Chapter 5

Sat-based & Bounded model checking

Course "Model checking"
Martin Steffen
Autumn 2021

Section

Introduction

Chapter 5 “Sat-based & Bounded model checking”
Course “Model checking”
Martin Steffen
Autumn 2021

Model checking

$$S \models^{?} \varphi$$

- origin [7][1] & [11]
- S (model of the) system,
- φ: formula in a suitable logic
  - LTL
  - CTL, CTL*, modal μ-calculus
  - ...
- ultimately a fancy “graph exploration problem” (with *big* graphs)

[1] the conference was 1981, the book was published 1982

Advantages of MC

- no proofs, “push button”
- diagnostic counterexamples
- logics used for MC can express many concurrency problems

Main “disadvantage”

- state space explosion problem (aka state explosion problem)
- problem “solution” space grows *exponential* is the problem “description” space
  - notably reachable state space exponential in the number of processes

The 4 big breakthroughs combatting the SSEP

Apart from

- advances in data structures,
- software engineering,
- tricks, optimizations, heuristics and
- general advances in processing power/memory.

Clarke identifies the following

"big 4" breakthroughs

1. symbolic techniques (notably using BDDs)[2]
2. partial order reduction
3. bounded model checking
4. CEGAR, localisation reduction [9] [4] [3]

[2] See later presentations

Section

SAT solving and SMT

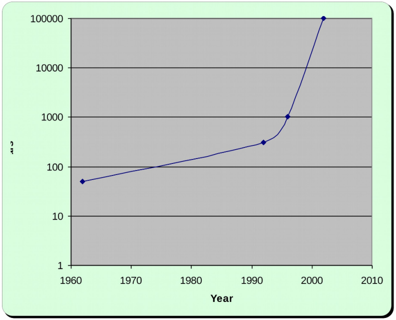
Chapter 5 “Sat-based & Bounded model checking”
Course “Model checking”
Martin Steffen
Autumn 2021

SAT

- (boolean) satisfiability
- famous, prototypical NP-complete problem
-

SAT solver progress

- highly competitive field
- yearly "SAT-competition"[3]



taken from [6]

[3] http://www.satcompetition.org/

Bounded model checking

- Origin: [1] (see also [2])

BMC starting point

Leverage sat-solving, a powerful a successful technique, to do model checking

Cf.: Symbolic model checking and BDDs

- See separate presentation
- successful technique
- used (most prominently for HW) in industrial uses of MC
- Two ingredients of SMC
  - operating symbolically on representation of sets of states
  - use *BDDs* (= specific kind of graph representation of *boolean* functions) to represent and operate on them
- like SMC/BDD-based MC: BMC based on "boolean encodings"

Bad news: the MC problem/reachability is *not* a SAT problem :-(

- MC *here:*[4]
  - models are kind of transition systems/Kripke structures ...
  - spec's are "temporal logic" formulas

solving an MC problem

It all boils down to some form of fancy graph reachability

- "reachability", however:
  - a form of "fixpoint" calculation[5]
  - fixpoints are emphatically not part of boolean logic.[6]

[4] The term "model checking", i.e., solving $M \models^? \varphi$ can be applied in different settings as well. A boolean assignment can be seen as *model* of a propositional formula, for instance. That *is* of course a SAT problem. But we are interested transition systems satisfying a TL formula.

[5] see also the presentation about μ-calculus.

[6] They are not even part of first-order logic. Implicitly they are part in temporal logics, though (eventually, until etc.)

Good news: *bounded* MC can be seen as SAT :-)

less ambitious goal

Can I find an error (conterexample) in the behavior of the system considering up-to k steps from the initial states

- price to pay: no more "verification"[7]
- bug-hunting
- simple core idea

[7] but MC is typically verification of a model/abstraction anyhow and/or verification up until the MC runs out of time/memory.

LTL and “existential” LTL

- remember: LTL (linear time temporal logic) and definition of

$$S \models \varphi$$

- φ must hold for all paths of S
- If $S \not\models \varphi$ (error), then exists a paths π such that $\pi \not\models \varphi$

For explicitness’ sake

path quantifiers[8]

$$\forall\varphi \quad \text{and} \quad \exists\varphi$$

- assume NNF

[8] one single quantifier as prefix to an LTL formula.

