P</th><th>Q</th><th>R</th><th>$\neg P \vee Q \vee R$</th><th>$\neg Q \vee R$</th><th>$\neg Q \vee \neg R$</th><th>$P \vee Q$</th><th></th></tr>
<tr><td rowspan="4">$\bot$</td><td rowspan="2">$\bot$</td><td>$\bot$</td><td rowspan="4">$\top$</td><td rowspan="2">$\top$</td><td rowspan="2">$\top$</td><td rowspan="2">$\bot$</td><td rowspan="8"></td></tr>
<tr><td>$\top$</td></tr>
<tr><td rowspan="2">$\top$</td><td>$\bot$</td><td>$\bot$</td><td>$\top$</td><td rowspan="2">$\top$</td></tr>
<tr><td>$\top$</td><td>$\top$</td><td>$\bot$</td></tr>
<tr><td rowspan="4">$\top$</td><td rowspan="2">$\bot$</td><td>$\bot$</td><td>$\bot$</td><td rowspan="2">$\top$</td><td rowspan="2">$\top$</td><td rowspan="4">$\top$</td></tr>
<tr><td>$\top$</td><td>$\top$</td></tr>
<tr><td rowspan="2">$\top$</td><td>$\bot$</td><td rowspan="2">$\top$</td><td>$\bot$</td><td>$\top$</td></tr>
<tr><td>$\top$</td><td>$\top$</td><td>$\bot$</td></tr>
</table>

breadth-first search

Problem: We have several copies of the formula in memory.

Instead of breadth-first search? depth-first, backtracking

SAT Solving

Version that only returns $\top$ (for satisfiable input), $\bot$ (unsatisfiable input)
(i.e., we do not yet compute a satisfiable assignment):

SAT(ϕ)=
- **let** $\phi' =$ simplify(ϕ)
- **if** ϕ' is a Boolean constant **then return** ϕ'
- **let** v be a free variable of ϕ'
- **return** SAT($\phi'[v \leftarrow \bot]$) or SAT($\phi'[v \leftarrow \top]$) // short-circuit eval

For the original specification we only have to remember the substitutions.

Which variable and branch ($\bot/\top$) to choose first? important heuristic

Which simplifications?

Basic Simplifications

Assumption: input in conjunctive normal form

- $\neg\top \rightsquigarrow \bot$
- $\neg\bot \rightsquigarrow \top$
- $\dots \vee \top \vee \dots \rightsquigarrow \top$
- $\dots \vee \bot \vee \dots \rightsquigarrow \dots \vee \dots$
- $\dots \wedge \top \wedge \dots \rightsquigarrow \dots \wedge \dots$
- $\dots \wedge \bot \wedge \dots \rightsquigarrow \bot$

Example:

SAT(ϕ)=

  let $\phi' = \text{simplify}(\phi)$
  if ϕ' is a Boolean constant **then return** ϕ'
  let v be a free variable of ϕ'
  return SAT($\phi'[v \leftarrow \bot]$) or SAT($\phi'[v \leftarrow \top]$) // short-circuit eval

choice of variables in alphabetical order, first $\bot$ then $\top$

call	simplification	further action
SAT($[P \vee Q] \wedge [P \vee \neg Q \vee R]$)		$P \leftarrow \bot$
SAT($[\bot \vee Q] \wedge [\bot \vee \neg Q \vee R]$)	$Q \wedge [\neg Q \vee R]$	$Q \leftarrow \bot$
SAT($\bot \wedge [\neg\bot \vee R]$)	$\bot$	backtrack, $Q \leftarrow \top$
SAT($\top \wedge [\neg\top \vee R]$)	R	$R \leftarrow \bot$
SAT($\bot$)		backtrack, $R \leftarrow \top$
SAT($\top$)		**return** $\top$

the choice $P \leftarrow \top$ would have resulted in immediate success!

More Simplifications

$$[\neg Q \vee R] \wedge [\neg Q \vee \neg R] \wedge Q \quad ???$$

Observation:

Sometimes a clause appears that contains only one literal.

In this case we can immediately assign a value to corresponding variable. (*unit propagation*)

$$[\neg Q \vee R \vee S \vee V] \wedge [\neg Q \vee \neg R \vee S \vee \neg V] \wedge [Q \vee V] \quad ???$$

A variable assignment satisfies a conjunction of clauses iff it satisfies every clause.

