Proof Certificates for SMT-based Model Checkers for Infinite State Systems

Alain Mebsout and Cesare Tinelli
FMCAD 2016
October 5th, 2016
Motivation

• Model checkers return error traces but no evidence when they say yes

• Complex tools
Motivation

- Model checkers return error traces but no evidence when they say yes
- Complex tools
- **Goal**: improve trustworthiness of these tools
- **Approach**: produce proof certificates
Motivation

- Model checkers return error traces but no evidence when they say yes
- Complex tools
  - **Goal**: improve trustworthiness of these tools
  - **Approach**: produce proof certificates
- Implemented in Kind 2
Certificate generation and checking
Proof certificate production as a two-steps process

System \( S \)
Property \( P \)

- Kind 2
- SMT2 certificate
- CVC4
- Validity proofs
- Safety proof
- LFSC

Signatures
- k-induction
- SMT Theories
Intermediate certificates

System $S$

Property $P$

Kind 2 $\rightarrow$ SMT2 certificate $\rightarrow$ CVC4 $\rightarrow$ safety proof $\rightarrow$ LFSC

Signatures

- $k$-induction
- SMT Theories

CVC4

validity proofs
where $\phi$ is $k$-inductive and implies the property $P$, 
$\Rightarrow$ enough to prove that $P$ holds in $S = (x, I, T)$
Intermediate Certificates

where \( \phi \) is \( k \)-inductive and implies the property \( P \),
\( \Rightarrow \) enough to prove that \( P \) holds in \( S = (x, I, T) \)
Minimization of Intermediate (SMT-LIB 2) Certificates

Two dimensions:

- reduce $k$
- simplify inductive invariant
  - with unsat cores
  - with counter-examples to induction

**Rationale:** easier to check a smaller/simpler certificate
A taste of certificate minimization

(1) Trimming invariants

certificate: \((1, \phi_1 \land \ldots \land \phi_n \land P)\)

\[
\phi_1 \land \ldots \land \phi_n \land P \land T \land \neg P' \models \bot
\]

invariants: R

property
A taste of certificate minimization

(1) Trimming invariants

certificate: $(1, \phi_1 \land \ldots \land \phi_n \land P)$

$$\phi_1 \land \ldots \land \phi_n \land P \land T \land \neg P' \models \bot$$

from unsat core: $R_0 \subseteq R$
A taste of certificate minimization

(1) Trimming invariants
certificate: \((1, \phi_1 \land \ldots \land \phi_n \land P)\)

\[
\phi_1 \land \ldots \land \phi_n \land P \land T \land \neg P' \models \bot
\]

invariants: \(R\)

property

from unsat core: \(R_0 \subseteq R\)

? \[
R_0 \land P \land T \models R_0' \land P'
\]
A taste of certificate minimization

(1) Trimming invariants

certificate: \((1, \phi_1 \land \ldots \land \phi_n \land P)\)

\[
\phi_1 \land \ldots \land \phi_n \land P \land T \land \neg P' \models \bot
\]

invariants: \(R\)

property

from unsat core: \(R_0 \subseteq R\)

\[
R_0 \land P \land T \models R_0' \land P'
\]

- yes: keep \(R_0 \cup P\)
- no: restart with \(P := R_0 \cup P\) and \(R := R \setminus R_0\)
(2) Cherry-picking invariants certificate: $(1, \underbrace{\phi_1 \land \ldots \land \phi_n \land P})$

$$P \land T \not\models P'$$
(2) Cherry-picking invariants certificate: \((1, \underbrace{\phi_1 \land \ldots \land \phi_n} \land P)\)

\[ P \land T \not\models P' \]

from model \(\mathcal{M} : \phi \in R\) such that \(\mathcal{M} \not\models \phi\)
(2) Cherry-picking invariants certificate: $(1, \overbrace{\phi_1 \land \ldots \land \phi_n}^{R} \land P)$

$$P \land T \not\models P'$$

from model $\mathcal{M} : \phi \in R$ such that $\mathcal{M} \not\models \phi$

$$P := \phi \land P \quad R := R \setminus \{\phi\}$$
Front End Certificates
Translation from one formalism to another are sources of error

In Kind 2,

- input = Lustre
- several intermediate representations
- many simplifications (slicing, path compression, encodings, …)
Translation from one formalism to another are sources of error

In Kind 2,

- input = Lustre
- several intermediate representations
- many simplifications (slicing, path compression, encodings, ...)

How to trust the translation from input language to internal FOL representation?
Translation from one formalism to another are sources of error

In Kind 2,

- input = Lustre
- several intermediate representations
- many simplifications (slicing, path compression, encodings, …)

How to trust the translation from input language to internal FOL representation?