Terminology: witnesses

counterexample for

$$S \models \Box p \quad \text{corresponds to} \quad S \models \forall \Box p$$

corresponds to the question if there exists a witness[9]

$$\Diamond \neg p$$

- Goal: find finite (fixed bound) prefixes as witness to an existential model checking problem (LTL)
- conceptually easy if original $\forall\varphi$ is a safety prop.
- liveness? witness for $\exists\Box$?

[9] in logics in general, a witness is a thing (here a path) that gives (constructive) evidence to an existential formula

Terminology: witnesses

counterexample for

$$S \models \Box p \quad \text{corresponds to} \quad S \models \forall\Box p$$

corresponds to the question if there exists a witness[9]

$$\Diamond\neg p$$

- Goal: find finite (fixed bound) prefixes as witness to an existential model checking problem (LTL)
- conceptually easy if original $\forall\varphi$ is a safety prop.
- liveness? witness for $\exists\Box$? $\Rightarrow$ loops

[9] in logics in general, a witness is a thing (here a path) that gives (constructive) evidence to an existential formula

Paths with and without loops

No loop

$$\bullet \longrightarrow \bullet_{s_i} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet_{s_k}$$

- only prefix with back loop can be witness for $\Box p$

(k, l)-loop

$$\begin{tikzcd}
\bullet_{s_0} \arrow[r] & \bullet \arrow[r] & \bullet_{s_l} \arrow[r] & \bullet \arrow[r] & \bullet_{s_i} \arrow[r] & \bullet_{s_k} \arrow[lll, bend left]
\end{tikzcd}$$

Loops

Given: TS/Kripke-structure. transition relation $\rightarrow$.

Definition

Assume $l \leq k$. A path π is a (k,l)-loop if $\pi_k \rightarrow \pi_l$ and

$$\pi = u \cdot v^\omega \tag{1}$$

with

$$u = \pi_0 \dots \pi_{l-1} \quad \text{and} \quad v = \pi_l \dots \pi_k$$

A path π is a k-loop if there exists an l with $0 \leq l \leq k$ s.t. π is a (k,l)-loop

- remember: paths π are (infinite) sequences of "states" (worlds)
- loops here is about those states (not "edges" of the picture)

Bounded semantics

- remember the “normal” semantics of LTL from before, relating formulas and paths
- $[\![\varphi]\!]$ or $\pi \models \varphi$
- now: the new “looping paths” (k-loops) as basis for bounded semantics, i.e., basis for BMC
- note: “finite” prefixes (loops) can give information for infinite paths, thus serve as witnesses
- boundes semantics for path

  with loop: “unchanged”

  without loop: be aware of the cut-off and be pessimistic

Bounded semantics: for loops

Definition (Bounded semantics: with lasso)

Let π be a k-loop. A formula φ is *valid* along π *with bound* k, written

$$\pi \models_k \varphi \ ,$$

iff $\pi \models \varphi$.

Bounded semantics: without loops

Definition

Let π be a path which is *not* a k-loop. Then an LTL formula φ is *valid along* π *with bound* k, written

$$\pi \models_k \varphi \ ,$$

iff $\pi \models_k^0 \varphi$, given below.

- earlier $\pi \models \varphi$, corresponding here to $\models^0$
- k is treated as "cut-off":
- what comes *afterward*: unknown
- if in doubt: "false", i.e., the path is not valid/does not satisfy the formula in the bounded manner
  - for $\bigcirc$: don't "look" beyond k
  - for $\Box$: be pessimistic
  - for $\Diamond$: positive answer at least possible within the bound

Bounded semantics: without loops ($\models^i_k$)

