SAT-Based Model Checking:
IC3 and Lazy Abstraction

Verification course
Lecture 11, June 19, 2017

Part A
Lazy Abstraction and SAT-Based Reachability (with IC3) in Hardware Model Checking

[Vizel, Grumberg, Shoham FMCAD 2012]
Model

• System is modeled as \((V, I, T)\), where:
  - \(V\) is a finite set of variables
  - \(I \subseteq 2^V\) is the set of initial states
  - \(T \subseteq 2^V \times 2^V\) is the set of transitions

Suitable for hardware: \(V\) is over \(\{0, 1\}\)

• A safety property of the form \(\text{AG} P\)
  - \(P\) is a propositional formula over \(V\)
Modeling hardware

Concrete (un-abstracted) model:

• $M_c(V, U, I, T)$, where:

  - $V$ is a finite set of variables
  - $U \subseteq V$ is a set of state variables
  - $V \setminus U$ is the set of input variables
  - $I(V) \subseteq 2^V$ is the set of initial states
  - $T(V, V') \subseteq 2^V \times 2^V$ is the set of transitions

\[
T(V, V') = \bigwedge_{v \in U} (v' = f_v(V, V'))
\]
Visible Variables Abstraction
Visible-variables abstract model

- $M_i(V_i, U_i, I, T_i)$ is an abstract model where:
  - $V$ is a finite set of variables
  - $U_i \subseteq U$ visible variables
  - The rest are input (invisible) variables
  - $T_i (V, U'_i) = \bigwedge_{v \in U_i} (v' = f_v (V, V'))$
  - $V_i = \{ v \mid v \in \text{Vars}(T_i) \text{ or } v' \in \text{Vars}(T_i) \}$
Properties of the abstract models

• $M_c < M_i$ (simulation relation)

• An abstract state $s_i$ represents the set of all concrete states $s$ that agrees with $s_i$ on $V_i$
  - $s < s_i$

• $T \Rightarrow T_i$
  every transition in $T$ has a corresponding transition in $T_i$
Abstraction-Refinement

- Abstract model may contain spurious behaviors
  - Spurious counterexample may exist

- Refinement is applied to remove the spurious behavior
Lazy Abstraction

• Different abstractions at different steps of verification

• Refinement is applied locally, where needed
IC3 Basics (reminder)

• Iteratively compute Over-approximated Reachability Sequence (OARS) \( \langle F_0, F_1, \ldots, F_k \rangle \)

s.t.
- \( F_0 = \text{INIT} \)
- \( F_i \Rightarrow p \) : \( p \) is an invariant up to \( k \)
- \( F_i \Rightarrow F_{i+1} \) : \( F_i \subseteq F_{i+1} \)
- \( F_i \land T \Rightarrow F'_{i+1} \) : Simulates one forward step

\( F_i \) - over-approximates the set of states reachable within \( i \) steps
Iteration of IC3

\[ F_{k+1} = p \]

\[ F_k \land T \not\Rightarrow F'_{k+1} \]
Locality in IC3

- IC3 applies checks of the form
  - $F_k \land T \land \neg P'$
    - Finds a state in $F_k$ that can reach $\neg P$
  - $F_i \land T \land s'$
    - Finds a predecessor in $F_i$ to the state $s$

- Using only one $T$
  - No unrolling
The lazy abstraction Approach - L-IC3

• Use IC3’s local checks for Lazy Abstraction
  - Different abstraction at different time frames
  - Use visible variables abstraction
    • Different variables are visible at different time frames
Concrete Model
Using Abstraction

INIT → F_1 → ... → F_{k-1} → F_k
Using Lazy Abstraction
Lazy Abstraction + IC3 = L-IC3

- $\langle F_0, F_1, \ldots, F_{k+1} \rangle$ - (abstract) sets of reachable states
  - OARS with respect to $T$

- $\langle U_0, U_1, \ldots, U_k \rangle$ - abstraction sequence
  - $U_i$ - set of visible variables
    - $U_i$ variables have a next state function
    - The rest, inputs
  - $U_i \subseteq U_{i+1}$: monotone
    - $U_{i+1}$ is a refinement of $U_i$
IC3 - Initialization (reminder)

- Check satisfiability of the two formulas:
  - $I \land \neg P$
  - $I \land T \land \neg P'$
- If both are unsatisfiable then:
  - $I \Rightarrow P$
  - $I \land T \Rightarrow P'$
- Therefore
  - $F_0 = I, F_1 = P$
    - $\langle F_0, F_1 \rangle$ is OARS
L-IC3 - Initialization

Initialization of L-IC3:
• \(< F_0, F_1 >\) - same as IC3 (OARS)
  - \( F_i \land T \Rightarrow F'_{i+1} \)

• \(< U_0 >\)
  \( U_0 = \text{vars}(P) \)
L-IC3 Iteration

- Initialize $F_{k+1}$ to $\mathcal{P}$
- Initialize $U_{k+1}$ to $U_k$
- Same problem, the sequence may not be an OARS
Abstract Counterexample

\[ F_i \wedge T_i \wedge s' \]

\[ F_k \wedge T_k \wedge \neg P' \]
Abstract counterexample

• An abstract counterexample is a sequence of abstract states $s_0, \ldots, s_j$, where
  • For each $i$, $(s_i, s_{i+1}) \models T_i$
  • $s_i \cap F_i \neq \emptyset$
  • $s_j \cap \neg P \neq \emptyset$
Check Spuriousness with IC3