If a variable occurs only positively or only negatively, we can immediately assign a truth value that evaluates all corresponding clauses to $\top$ (*pure literal elimination*)

Resulting Algorithm

algorithm SAT()+
basic simplifications+unit propagation+pure literal elimination:

Davis-Putnam-Logemann-Loveland algorithm (DPLL), 1962

Example:

$$SAT([\neg P \vee Q \vee R] \wedge [\neg Q \vee R] \wedge [\neg Q \vee \neg R] \wedge [P \vee Q])$$

No simplification, recursive call with $[P \leftarrow \bot]$

$$SAT([\neg\bot \vee Q \vee R] \wedge [\neg Q \vee R] \wedge [\neg Q \vee \neg R] \wedge [\bot \vee Q])$$

Basic simplifications:

$$[\neg Q \vee R] \wedge [\neg Q \vee \neg R] \wedge Q$$

Unit propagation $[Q \leftarrow \top]$

$$[\neg\top \vee R] \wedge [\neg\top \vee \neg R] \wedge \top$$

Basic simplification

$$R \wedge \neg R$$

Unit propagation $[R \leftarrow \top]$:

$$\top \wedge \neg\top$$

Basic simplification

$$\bot$$

backtrack, recursive call with $[P \leftarrow \top]$

Example:

$$SAT([\neg\top \vee Q \vee R] \wedge [\neg Q \vee R] \wedge [\neg Q \vee \neg R] \wedge [\top \vee Q])$$

Basic simplifications:

$$[Q \vee R] \wedge [\neg Q \vee R] \wedge [\neg Q \vee \neg R]$$

recursive call with $[Q \leftarrow \bot]$

$$SAT([\bot \vee R] \wedge [\neg\bot \vee R] \wedge [\neg\bot \vee \neg R])$$

Basic simplification

$$R$$

Unit propagation $[R \leftarrow \top]$:

$$\top$$

Shortcut evaluation

So: input formula satisfiable by assignment $\{P \mapsto \top, Q \mapsto \bot, R \mapsto \top\}$

Further Improvements

Modern SAT solvers contain additional improvements, for example
conflict driven clause learning (CDCL):

c_1: $v_{223} \vee \neg v_{355}$
c_2: $v_{343} \vee v_{355} \vee v_{634}$
c_3: $v_{343} \vee v_{355} \vee \neg v_{634}$
...
c_r:

After $v_{223} \leftarrow \perp$:

c_2: $v_{343} \vee v_{634}$
c_3: $v_{343} \vee \neg v_{634}$
...
c_r:

After $v_{343} \leftarrow \perp$: backtrack

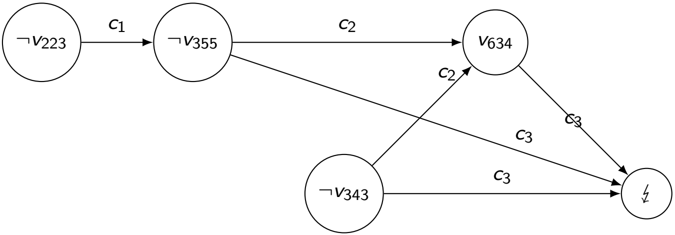
Same situation for all $2^{\cdots}$ assignments with $v_{223} = \perp, v_{343} = \perp$

Conflict Driven Clause Learning

c_1: $v_{223} \vee \neg v_{355}$

c_2: $v_{343} \vee v_{355} \vee v_{634}$

c_3: $v_{343} \vee v_{355} \vee \neg v_{634}$



How to avoid?

Add clause $\neg [\neg v_{223} \wedge \neg v_{343}]$, that is $v_{223} \vee v_{343}$
(*conflict clause*)

Implication Graph

- Vertices: literals, special vertex $\lightning$
- Edges: clauses (from original problem, without simplification)

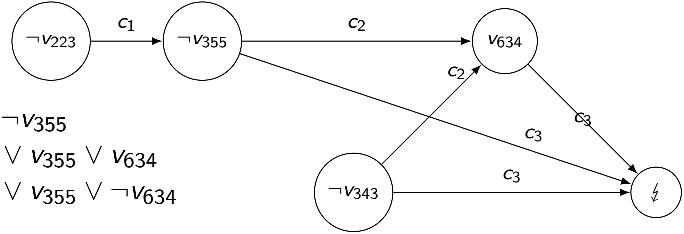
s.t. for the SAT algorithm execution, in the corresponding implication graph,

- each assignment of $\top$ to a variable v corresponds to a vertex v,
- each assignment of $\bot$ to a variable v corresponds to a vertex $\neg v$,
- each unit propagation of a literal l that
  belongs to a clause $l_1 \vee \dots \vee l_k \vee l$ of the original formula,
  corresponds, for each $i \in \{1, \dots, k\}$, to
  an edge from $\neg l_i$ to l labeled with $l_1 \vee \dots \vee l_k \vee l$, and
- each derivation of $\bot$ that
  belongs to a clause $l_1 \vee \dots \vee l_k$ of the original formula,
  corresponds, for each $i \in \{1, \dots, k\}$, to
  an edge from $\neg l_i$ to $\lightning$ labeled with $l_1 \vee \dots \vee l_k$.