Definition (Bounded semantics: without lasso)

$$
\begin{array}{lll}
\pi \models^i_k p & \text{iff} & p \in L(\pi_i) \\
\pi \models^i_k \neg p & \text{iff} & p \notin L(\pi_i) \\
\pi \models^i_k \varphi_1 \wedge \varphi_2 & \text{iff} & \pi \models^i_k \varphi_1 \quad \text{and} \quad \pi \models^i_k \varphi_2 \\
\pi \models^i_k \varphi_1 \vee \varphi_2 & \text{iff} & \pi \models^i_k \varphi_1 \quad \text{or} \quad \pi \models^i_k \varphi_2 \\
 & & \\
\pi \models^i_k \Box\varphi & & \text{is always false} \\
\pi \models^i_k \Diamond\varphi & \text{iff} & \exists j. i \leq j \leq k. \quad \pi \models^j_k \varphi \\
\pi \models^i_k \bigcirc\varphi & \text{iff} & i < k \quad \text{and} \quad \pi \models^{i+1}_k \varphi \\
\pi \models^i_k \varphi_1 \; U \; \varphi_2 & \text{iff} & \exists j, i \leq j \leq k. \pi \models^j_k \varphi_2 \text{ and } \forall n, i \leq n < j. \pi \models^n_k \varphi_1 \\
\pi \models^i_k \varphi_1 \; R \; \varphi_2 & \text{iff} & \exists j, i \leq j \leq k. \pi \models^j_k \varphi_1 \text{ and } \forall n, i \leq n < j. \pi \models^n_k \varphi_2
\end{array}
$$

Bounded $\rightarrow$ unbounded semantics

- Note, the connection is done for existential LTL (formulas of the form $\exists\varphi$, not like $\forall\varphi$)
- unbounded semantics as limit of the bounded ones (for all/arbitrary bounds k)

Lemma (Easy direction (per path))

$$\pi \models_k \varphi \quad \textit{implies} \quad \pi \models \varphi$$

Lemma (For TSs/KSs)

$$S \models \exists\varphi \quad \textit{implies} \quad S \models_k \exists\varphi \qquad \textit{for some } k \geq 0$$

Theorem

$$S \models \exists\varphi \quad \textit{iff} \quad S \models_k \exists\varphi \qquad \textit{for some } k \geq 0$$

Section

Reducing bounded model checking to SAT

Chapter 5 “Sat-based & Bounded model checking”
Course “Model checking”
Martin Steffen
Autumn 2021

BMC via SAT

- so far:
  - definition of the bounded MC problem
  - we convinced ourself: BMC approximates MC (at least for existential path formulas)
- Now: reduce to sat-solving

Goal

$[\![S, \varphi]\!]_k$ is *satisfiable* iff π is a witness for φ

  - sat -problems: formula with (propositional) variables
- encoding given in 3 parts. given k
  1. valid initial path for S and
  2. satisfaction of formula if
     - there's a loop or
     - there's no loop
- remember symbolic CTL model checking

Kripke-structure/transition system

Definition (Kripke structure)

A Kripke structure or transition system is a tuple $(S, I, \rightarrow, V)$ where S is the set of states,$I \subseteq S$ the set of initial states, $\rightarrow \subseteq\ S \times S$ the transition relation, and $V : S \rightarrow 2^P$ the *valuation* function (aka. (state) labelling function).

- transition relation: a predicate:[10] $\rightarrow : S^2 \rightarrow \mathbb{B}$
- initial states: a predicate $I\ :\ S \rightarrow \mathbb{B}$

[10] [2] write $T(s_1, s_2)$ for our infix relational notation $s_1 \rightarrow s_2$, where T is the transition relation predicate.

1^{st} component: Translating S

- remember transition system/Kripke stuctures S
  - states s_i. Consider s_i as *variables*
  - transition relation: as predicate $T(s_k, s_l)$, we write still infix $s_k \to s_l$
- unfolding of the transition relation

$$[\![T]\!]_k \triangleq I(s_0) \wedge \bigwedge_{i=0}^{k-1} s_i \to s_{i+1} \tag{2}$$

- remember in CTL how we encoded $S \times S$
- states in KS: *propositional variables* s_k

