SMT-Based Verification of Parameterized Systems

Arie Gurfinkel, Sharon Shoham, and Yuri Mesham

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

TECHNION
Israel Institute of Technology

UNIVERSITY OF WATERLOO

Parameterized Verification

System(s)

$P_1, P_2, \ldots P_N$

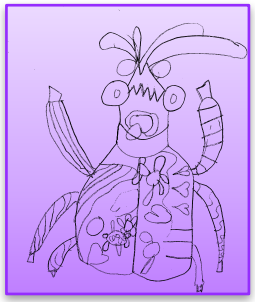
Property

φ

Parameterized Model Checker

$\forall\, i \,.\, P_i \models \varphi$

$\neg(\forall\, i \,.\, P_i \models \varphi)$



Our Contributions

Verification Condition for Safety Properties of Parameterized Systems

- expressed as Constrained Horn Clauses (CHC)
- infinite state transition relation expressed in SMT (arithmetic + arrays)
- invariants/models expressed as *(universally) quantified* first order formulae

Proof-rules for inferring Universally Quantified Invariants

- generalizes Owicki-Gries proof scheme to *k* processes
- systematically derived from the VC above

Decision procedure for specialized finite-data sub-classes

Prototype implementation in Python

- verification of small (in code) infinite state protocols with no finite state abs.
  - bakery, ticket, dining philosophers,…

SYMBOLIC REACHABILITY

Symbolic Reachability Problem

P = (V, Init, $\mathcal{T}r$, Bad)

P is UNSAFE if and only if there exists a number *N* s.t.

$$Init(X_0) \wedge \left(\bigwedge_{i=0}^{N-1} Tr(X_i, X_{i+1}) \right) \wedge Bad(X_N) \not\Rightarrow \bot$$

P is SAFE if and only if there exists a *safe inductive invariant Inv s.t.*

$$\left.\begin{array}{r} Init \Rightarrow Inv \\ Inv(X) \wedge Tr(X, X') \Rightarrow Inv(X') \end{array}\right\}$$

Inductive

$$Inv \Rightarrow \neg Bad$$

Safe

Constrained Horn Clauses (CHC)

A Constrained Horn Clause (CHC) is a FOL formula of the forms

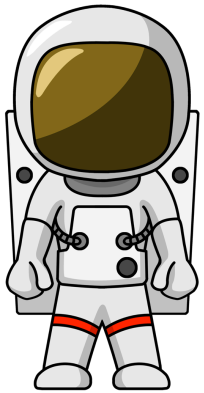
$$\forall\ V\ .\ (\phi \wedge p_1[X_1] \wedge \ldots \wedge p_n[X_n] \rightarrow p_{n+1}[X])$$

$$\forall\ V\ .\ (\phi \wedge p_1[X_1] \wedge \ldots \wedge p_n[X_n] \rightarrow \text{false})$$

where

- ϕ is a constraint in a background theory A
  - usually, A is the combined theory of Linear Arithmetic, Arrays, Bit-Vectors, …
- $p_1, \ldots, p_{n+1}$ are n-ary predicates
- $p_i[X]$ is an application of a predicate to first-order terms