- An abstract CEX of length k+1 exists
- Use an IC3 iteration with the concrete T
- If a real CEX exists, it will be found

IC3 works on the already computed sequence <F₀,...,Fₖ+₁>
IC3 - Iteration k=2 (reminder)

• New iteration, check $F_2 \land T \land \neg P'$
  - If satisfiable, get s that can reach $\neg P$
  - Now check if s can be reached from $F_1$ by $F_1 \land T \land s'$
  - If it can be reached, get t and try to block it
IC3 - Iteration k=2 (reminder)

• To block \( t \), check \( F_0 \land T \land t' \)
  - If satisfiable, a CEX
  - If not, \( t \) is blocked, get a “new” \( t \) by \( F_1 \land T \land s' \)
  - If it can be reached, get \( t^* \) and try to block it
IC3 – Iteration (reminder)

• Given an OARS \( \langle F_0, F_1, \ldots, F_k \rangle \), define \( F_{k+1} = P \)
• Apply a backward search
  - Find predecessor \( s \) in \( F_k \) that can reach a bad state
    • Check \( F_k \land T \land \neg P' \)
  - If none exists \( (F_k \land T \Rightarrow P') \), move to next iteration
  - If exists, try to find a predecessor \( t \) to \( s \) in \( F_{k-1} \)
    • \( (F_{k-1} \land T \land s') \)
  - If none exists \( (F_{k-1} \land T \Rightarrow \neg s') \), \( s \) is removed from \( F_k \)
    • \( F_k = F_k \land \neg s \)
    • Actually a generalization \( c \) of \( \neg s \) (\( c \Rightarrow \neg s \)) is removed
  - Otherwise: Recur on \( (t, F_{k-1}) \)
• If we can reach \( I \), a CEX exists
Check Spuriousness (2)

• If no real CEX exists:
  - Get a *strengthened* sequence
    \[ <F_{r0}, F_{r1}, \ldots, F_{rk+1}> \]
    • Strengthening by IC3 algorithm
  - The strengthened sequence is an OARS
  - Strengthening eliminates all (real) CEXs of length \( k+1 \)

Strengthening removes states from \( F_i \)
But adds variables to \( F_i \) (due to using \( T \)
Lazy Abstraction Refinement

• If no real CEX is found by (concrete) IC3 even though (abstract) L-IC3 strengthening failed
  - Abstraction is too coarse

Explanation:

• $F_i \land T_i \land t_a'$ could not be made unsatisfiable but
• $F_i^r \land T \land t'$ is unsatisfiable
  - For each $t < t_a$
Lazy Abstraction Refinement

• Refine the sequence \(<U_1, U_2, ..., U_{k+1}>\) as follows:

• Since \(F_{r_i} \land T \Rightarrow F_{r_{i+1}}'\) (it is OARS)
  - \(F_{r_i} \land T \land \neg F_{r_{i+1}}'\) is unsatisfiable
  - Use the UnSAT Core to add visible variables
    • \(U_{r_{i+1}} = U_{i+1} \cup U_{\text{core}_i}\)

• Now \(F_{r_i} \land T_i \land \neg F_{r_{i+1}}'\) is unsat

• “Lazy” – different sets of variables at different time frames
Lazy Abstraction Refinement

- Variables added to $U^r_i$ are also added to all $U^r_j$, $j>i$, in order to maintain monotonicity of abstraction sequence

- The additional variables do not harm the unsatisfiability of $F^r_j \land T_j \land \neg F'^r_{j+1}$
• Before proceeding to the next step: check for fixpoint on $\langle F_0, \ldots, F_k \rangle$
Incrementality

• The concrete IC3 iteration works on the already computed sequence \(<F_0,F_1,\ldots,F_{k+1}>\)

• At the end of refinement, L-IC3 continues from iteration \(k+2\)
## Experiments - Laziness

| Test | #Vars | #TF | #AV | #T | #AV | #TF | #AV | #TF | #AV | #TF | #AV | #TF |
|------|-------|-----|-----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Ind 2 | 5693  | 7-1 | 31  | 8  | 42  | 9   | 51  | 10-14 | 54 |
| Ind 3 | 11866 | 1   | 323 | 2  | 647 | 3   | 686 | 4   | 699 | 5   | 705 |
| Ind 5 | 3854  | 1   | 428 | 2  | 453 | 3   | 495 | 4   | 499 | 5   | 503 |
|      | 6     | 7   | 560 | 7  | 574 | 8   | 657 | 9-11 | 577 |
Summary

• Lazy abstraction algorithm for hardware model checking
• Abstraction-Refinement in L-IC3 is done incrementally
• We compared our method (L-IC3) to Bradley’s method (IC3)
  - Up to two orders of magnitude runtime improvement
Conclusions

• L-IC3 combines two approaches to fight the state-explosion problem
• L-IC3 exposes and exploits the abstraction, implicit in IC3
The End