Here, double negations cancel out.

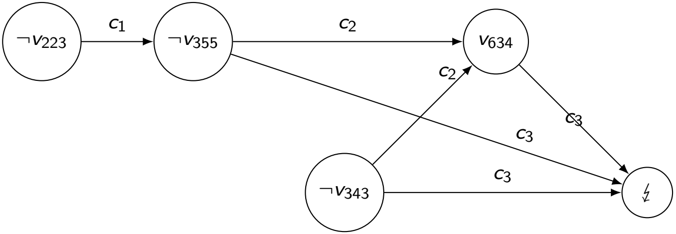
Example

- each assignment of $\top$ to a variable v corresponds to a vertex v,
- each assignment of $\bot$ to a variable v corresponds to an vertex $\neg v$,
- each unit propagation of a literal l that
  belongs to a clause $l_1 \vee \cdots \vee l_k \vee l$ of the original formula,
  corresponds, for each $i \in \{1, \ldots, k\}$, to
  an edge from $\neg l_i$ to l labeled with $l_1 \vee \cdots \vee l_k \vee l$, and
- each derivation of $\bot$ that
  belongs to a clause $l_1 \vee \cdots \vee l_k$ of the original formula,
  corresponds, for each $i \in \{1, \ldots, k\}$, to
  an edge from $\neg l_i$ to $\lightning$ labeled with $l_1 \vee \cdots \vee l_k$.



c_1 :	$v_{223} \vee$			$\neg v_{355}$			
c_2 :		v_{343}	$\vee$	v_{355}	$\vee$	v_{634}	
c_3 :		v_{343}	$\vee$	v_{355}	$\vee$	$\neg v_{634}$	

Implication Graph in Algorithm



SAT(ϕ)=

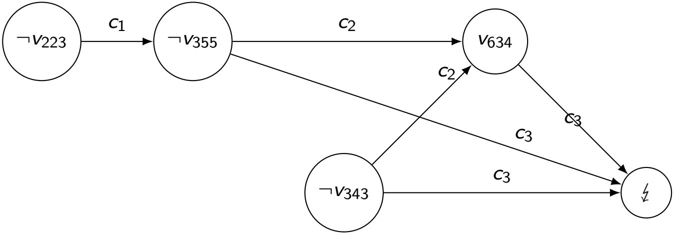
let $\phi' = \text{simplify}(\phi)$ // *add internal vertices,* $\lightning$*, edges*

if ϕ' is a Boolean constant **then return** ϕ' // *backtrack*

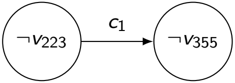
let v be a free variable of ϕ'

return SAT($\phi'[v \leftarrow \bot]$) or SAT($\phi'[v \leftarrow \top]$) // *add source vertex*

Backtracking

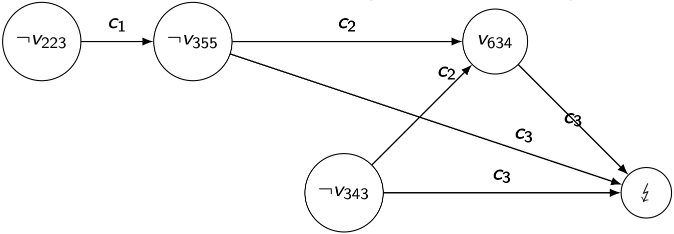


assignment $v_{343} \leftarrow \bot$ resulted in conflict, backtrack to



next assignment $v_{343} \leftarrow \top$

Conflict Clause from Implication Graph



$\neg v_{634} \vee v_{355} \vee v_{343}$, $v_{355} \vee v_{343}$, $v_{223} \vee v_{343}$

cc is a *conflict clause* iff there is a subgraph C of the implication graph s.t.