Spacer: Solving SMT-constrained CHC



Spacer: a solver for SMT-constrained Horn Clauses

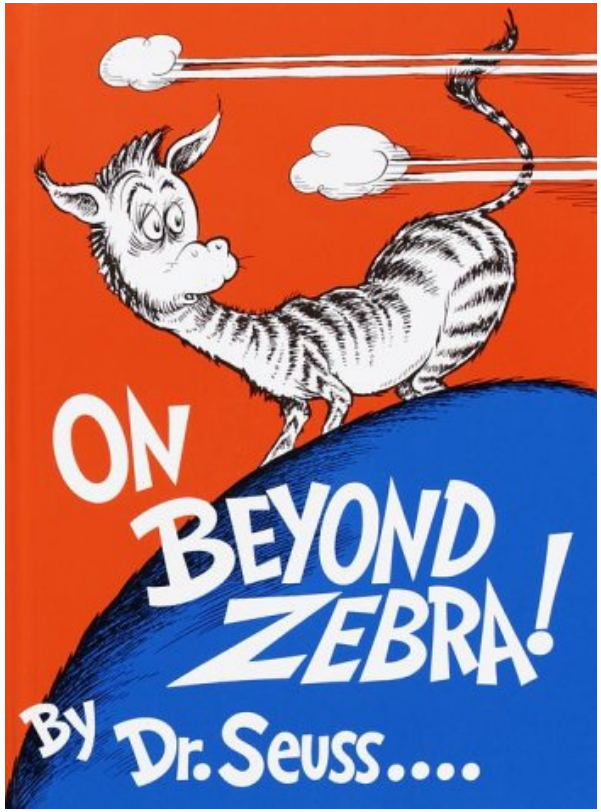
- stand-alone implementation in a fork of Z3
- http://bitbucket.org/spacer/code

Support for Non-Linear CHC

- model procedure summaries in inter-procedural verification conditions
- model assume-guarantee reasoning
- uses MBP to under-approximate models for finite unfoldings of predicates
- uses MAX-SAT to decide on an unfolding strategy

Supported SMT-Theories

- Best-effort support for arbitrary SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic
- Full support for Linear arithmetic (rational and integer)
- Quantifier-free theory of arrays
  - only quantifier free models with limited applications of array equality



PARAMETRIZED SYMBOLIC REACHABILITY

Motivation: Collision Avoidance Protocol

$$
\begin{array}{l}
\textbf{local} \\
\quad pc : \{\text{CHOOSE}, \text{TRY}, \text{WAIT}, \text{MOVE}\} \ ; \\
\quad curr, next, desired : \text{Location} \\
\textbf{def } \texttt{proc}(i) : \\
\quad \textbf{do} \\
\qquad pc[i] = \text{CHOOSE} \ : desired[i] \leftarrow * \ ; pc[i] \leftarrow \text{TRY}; \\
\qquad pc[i] = \text{TRY} \wedge \forall j \,.\, i < j \Rightarrow curr[j] \neq desired[i] \wedge next[j] \neq desired[i] \\
\qquad\ : \\
\qquad\qquad next[i] \leftarrow desired[i] \ ; pc[i] \leftarrow \text{WAIT} \ ; \\
\qquad pc[i] = \text{WAIT} \wedge \forall j \,.\, j < i \Rightarrow next[i] \neq curr[j] \wedge next[i] \neq next[j] \ : \\
\qquad\qquad pc[i] \leftarrow \text{MOVE} \ ; \\
\qquad pc[i] = \text{MOVE} : \\
\qquad\qquad curr[i] \leftarrow next[i] \ ; pc[i] \leftarrow \text{CHOOSE}; \\
\textbf{def } \texttt{init}(i, j) : \\
\quad pc[i] = \text{CHOOSE} \wedge curr[i] = next[i] \wedge (i \neq j \Rightarrow curr[i] \neq curr[j]) \\
\textbf{def } \texttt{bad}(i, j) : \\
\quad i \neq j \wedge curr[i] = curr[j]
\end{array}
$$

Parameterized Symbolic Reachability Problem

T = (**v**, *Init*(N,**v**), *Tr*(*i*, N, **v**, **v**'), *Bad* (N,**v**))

- **v** is a set of state variables
  - each $v_k \in$ **v** is a map *Nat*→*Rat*
  - **v** is partitioned into *Local*(**v**) and *Global*(**v**)
- *Init*(N,**v**) and *Bad*(N,**v**) are initial and bad states, respectively
- *Tr*(*i*, N, **v**, **v**') is a transition relation, parameterized by a process identifier *i* and total number of processes N