Loop condition

- Remember the def. of (k, l)-loop

$$\begin{tikzcd} \bullet_{s_0} \arrow[r] & \bullet \arrow[r] & \bullet_{s_l} \arrow[r] & \bullet \arrow[r] & \bullet_{s_i} \arrow[r] & \bullet_{s_k} \arrow[lll, bend left] \end{tikzcd}$$

- simple abbreviation

$$_l L_k \triangleq s_k \rightarrow s_l$$

- loop condition holds[11] iff there is a back loop from a state s_k back to a previous state s_l (which can be s_k)

Definition (Loop condition)

$$L_k \triangleq \bigvee_{l=0}^{k} {}_l L_k$$

[11] resp. it will hold when applied to a path consisting of a sequence of states s_i, which are considered as propositional variables, as said. the word "back" makes sense only if one interprets the variables to be "in a sequence".

Successor in a loop

a rather unsurprising definition: define "successor"

$succ(i)$ of i in a (k, l)-loop as

- $succ(i) = i + 1$ for $i < k$
- $succ(i) = l$ for k

2^{nd} component: translating formula with a loop

propositional part: boring

$$
\begin{array}{rcl}
{}_l[\![p]\!]_k^i & \triangleq & p(s_i) \\
{}_l[\![\neg p]\!]_k^i & \triangleq & \neg p(s_i) \\
{}_l[\![\varphi_1 \wedge \varphi_2]\!]_k^i & \triangleq & {}_l[\![\varphi_1]\!]_k^i \wedge {}_l[\![\varphi_2]\!]_k^i \\
{}_l[\![\varphi_1 \vee \varphi_2]\!]_k^i & \triangleq & {}_l[\![\varphi_1]\!]_k^i \vee {}_l[\![\varphi_2]\!]_k^i
\end{array}
$$

Cont'd

Actually straightforward

- loop $\rightarrow$ no cut-off $\rightarrow$ "standard semantics"
- remember *unrolling* of fixpoints[12]

temporal part: a bit more interesting

$$
\begin{array}{rcl}
{}_l[\![\Box\varphi]\!]_k^i & \triangleq & {}_l[\![\varphi]\!]_k^i \wedge {}_l[\![\Box\varphi]\!]_k^{succ(i)} \\
{}_l[\![\Diamond\varphi]\!]_k^i & \triangleq & {}_l[\![\varphi]\!]_k^i \vee {}_l[\![\Diamond\varphi]\!]_k^{succ(i)} \\
{}_l[\![\bigcirc\varphi]\!]_k^i & \triangleq & {}_l[\![\varphi]\!]_k^{succ(i)} \\
{}_l[\![\varphi_1 \ U \ \varphi_2]\!]_k^i & \triangleq & {}_l[\![\varphi_1]\!]_k^i \vee {}_l[\![\varphi_1 \ U \ \varphi_2]\!]_k^{succ(i)} \\
{}_l[\![\varphi_1 \ R \ \varphi_2]\!]_k^i & \triangleq & {}_l[\![\varphi_2]\!]_k^i \wedge {}_l[\![\varphi_1 \ R \ \varphi_2]\!]_k^{succ(i)}
\end{array}
$$

[12] Cf. also the presentation about the μ-calculus. Also in the construction of the Büchi-automaton from an LTL formula, that

Translation without a loop

- same principles
- "index" l not needed
- instead of the more complex $succ(i)$: simply $i + 1$.
- otherwise: the definition stays "the same")

3^{rd} component: translating formula without a loop

Inductive case $\forall i \leq k$:

propositional part: boring again

$$
\begin{array}{rcl}
[\![p]\!]_k^i & \triangleq & p(s_i) \\
[\![\neg p]\!]_k^i & \triangleq & \neg p(s_i) \\
[\![\varphi_1 \wedge \varphi_2]\!]_k^i & \triangleq & [\![\varphi_1]\!]_k^i \wedge [\![\varphi_2]\!]_k^i \\
[\![\varphi_1 \vee \varphi_2]\!]_k^i & \triangleq & [\![\varphi_1]\!]_k^i \vee [\![\varphi_2]\!]_k^i
\end{array}
$$

Loop-case (cont'd)