Lightweight verification akin to Multiple-Version Dissimilar Software Verification of DO-178C (12.3.2)
Front end certificates in Kind 2: approach

Previous certification chain for Kind 2

Lustre input file

JKind frontend

Kind 2 frontend

\[ S_1 = (x_1, I_1, T_1) \]

\[ P_1 \]

\[ S_2 = (x_2, I_2, T_2) \]

\[ P_2 \]

Observer of equivalence (OBS)

\[ x_{obs} = x_1 \uplus x_2 \]

\[ S_{obs} \]

\[ P_{obs}(x_{obs}) = x_1 \sim x_2 \]

Native input

Kind 2 core

Previous certification chain for Kind 2

CVC4 + LFSC

SMT-LIB 2

\[ C(S_{obs}, P_{obs}) \]

SMT2 Front End certificate (FEC)
LFSC Proofs
Producing proofs

System $\mathcal{S}$
Property $P$

Kind 2 $\rightarrow$ SMT2 certificate $\rightarrow$ CVC4

validity proofs $\rightarrow$ safety proof $\rightarrow$ LFSC

Signatures

k-induction
SMT Theories

documentation
Producing proofs of invariance

\[ S = (s, I[s], T[s, s']) : \text{input system} \]
\[ P[s] : \text{property proven invariant for } S \]
\[ (k, \phi[s]) : \text{certificate produced by Kind 2} \]

- We can formally check that \( \phi \)
  1. is \( k \)-inductive
  2. implies \( P \)

- **Our goal:** produce a detailed, self-contained and independently machine-checkable proof
Proving invariance by $k$-induction

$S = (s, l[s], T[s, s'])$ : input system

$P[s]$ : property proven invariant for $S$

$(k, \phi[s])$ : certificate produced by Kind 2

$\phi$ is a $k$-inductive strengthening of $P$:

$l[s_0] \land T[s_0, s_1] \land \ldots \land T[s_{k-2}, s_{k-1}] \models \phi[s_0] \land \ldots \land \phi[s_{k-1}]$

$(base_k)$

$\phi[s_0] \land T[s_0, s_1] \land \ldots \land \phi[s_{k-1}] \land T[s_{k-1}, s_k] \models \phi[s_k]$

$(step_k)$

$\phi[s] \models P[s]$

(implication)
Proving invariance by \( k \)-induction

\( S = (s, I[s], T[s, s']) \) : input system

\( P[s] \) : property proven invariant for \( S \)

\( (k, \phi[s]) \) : certificate produced by Kind 2

\( \phi \) is a \( k \)-inductive strengthening of \( P \):

\[
l[s_0] \land T[s_0, s_1] \land \ldots \land T[s_{k-2}, s_{k-1}] \models \phi[s_0] \land \ldots \land \phi[s_{k-1}]
\]

\( \text{(base}_k \text{)} \)

\[
\phi[s_0] \land T[s_0, s_1] \land \ldots \land \phi[s_{k-1}] \land T[s_{k-1}, s_k] \models \phi[s_k]
\]

\( \text{(step}_k \text{)} \)

\[
\phi[s] \models P[s]
\]

(implication)
Use CVC4 to generate proofs for the validity of each sub-case.

Kind 2 generates a proof of invariance by $k$-induction and reuses the proofs of CVC4.
LFSC rules

System $\mathcal{S}$

Property $P$

Kind 2 → SMT2 certificate → CVC4 → safety proof → LFSC

SMT Theories

Signatures

k-induction

CVC4

validity proofs

LFSC

proofs

Certificate

Validity

Safety
Encoding of Lustre variables as functions over naturals (indexes)

In Lustre

```
node main (a: bool) returns (OK: bool)
var b: bool;
...
```

In the LFSC signature:

```
(declare index sort)
(declare ind int -> index)
```

In the LFSC proof:

```
(declare a (term (arrow index Bool)))
(declare b (term (arrow index Bool)))
(declare OK (term (arrow index Bool)))
...
```
Predicates and relations over copies of the same state
⇒ predicates/relations over indexes

• $P[s_i] \leadsto P_s(i)$
• $R[s_i, s_j] \leadsto R_s(i, j)$
Predicates and relations over copies of the same state $\leadsto$ predicates/relations over indexes

- $P[s_i] \leadsto P_s(i)$
- $R[s_i, s_j] \leadsto R_s(i,j)$

In the LFSC signature:

;; relations over indexes (used for transition relation)
(define rel int $\rightarrow$ int $\rightarrow$ formula)

;; sets over indexes (used for initial formula and properties)
(define set int $\rightarrow$ formula)