All formulas are over the combined theories of arrays and LRA

*Init(*N*,***v***)* and *Bad(*N*,***v***)* contain at most 2 quantifiers

- Init(N,**v**) = $\forall$ x,y . φ_{Init}(N, x, y, **v**), where φ_{Init} is quantifier free (QF)
- Bad(N,**v**) = $\forall$ x,y . φ_{Bad}(N, x, y, **v**), where φ_{Bad} is QF

Tr contains at most 1 quantifier

- *Tr*(*i*, N, **v**, **v**') = $\forall$ *j* . ρ (*i*, *j*, N, **v**, **v**')

A State of a Parameterized System

Global			
v_0	v_1	v_2	v_3

PID	Local			
	v_4	v_5	v_6	v_7
0				
1				
2				
3				
4				
5				
6				
…				
N				

Parameterized Symbolic Reachability

T = (**v**, *Init*, $\mathcal{T}r$, *Bad*)

T is UNSAFE if and only if there exists a number *K* s.t.

$$Init(\boldsymbol{v}_0) \wedge (\bigwedge_{s \in [0,K)} Tr(i_s, N, \boldsymbol{v}_s, \boldsymbol{v}_{s+1})) \wedge Bad(\boldsymbol{v}_K) \not\Rightarrow \bot$$

T is SAFE if and only if there exists a *safe inductive invariant Inv s.t.*

$$\left.\begin{aligned} Init(\boldsymbol{v}) &\Rightarrow Inv(\boldsymbol{v}) \\ Inv(\boldsymbol{v}) \wedge Tr(i, N, \boldsymbol{v}, \boldsymbol{v}') &\Rightarrow Inv(\boldsymbol{v}') \\ Inv(\boldsymbol{v}) &\Rightarrow \neg Bad(\boldsymbol{v}) \end{aligned}\right\} \textbf{VC(T)}$$

Parameterized vs Non-Parameterized Reachability

$$\left.\begin{aligned} Init(\boldsymbol{v}) &\Rightarrow Inv(\boldsymbol{v}) \\ Inv(\boldsymbol{v}) \wedge Tr(i, N, \boldsymbol{v}, \boldsymbol{v}') &\Rightarrow Inv(\boldsymbol{v}') \\ Inv(\boldsymbol{v}) &\Rightarrow \neg Bad(\boldsymbol{v}) \end{aligned}\right\} \textbf{VC(T)}$$

Init, Bad, and Tr might contain quantifiers

- *e.g., "ALL processes start in unique locations"*
- *e.g., "only make a step if ALL other processes are ok"*
- *e.g., "EXIST two distinct process in a critical section"*

Inv cannot be assumed to be quantifier free

- QF *Inv* is either non-parametric or trivial

Decide existence of **quantified** solution for CHC

- stratify search by the number of quantifiers
- solutions with 1 quantifier, 2 quantifiers, 3 quantifiers, etc…

ONE QUANTIFIER
TWO QUANTIFIER

One Quantifier (Solution)

$$\left.\begin{aligned} Init(i,i,\boldsymbol{v}) &\implies Inv_1(i,\boldsymbol{v}) \\ Inv_1(i,\boldsymbol{v}) \wedge Tr(i,\boldsymbol{v},\boldsymbol{v}') &\implies Inv_1(i,\boldsymbol{v}') \\ j \neq i \wedge Inv_1(i,\boldsymbol{v}) \wedge Inv_1(j,\boldsymbol{v}) \wedge Tr(j,\boldsymbol{v},\boldsymbol{v}') &\implies Inv_1(i,\boldsymbol{v}') \\ Inv_1(i,\boldsymbol{v}) \wedge Inv_1(j,\boldsymbol{v}) &\implies \neg Bad(i,j,\boldsymbol{v}) \end{aligned}\right\} \textbf{VC}_1\textbf{(T)}$$

Theorem

- If $VC_1(T)$ is QF-SAT then VC(T) is SAT
- If *Tr* does not contain functions that range over PIDs, then $VC_1(T)$ is QF-SAT only if VC(T) admits a solution definable by a *simple* single quantifier formula
  - simple == quantified id variables do not appear as arguments to functions