- C contains $\lightning$
- there is a clause c s.t.
  C contains all edges labeled with c and leading to $\lightning$,
- for each internal vertex l of C
  there is a clause c s.t.
  C contains all edges labeled with c and leading to l, and
- the negation of each root of C is a literal of *cc*.

Implementation Techniques

let $\phi' = \text{simplify}(\phi)$
if ϕ' is a Boolean constant **then return** ϕ'
let v be a free variable of ϕ'
return $\text{SAT}(\phi'[v \leftarrow \bot])$ or $\text{SAT}(\phi'[v \leftarrow \top])$

Observation: We do

- a substitution $\phi'[v \leftarrow \bot]$,
- backtracking back to ϕ',
- a further substitution $\phi'[v \leftarrow \top]$.

So we have to store copies in memory.

Better: Do not change formulas
(neither by substitution, nor by simplification),
just store newly computed information.

Instead of recursion, loop.

Local Search

Original used for continuous problems

- random choice of assignment
- gradual change such that the number of satisfied clauses increases

GSAT(ϕ):

for $i \leftarrow 1$ **to** MAXTRIES **do**
    $V \leftarrow$ random assignment
    for $i \leftarrow 1$ **to** MAXFLIPS **do**
        if V satisfies ϕ **then return** $\top$
        else
            Flip any variable in ϕ that results in
                greatest decrease in the number of unsatisfied clauses

Incomplete method, cannot prove unsatisfiability.

But can often find satisfying assignment fast.

Various variants, for example Walksat
(`http://www.cs.rochester.edu/u/kautz/walksat/`)

SAT modulo theory

Extended task:

We do not only allow Boolean variables,
but also variables with domain $\mathbb{N}$, $\mathbb{R}$, arrays, lists, etc.
an corresponding constraints (e.g., $2x + 3y^2 \leq 0$)

see MI-FME

Example: [Fränzle, Herde, Ratschan, Schubert, and Teige, 2007]
(first SAT solvers world-wide that was able to handle non-linear equalities and inequalities over the real numbers).

Improvements for Bounded Model Checking

BMC($\mathbf{G}p, n$) :

$$S_0(\vec{v}^0) \wedge \bigwedge_{i=0}^{n-2} T(\vec{v}^i, \vec{v}^{i+1}) \wedge \bigvee_{i=0}^{n-1} \neg p(\vec{v}^i)$$

unsat of $\bigwedge_{i=s}^{t} \hat{T}(\vec{v}^i, \vec{v}^{i+1})$ also holds for shifted version

$$\bigwedge_{i=s+1}^{t+1} \hat{T}(\vec{v}^i, \vec{v}^{i+1}), \bigwedge_{i=s+2}^{t+2} \hat{T}(\vec{v}^i, \vec{v}^{i+1}) \dots$$

So: shifted conflict clauses

Check for $\mathrm{BMC}(\mathbf{G}\ \mathbf{p}, 0), \mathrm{BMC}(\mathbf{G}\ \mathbf{p}, 1), \mathrm{BMC}(\mathbf{G}\ \mathbf{p}, 2), \dots$

Re-use information: unsat implied by
$\hat{S}_0(\vec{v}^0) \wedge \bigwedge_{i=0}^{k} \hat{T}(\vec{v}^i, \vec{v}^{i+1})$ unsat also holds for higher k

So: re-use conflict clauses

General Usage of SAT Algorithms

- Basic NP-complete problem: Other NP-complete problems are translated to SAT, often more efficient result than more specific algorithms
- Main workhorses for discrete reasoning:
  - MILP
  - Constraint programming
  - SAT

Example applications:

- Intel, AMD, ... commonly use SAT solvers to
  check the correctness of their chips
  `https://www.research.ibm.com/haifa/projects/verification/Formal_Methods-Home/index.shtml`
- Microsoft uses SAT for test generation (see MI-FME)
- Resolution of Linux package dependencies (ZYpp)
- Gene analysis [Lynce and Marques-Silva, 2006]

Literature I

Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009. ISBN 978-1-58603-929-5.

M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT—Journal on Satisfiability, Boolean Modeling and Computation, Special Issue on SAT/CP Integration*, 1:209–236, 2007.

Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2008.

Inês Lynce and João Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 104. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Literature II

Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, August 2009. ISSN 0001-0782. doi: 10.1145/1536616.1536637. URL `http://doi.acm.org/10.1145/1536616.1536637`.

Steven Prestwich. CNF encodings. In Biere et al. [2009], pages 75–97. ISBN 978-1-58603-929-5.