;; derivability judgment for invariance proofs
(declare invariant set $\rightarrow$ rel $\rightarrow$ set $\rightarrow$ type)
Predicates and relations over copies of the same state
→ predicates/relations over indexes

- \( P[S_i] \rightarrow P_s(i) \)
- \( R[S_i, S_j] \rightarrow R_s(i, j) \)

In the LFSC proof:

;; encoding of property
(define P : set
  (λ i. (p_app (apply _ _ OK (ind i)))))

;; encoding of transition relation
(define T : rel
  (λ i. λ j. ...))
LFSC rules – $k$-induction

(declare $k$-ind
   $\Pi k::\text{int}$. ; bound $k$
   $\Pi I::\text{set}$. ; initial states
   $\Pi T::\text{rel}$. ; transition relation
   $\Pi P::\text{set}$. ; $k$-inductive invariant

; $B$ is formula for base case
$\Pi r1:^B = (\text{base }I \ T \ P \ k)$.

; $S$ is formula for step case
$\Pi r2:^S = (\text{step }T \ P \ k)$.

; proof of base case
$\Pi ub:(\text{th\_holds } B)$.

; proof of step case
$\Pi us:(\text{th\_holds } S)$.

;-----------------------------------
invariant $I \ T \ P$
)

\begin{align*}
\text{ub} & \quad \vdash B \\
\text{us} & \quad \vdash S \\
\text{K-IND} & \quad \text{Invariant}(I, T, P) \\
B & = \text{base}_k(I, T, P) \\
S & = \text{step}_k(I, T, P)
\end{align*}
(declare inv-impl
   \Pi I : set. \Pi T : rel.
   \Pi P1 : set. \Pi P2 : set.

   ;; proof that \( P1 \Rightarrow P2 \)
   \Pi u :
      \Pi k : int.
      th_holds ((P1 k) \Rightarrow (P2 k)).

   ;; proof that \( P1 \) is invariant
   \Pi i :
      invariant I T P1.

   ;-----------------------------
   invariant I T P2
)

INV-IMPL

\( \equiv P1 \Rightarrow P2 \)

Invariant(I, T, P1)

Invariant(I, T, P2)
Self-contained proofs

;; derivability judgment for safety
(declare safe set → rel → set → type)

safety¹ =

  invariance of property in encoded system

  +

  existence of another system which is weak-observational equivalent to it

¹as defined in this signature
Sketch of derivation tree for LFSC proofs of safety produced by Kind 2

\[
\begin{align*}
\text{K-IND} & \quad k \in \mathbb{N} & \quad \text{SMT} & \quad B_k & \quad \text{SMT} & \quad S_k & \quad \text{SMT} & \quad \vdash P \quad \text{safe}(I, T, P) \\
\text{INV-IMPL} & \quad \text{invariant}(I, T, \phi) & \quad \text{SMT} & \quad \phi \vdash P \\
\text{INV+OBS} & \quad \text{invariant}(I, T, P) & \quad \text{K-IND} & \quad \text{INV-IMPL} & \quad \text{OBSEQ} & \quad \text{SMT} & \quad \vdash P_0 \quad \text{woe}(I, T, P, I', T', P')
\end{align*}
\]
Sketch of derivation tree for LFSC proofs of safety produced by Kind 2

\[
\begin{align*}
\text{inv}_{-\text{impl}} & : 
\begin{array}{c}
\text{K-IND} \\
\text{INV-IMPL} \\
\text{INV+OBS}
\end{array}
\quad \begin{array}{c}
\text{SMT} \quad \text{inv}_{-\text{impl}}(I, T, \phi) \\
\text{inv}_{-\text{impl}}(I, T, P)
\end{array}
\quad \begin{array}{c}
\text{SMT} \\
\phi \models P \\
\text{SMT} \quad \text{safe}(I, T, P)
\end{array}
\end{align*}
\]
Sketch of derivation tree for LFSC proofs of safety produced by Kind 2

\[ \text{safe}(I, T, P) \]

\[ I_{o}(i) = \text{same}_\text{inputs}(i) \land I(i) \land I'(i) \]

\[ T_{o}(i, j) = \text{same}_\text{inputs}(i) \land T(i, j) \land T'(i, j) \]

\[ P_{o}(k) = P(k) \iff P'(k) \]
Sketch of derivation tree for LFSC proofs of safety produced by Kind 2
Sketch of derivation tree for LFSC proofs of safety produced by Kind 2