$VC_1(T)$ is essentially Owicki-Gries for 2 processes *i* and *j*

If there are no global variables then (3) is unnecessary

- $VC_1(T)$ is linear

How do we get it

1. Restrict Inv to a fixed number of quantifiers
   - e.g., replace Inv(N, v) with $\forall k.Inv_1(k, N, v)$

2. Case split consecution Horn clause based on the process that makes the move
   - w+1 cases for w-quantifiers
     - one for each quantified id variable
     - one for interference by “other” process (only for global variables)

3. Instantiate the universal quantifier in $\forall k.Inv_1(k, N, v)$
   - use symmetry to reduce the space of instantiations

4. Other instantiations might be needed for quantifiers if
   - id variables appear as arguments to functions

How do we get it

$$Inv(\boldsymbol{v}) \wedge Tr(j, \boldsymbol{v}, \boldsymbol{v}') \implies Inv(\boldsymbol{v}')$$

Restrict

$$(\forall k \cdot Inv_1(k, \boldsymbol{v})) \wedge Tr(j, \boldsymbol{v}, \boldsymbol{v}') \implies Inv_1(i, \boldsymbol{v}')$$

Cases

$$(\forall k \cdot Inv_1(k, \boldsymbol{v})) \wedge Tr(i, \boldsymbol{v}, \boldsymbol{v}') \implies Inv_1(i, \boldsymbol{v}')$$
$$(\forall k \cdot Inv_1(k, \boldsymbol{v})) \wedge j \neq i \wedge Tr(j, \boldsymbol{v}, \boldsymbol{v}') \implies Inv_1(i, \boldsymbol{v}')$$

Instantiate

$$Inv_1(i, \boldsymbol{v}) \wedge Tr(i, \boldsymbol{v}, \boldsymbol{v}') \implies Inv_1(i, \boldsymbol{v}')$$
$$Inv_1(i, \boldsymbol{v}) \wedge Inv_1(j, \boldsymbol{v}) \wedge j \neq i \wedge Tr(j, \boldsymbol{v}, \boldsymbol{v}') \implies Inv_1(i, \boldsymbol{v}')$$

Two Quantifier Solution

$$Init(i,j,\boldsymbol{v}) \wedge Init(j,i,\boldsymbol{v}) \wedge Init(i,i,\boldsymbol{v}) \wedge Init(j,j,\boldsymbol{v}) \Rightarrow I_2(i,j,\boldsymbol{v})$$
$$I_2(i,j,\boldsymbol{v}) \wedge Tr(i,\boldsymbol{v},\boldsymbol{v}') \Rightarrow I_2(i,j,\boldsymbol{v}')$$
$$I_2(i,j,\boldsymbol{v}) \wedge Tr(j,\boldsymbol{v},\boldsymbol{v}') \Rightarrow I_2(i,j,\boldsymbol{v}')$$
$$I_2(i,j,\boldsymbol{v}) \wedge I_2(i,z,\boldsymbol{v}) \wedge I_2(j,z,\boldsymbol{v}) \wedge Tr(z,\boldsymbol{v},\boldsymbol{v}') \wedge z \neq i \wedge z \neq j \Rightarrow I_2(i,j,\boldsymbol{v}')$$
$$I_2(i,j,\boldsymbol{v}) \Rightarrow \neg Bad(i,j,\boldsymbol{v})$$