Inductive case $\forall i \leq k$:

temporal part: a bit more interesting

$$
\begin{array}{rcl}
[\![\Box\varphi]\!]_k^i & \triangleq & [\![\varphi]\!]_k^i \wedge [\![\Box\varphi]\!]_k^{i+1} \\
[\![\Diamond\varphi]\!]_k^i & \triangleq & [\![\varphi]\!]_k^i \vee [\![\Diamond\varphi]\!]_k^{i+1} \\
[\![\bigcirc\varphi]\!]_k^i & \triangleq & [\![\varphi]\!]_k^{i+1} \\
[\![\varphi_1 \; U \; \varphi_2]\!]_k^i & \triangleq & [\![\varphi_1]\!]_k^i \vee [\![\varphi_1 \; U \; \varphi_2]\!]_k^{i+1} \\
[\![\varphi_1 \; R \; \varphi_2]\!]_k^i & \triangleq & [\![\varphi_2]\!]_k^i \wedge [\![\varphi_1 \; R \; \varphi_2]\!]_k^{i+1}
\end{array}
$$

- base case: $[\![\varphi]\!]_k^{k+1} \triangleq \mathit{false}$

Putting it together

$$
\begin{aligned}
[\![S,\varphi]\!]_k \triangleq\ & [\![S]\!]_k \wedge \\
& (\ (\neg L_k \wedge [\![\varphi]\!]_k^0) \\
& \quad \vee \quad (\bigvee_{l=0}^{k} ({}_l L_k \wedge\ {}_l[\![\varphi]\!]_k^0) \quad)
\end{aligned}
\tag{3}
$$

Theorem

$$[\![S,\varphi]\!]_k \textit{satisfiable} \quad \textit{iff} \quad S \models_k \exists\varphi\ .$$

Further info

- The technical slides here recap parts of the journal article [2] by the inventors of BMC
- BMC for software [8]
- Survey [10]

References I

Bibliography

[1] Biere, A., Cimatti, A., Clarke, E. M., Fujita, M., and Zhu, Y. (2009). Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of DAC'09: Design Automation Conference*, pages 317–320. ACM.

[2] Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., and Zhu, Y. (2003). Bounded model checking. *Advances in Computers*, 58(11):117–148.

[3] Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2000). Counterexample-guided abstraction refinement. In Emerson, E. A. and Sistla, A. P., editors, *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV '00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer Verlag.

[4] Clarke, E. C., Kurshan, R. P., and Veith, H. (2010). The localization reduction and counter-examble guided abstraction refinement. In Manna, Z. and Peled, D., editors, *Pnueli Festschrift*, volume 6200 of *Lecture Notes in Computer Science*, pages 61–71. Springer Verlag.

[5] Clarke, E. M. (2008). Model checking – my 27-year quest to overcome the state explosion problem. In Cervesato, I., Veith, H., and Voronkov, A., editors, *Logic for Programming, Artificial Intelligence, and Reasoning: 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, Lecture Notes in Artificial Intelligence, pages 182–182. Springer Verlag.

[6] Clarke, E. M. (2017). SAT-based bounded and unbounded model checking. Available electronically on the net. Data of publication unknown.

[7] Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronisation skeletons using branching time temporal logic specifications. In Kozen, D., editor, *Proceedings of the Workshop on Logic of Programs 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 244–263. Springer Verlag.

[8] Kroening, D., Lerda, F., and Clarke, E. (2004). Bounded model checking for software. In Jensen, K. and Podelski, A., editors, *Proceedings of TACAS 2004*, volume 2988 of *Lecture Notes in Computer Science*. Springer Verlag.

[9] Kurshan, R. P. (1993). *Automata Theoretic Verification of Coordinating Processes*. Princeton University Press.

References II

[10] Prasad, M. R., Biere, A., and Gupta, A. (2005). A survey of recent advances in sat-based formal verification. *International Journal on Software Tools for Technology Transfer*, 7(2):156–173.

[11] Queille, J. P. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In Dezani-Ciancaglini, M. and Montanari, U., editors, *Proceedings of the 5th International Symposium on Programming 1981*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer Verlag.