\[ \begin{align*}
\text{INV+OBS} & \quad \text{inv}(I; T; P) \\
\text{INV-IMPL} & \quad \text{inv}(I; T; \phi) \\
\text{K-IND} & \quad k \in \mathbb{N} \quad \models B_k \quad \models S_k \\
\text{SMT} & \quad \phi \models P \\
\text{INV-IMPL} & \quad \text{inv}(I_0; T_0; \phi_0) \quad \text{inv}(I_0; T_0; P_0) \\
\text{OBSEQ} & \quad \text{woe}(I, T, P, I', T', P') \\
\text{SMT} & \quad \phi_0 \models P_0 \\
\text{K-IND} & \quad \text{inv}(I_0; T_0; \phi_0) \quad \text{inv}(I_0; T_0; P_0) \\
\text{SMT} & \quad \phi_0 \models P_0 \\
\text{safe}(I, T, P)
\end{align*} \]
Small Lustre node: detection of rising edge:

```plaintext
node edge (x: bool) returns (y: bool);
var OK: bool;
let
  y = false -> x and not pre x;
  OK = not x => not y;
--%PROPERTY OK;
tel
```
LFSC proof for rising edge node

;; LFSC proof produced by kind2 v0.8.0-425-g294ec4d and CVC4
;; from original problem ex.lus

;; Declarations and definitions
(define edge.usr.x (term (arrow index Bool)))
(define edge.usr.y (term (arrow index Bool)))
(define edge.res.init_flag (term (arrow index Bool)))
(define edge.impl.usr.OK (term (arrow index Bool)))

(define I (: (! _ int formula)
  (\ I%1 (@ let3 (ind I%1)) (@ let4 (p_app (apply _ edge.usr.y (ind I%1))) (and (iff let4 false)
  (and (iff (p_app (apply _ edge.impl.usr.OK (ind I%1))) (impl (not (p_app (apply _ edge.usr.x (ind I%1)))) (not let4)))
  (and (p_app (apply _ edge.res.init_flag (ind I%1))) true))))))

(define T (: (! _ int (! _ int formula))
  (\ T%1 (@ T%2 (ind T%2)) (@ let22 (ind T%2)) (@ let23 (p_app (apply _ edge.usr.y (ind T%2))) (and (iff let23 (and let24 (not (p_app (apply _ edge.usr.x (ind T%2)))) (impl (not let24) (not let23))) (and (iff (p_app (apply _ edge.impl.usr.OK (ind T%2)))
  (impl (not let24) (not let23))) (and (not (p_app (apply _ edge.res.init_flag (ind T%2)))) true))))))

(define P (: (! _ int formula) (\ P%1 (p_app (apply _ edge.impl.usr.OK (ind P%1))))))

(define PHI (: (! _ int formula) (\ PHI%1 (p_app (apply _ edge.impl.usr.OK (ind PHI%1))))))
LFSC proof for rising edge node (cont.)

(define base
  (: (! A0 (th_holds (@ let1 (ind 0) (@ let2 (p_app (apply __ edge.usr.y (ind 0))) (@ let5 (p_app (apply __ edge.impl.usr.OK (ind 0)))) (and (and (iff let2 false) (and (iff let5 (not (p_app (apply __ edge.x (ind 0)))) (not let2)) (and (p_app (apply __ edge.res.init_flag (ind 0)) true))) (not let7)))))) (holds cln)) (\ A0 (th_let_pf _ (trust_f false) (\ .PA193 (th_let_pf _ (trust_f (not false)) (\ .PA197 (decl_atom false (\ .V1 (\ .A1 (satlem __ (asf _ _ _ .A1 (\ .L2 (clausify_false (contra _ .PA193 .L2)))) (\ .PB4 (satlem_simplify _ _ (R __ pb4 pb3 .V1) (\empty empty)))))))))))))
)

(define induction
  (: (! A0 (th_holds (@ let1 (ind 0) (@ let3 (ind 1) (@ let4 (p_app (apply __ edge.usr.y (ind 1))) (@ let5 (p_app (apply __ edge.x (ind 1))) (@ let10 (p_app (apply __ edge.impl.usr.OK (ind 1)))) (and (and (p_app (apply __ edge.impl.usr.OK (ind 0)))) (and (iff let4 (and let5 (not (p_app (apply __ edge.x (ind 0)))) (not let4)) (and (iff let10 (not let5) (not let4)) (and (not (p_app (apply __ edge.res.init_flag (ind 1)) true))) (not let10)))))))) (holds cln)) (\ A0 (th_let_pf _ (trust_f false) (\ .PA193 (th_let_pf _ (trust_f (not false)) (\ .PA197 (decl_atom false (\ .V1 (\ .A1 (satlem __ (asf _ _ _ .A1 (\ .L2 (clausify_false (contra _ .PA193 .L2)))) (\ .PB4 (satlem_simplify _ _ (R __ pb4 pb3 .V1) (\empty empty)))))))))))))
)