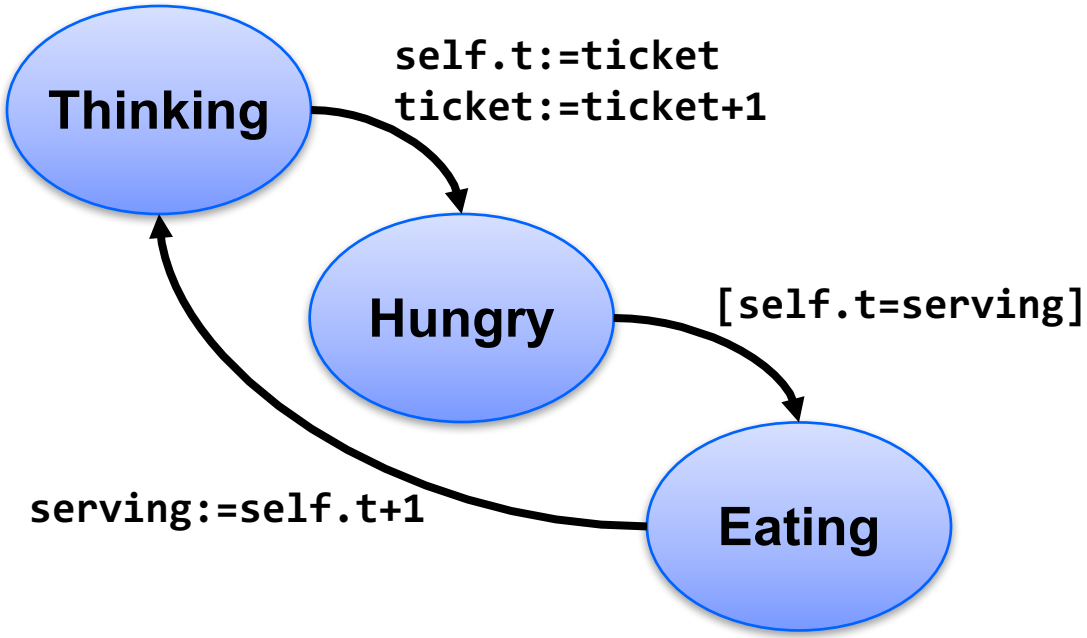
Theorem

- If $VC_2(T)$ is QF-SAT then VC(T) is SAT
- If *Tr* does not contain functions that range over PIDs, then $VC_2(T)$ is QF-SAT only if VC(T) admits a solution definable by a *simple* two quantifier formula

- At least 2 quantifiers are “needed” for systems with global guards

Extends to *k*-quantifiers

Ticket Protocol




```
INIT: pc=Thinking, ticket=0, serving=0

BAD: self.pc=Eating, other.pc=Eating
```

Emacs-x86_64-10_9 Prelude - /tmp/ticket.log

```
(define-fun Inv ((id0 Int) (id1 Int) (pc (Array Int Int)) (t (Array Int Int)) (s
erving Int) (ticket Int)) Bool
(let ((a!1 (<= (+ (select t id1) (* (- 1) (select t id0))) (- 1)))
      (a!2 (<= (+ (select t id0) (* (- 1) (select t id1))) (- 1)))
      (a!3 (or (<= (select pc id1) 1) (<= (+ serving (* (- 1) ticket)) (- 1))))
      (a!4 (or (<= (select pc id0) 0)
               (<= (+ (select t id0) (* (- 1) ticket)) (- 1))))
```

$$\forall i, j, i \neq j \Rightarrow (i.pc \neq E) \vee (j.pc \neq E)$$

$$\forall i, i.pc = E \Rightarrow serving < ticket$$

$$\forall i, i.pc \in \{H, E\} \Rightarrow i.t < ticket$$

$$\forall i, j, i \neq j \Rightarrow (i.pc \in \{H, E\} \wedge j.pc \in \{H, E\} \Rightarrow i.t \neq j.t)$$

$$\forall i, j, i \neq j \Rightarrow (i.pc = E \wedge j.pc \in \{H, E\} \Rightarrow j.t \neq serving)$$

```
      (or (<= (select pc id1) 0) (<= (select pc id0) 0) a!1 a!2)
      a!3
      a!4
      a!5
      a!6
      (or a!7 (<= (select pc id1) 1) a!8 (<= (select pc id0) 0))
      a!10)))
)
```

Putting it all together

Solve for Inductive Invariant

$k := 1$;
while *true* **do**
  $Inv_k(i_1, \ldots, i_k, \boldsymbol{v}) := \text{Solve}(U^k(VC^\omega(T)))$;
  if $Inv_k(i_1, \ldots, i_k, \boldsymbol{v}) \neq null$ **then**
    return "*inductive invariant found:* $\forall i_1, \ldots, i_k \,.\, Inv(i_1, \ldots, i_k, \boldsymbol{v})$"
  $res := \text{ModelCheck}(T_k)$;
  if $res = cex$ **then**
    return "*counterexample found for k processes*"
  $k := k + 1$

Look for bugs

Finite vs Infinite Number of Processes

Init b[i] and move to pc=D

Move to pc=E when all distinctly init

$$\textbf{def } \texttt{proc}(i) :$$
$$\textbf{do}$$
$$pc[i] = I \ : \ pc[i] := D; b[i] := 1;$$
$$pc[i] = I \ : \ pc[i] := D; b[i] := 0;$$
$$(\forall j \neq i \,.\, pc[i] = D \wedge pc[j] \neq I \wedge b[j] \neq b[i]) \ : \ pc[i] := E;$$
$$\textbf{def } \texttt{init}(i, j) : \ pc[i] = I;$$
$$\textbf{def } \texttt{bad}(i, j) : \ i \neq j \wedge pc[i] = E \wedge pc[j] = E;$$

Tr does not depend on N (number of processes)

Safe for infinitely many processes. Invariant is:

$$\begin{aligned} Inv = (\forall i, j \,.\, i \neq j \Rightarrow (pc[i] \neq E \vee pc[j] \neq E)) \wedge \\ (\forall i \,.\, pc[i] \neq I \Rightarrow b[i] \in [0, 1]) \wedge \\ (\forall i, j \,.\, (pc[i] = E \wedge i \neq j) \Rightarrow (pc[j] \neq I \wedge b[i] \neq b[j])). \end{aligned}$$

Unsafe for N = 2!

Evaluation and Implementation

Python-based Implementation

- Simple language for specifying concurrent protocols
- Local and Universally guarded transitions
- Constraints over arrays and integer arithmetic
- Reduce to CHC using the rules and solve using Spacer

Evaluated on Simple/Tricky Well-Know Protocols

- Dining philosophers, bakery1, bakery2, collision avoidance, ticket
- Models are pretty close to an implementation
  - limit abstraction in modeling, try to make verification hard
- Safe inductive invariants computed within seconds

Related Work

Kedar Namjoshi et al.

- Local Proofs for Global Safety Properties, and many other papers
- systematic derivation of proof rules for *concurrent* systems
- finite state and fixed number of processes

Andrey Rybalchenko et al.

- Compositional Verification of Multi-Threaded Programs, and others
- compositional proof rules for concurrent systems are CHC
- infinite state and fixed number of processes

Lenore Zuck et al.

- Invisible Invariants
- finite state and parametric number of processes
- finite model theorem for special classes of parametric systems

Nikolaj Bjørner, Kenneth L. McMillan, and Andrey Rybalchenko

- On Solving Universally Quantified Horn Clauses. SAS 2013:

Conclusion

Parameterized Verification == Quantified solutions for CHC

Quantifier instantiation to *systematically* derive proof rules for verification of safety properties of parameterized systems