(define implication
  (: (! %%k int (! A0 (th_holds (@ let2 (p_app (apply __ edge.impl.usr.OK (ind %%k)))) (not (impl let2 let2))) (holds cln))) (\ %%k (\ A0 (th_let_pf _ (trust_f false) (\ .PA193 (th_let_pf _ (trust_f (not false)) (\ .PA197 (decl_atom false (\ .V1 (\ .A1 (satlem __ (asf _ _ _ .A1 (\ .L2 (clausify_false (contra _ .PA193 .L2)))) (\ .PB3 (satlem_simplify _ _ (R __ pb4 pb3 .V1) (\empty empty)))))))))
)

;; Proof of invariance by 1-induction
(define proof_inv
  (: (invariant I T P)
    (inv-impl I T PHI P implication
      (k-ind 1 I T PHI __ base induction))))

(check proof_inv)
LFSC proof for rising edge node (cont.)

LFSC proof produced by kind2 v1.0.alpha1-208-gae70098 and
CVC4 version 1.5-prerelease [git proofs 7ba546df]
for frontend observational equivalence and safety
(depending on proof.lfsc)

System generated by JKind

(declare JKind.$x$ (term (arrow index Bool)))
(declare JKind.$y$ (term (arrow index Bool)))
(declare f1 (term (arrow index Bool)))
(declare JKind.$OK$ (term (arrow index Bool)))

(define I2 (: (! _ int formula) ...))
(define T2 (: (! _ int (! _ int formula)) ...))
(define P2 (: (! _ int formula) ...))

System generated for Observer

(define same_inputs (: (! _ int formula)
      \ same_inputs%1 (@ let73 (ind same_inputs%1)
                  (iff (p_app (apply _ _ edge.usr.x let73))
                       (p_app (apply _ _ JKind.$x$ let73)))))

(define IO (: (! _ int formula) ...))
(define TO (: (! _ int (! _ int formula)) ...))
(define PO (: (! _ int formula) ...))
LFSC proof for rising edge node (cont.)

;; k-Inductive invariant for observer system
(define PHIO (: (! _ int formula) ...))

;; Proof of base case
(define base_proof_2 ...)

;; Proof of inductive case
(define induction_proof_2 ...)

;; Proof of implication
(define implication_proof_2 ...)

;; Proof of invariance by 1-induction
(define proof_obs (: (invariant IO TO PO)
  (inv-impl IO TO PHIO PO implication_proof_2
    (k-ind 1 IO TO PHIO _ _ base_proof_2 induction_proof_2))))

;; Proof of observational equivalence
(define proof_obs_eq (: (weak_obs_eq I T P I2 T2 P2)
  (obs_eq I T P I2 T2 P2 same_inputs proof_obs)))

;; Final proof of safety
(define proof_safe (: (safe I T P) (inv+obs I T P I2 T2 P2 proof_inv proof_obs_eq)))

(check proof_safe)
Checking the proof

Proof checker

proof checker generator

proof rules

proof

> lfsc-checker sat.plf smt.plf th_base.plf th_int.plf th_real.plf kind.plf proof.lfsc

signature for SAT solving (resolution)
signature for SMT (cnf + theory)
signature for EUF theory
symbols for linear integer arithmetic
symbols for linear real arithmetic
signature for $k$-induction
- proved invariance (of encoded system) for 80%
  
  (rest is unsupported fragment of proofs for CVC4)
The trusted core of our approach consists in:

1. LFSC checker (5300 lines of C++ code)

2. LFSC signatures comprising the overall proof system LFSC (for a total of 444 lines of LFSC code)

3. Assumption that Kind 2 and JKind do not have identical defects that could escape the observational equivalence check. (reasonable considering the differences between the two model checkers)
Current limitations

- Holes in proofs produced by CVC4 (`trust_f` rule):
  - pre-processing
  - arithmetic lemmas

Generate additional sub-goals whose proof has to be filled in (manually, or other)

- Doesn’t work with combination of both real and integer arithmetic for now
• Kind 2 generates machine checkable proofs of invariance and safety in LFSC

• Currently limited by CVC4 capabilities for proofs ...

• ... but ready for when CVC4 will produce proofs for more theories
Ongoing and future work

• Support **compositional proofs with abstraction** (by extending the LFSC signature)

• Leverage proofs for **tool qualification** — DO-178C, DO-330 (ongoing, collaboration with Rockwell Collins and NASA)

• **Prove correctness** of rules and side-conditions in a proof assistant like Coq or Isabelle
Thank you