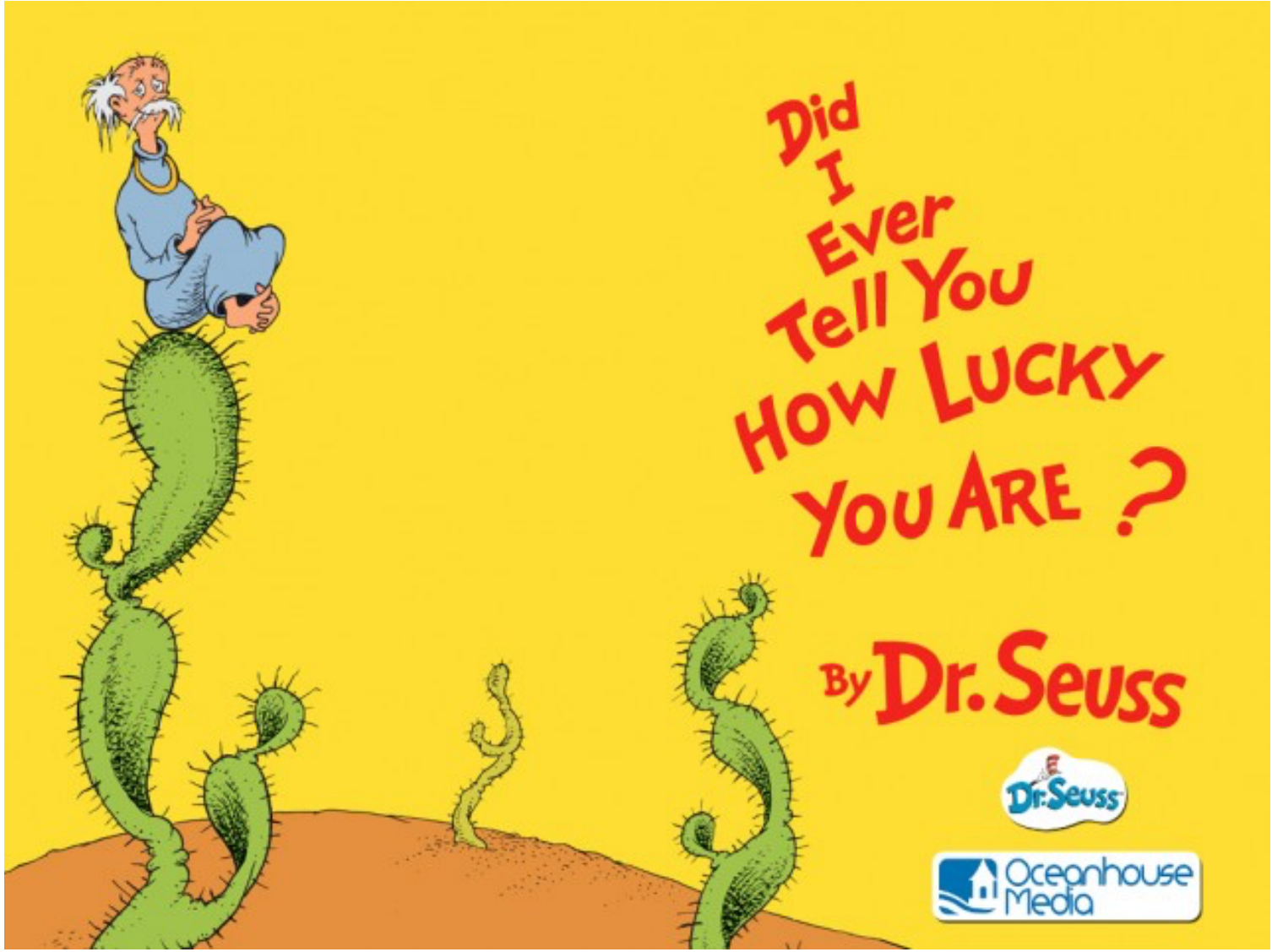
- Parameterized systems definable with SMT-LIB syntax

Lazy vs Eager Quantifier Instantiation

- eager instantiation in this talk
- would be good to extend to lazy / dynamic / model-based instantiation

Connections with other work in parameterized verification

- complete instantiation = decidability ?
- relative completeness
- …


Did I Ever Tell You How Lucky You Are ?
By Dr. Seuss
Dr. Seuss
Oceanhouse Media

?

?

?

?

?

?