Towards Synthesis from Assume-Guarantee Contracts involving Infinite Theories: A Preliminary Report

FormaliSE 2016

Andreas Katis [1] Andrew Gacek [2] Michael W. Whalen [1]

[1]Department of Computer Science and Engineering, University of Minnesota

[2]Rockwell Collins Advanced Technology Center

May 15, 2016

Outline

Motivation: Solving the Architectural Analysis Problem

- Critical embedded systems development
- Safety properties for infinite state reactive systems
- “Does there exist an implementation for the given requirements?”
  “Is the given specification (**contract**) ***realizable***?”
- Previous work: Gacek, Andrew, et al. "Towards realizability checking of contracts using theories." NASA Formal Methods. Springer International Publishing, 2015. 173-187.

Motivation: Solving the Architectural Analysis Problem

- Critical embedded systems development
- Safety properties for infinite state reactive systems
- “Does there exist an implementation for the given requirements?”
  “Is the given specification (**contract**) ***realizable***?”
- Previous work: Gacek, Andrew, et al. "Towards realizability checking of contracts using theories." NASA Formal Methods. Springer International Publishing, 2015. 173-187.
- **Current goal:** “Can we synthesize implementations from realizable requirements?”

Assume-Guarantee Contracts

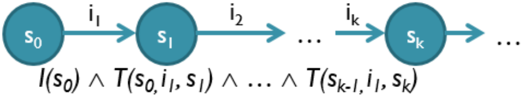
- Assumptions A: Constraints on the component's input
- Guarantees G: Constraints on the output
- Is the Contract (A,G) realizable?

Assume-Guarantee Contracts

- Assumptions A: Constraints on the component's input
- Guarantees G: Constraints on the output
- Is the Contract (A,G) realizable? YES
- Not realizable if we remove the assumption

Definitions

- Systems are defined in terms of *inputs* and *states*, ranged over by variables i and s.
- A symbolic transition system is defined: (I, T)



- A contract is a pair (A, G) with
  Assumptions $A : (state \times input) \rightarrow bool$
  Guarantees G: $\begin{cases} G_I : state \rightarrow bool \\ G_T : (state \times input \times state) \rightarrow bool \end{cases}$

Definitions

Definition (Viable)

$$Viable(s) = \forall i . A(s, i) \Rightarrow \exists s' . G_T(s, i, s') \land Viable(s')$$

Definition (Realizable Contract)

$$\exists s . G_I(s) \land Viable(s)$$

Definition (Synthesized Implementation)

A synthesized implementation is a witness of the contract's realizability

Algorithm

Definition (Finite Viability)

A state s is viable for n steps, written $Viable_n(s)$ if G_T can keep responding to valid inputs for at least n steps.

$$\begin{aligned}&\forall i_1.A(s,i_1) \Rightarrow \exists s_1.G_T(s,i_1,s_1)\wedge\\ &\qquad \forall i_2.A(s_1,i_2) \Rightarrow \exists s_2.G_T(s_1,i_2,s_2) \wedge \ldots \wedge\\ &\qquad\qquad \forall i_n.A(s_{n-1},i_n) \Rightarrow \exists s_n.G_T(s_{n-1},i_n,s_n)\end{aligned}$$

Definition (One-step Extension)

A state s is extendable after n steps, written $Extend_n(s)$ if any valid path of length n from s can be extended in response to any input.

$$\begin{aligned}&\forall i_1,s_1,\ldots,i_n,s_n.\\ &\qquad A(s,i_1) \wedge G_T(s,i_1,s_1) \wedge \ldots \wedge A(s_{n-1},i_n) \wedge G_T(s_{n-1},i_n,s_n) \Rightarrow\\ &\qquad\qquad \forall i.A(s_n,i) \Rightarrow \exists s'.G_T(s_n,i,s')\end{aligned}$$

Algorithm

- Checking Algorithm: Find n such that both checks are true:

$$BaseCheck(n) = \exists s. G_I(s) \wedge Viable_n(s)$$

$$ExtendCheck(n) = \forall s. Extend_n(s)$$

- $2n$ quantifier alternations in *BaseCheck*
    - Extremely difficult SMT problem
    - Solvers fail very quickly

- Instead, use an approximation:

$$BaseCheck'(n) = \forall k \leq n. (\forall s. G_I(s) \Rightarrow Extend_k(s))$$

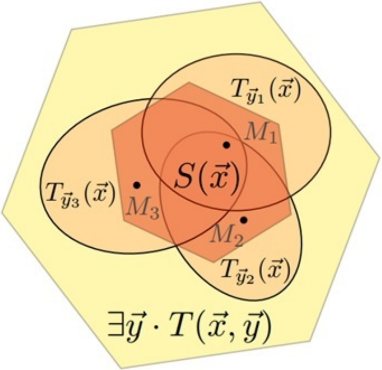
Goal

- Can we effectively use our method to solve the synthesis problem?
- Problem: SMT-solvers cannot be used directly (nested quantifiers)
- Solution: **AE-VAL: Horn-based Skolemizer for $\forall\exists$ formulas**
  Fedyukovich, Grigory, Arie Gurfinkel, and Natasha Sharygina."Automated discovery of simulation between programs." Logic for Programming, Artificial Intelligence, and Reasoning. Springer Berlin Heidelberg, 2015.

AE-VAL Skolemizer for ∀∃ formulas

- $S(\vec{x}) \Rightarrow \exists \vec{y}. T(\vec{x}, \vec{y})$
- Model Based Projection to extract Skolem relations
- Linear Integer Arithmetic

$$Sk_{\vec{y}}(\vec{x}, \vec{y}) \equiv \begin{cases} \phi_{\vec{y}_1}(\vec{x}, \vec{y}) & \text{if } I_1(\vec{x}) \\ \phi_{\vec{y}_2}(\vec{x}, \vec{y}) & \text{else if } I_2(\vec{x}) \\ \cdots & \cdots \\ \phi_{\vec{y}_n}(\vec{x}, \vec{y}) & \text{else } I_n(\vec{x}) \end{cases}$$



Using AE-VAL for Synthesis

- Two separate phases for *BaseCheck'* and *ExtendCheck*

$$BaseCheck'(n) = \forall k \leq n.(\forall s.G_I(s) \Rightarrow Extend_k(s))$$

$$ExtendCheck(n) = \forall s.Extend_n(s)$$

$Extend_n(s) =$

$$\forall i_1, s_1, \ldots, i_n, s_n.$$
$$A(s, i_1) \wedge G_T(s, i_1, s_1) \wedge \ldots \wedge A(s_{n-1}, i_n) \wedge G_T(s_{n-1}, i_n, s_n) \Rightarrow$$
$$\forall i.A(s_n, i) \Rightarrow \exists s'.G_T(s_n, i, s')$$

$$\Downarrow$$

$$\forall i_1, s_1, \ldots, i_n, s_n, i.$$
$$A(s, i_1) \wedge G_T(s, i_1, s_1) \wedge \ldots \wedge$$
$$A(s_{n-1}, i_n) \wedge G_T(s_{n-1}, i_n, s_n) \wedge A(s_n, i) \Rightarrow$$
$$\exists s'.G_T(s_n, i, s')$$

Synthesis Algorithm

```
assign_GI_witness_to_S;
update_array_history;

// Perform bounded 'base check' synthesis
read_inputs;
base_check'_1_solution;
update_array_history;
...
read_inputs;
base_check'_k_solution;
update_array_history;

// Perform recurrence from 'extends' check
while(1) {
 read_inputs;
 extend_check_k_solution;
 update_array_history;
}
```

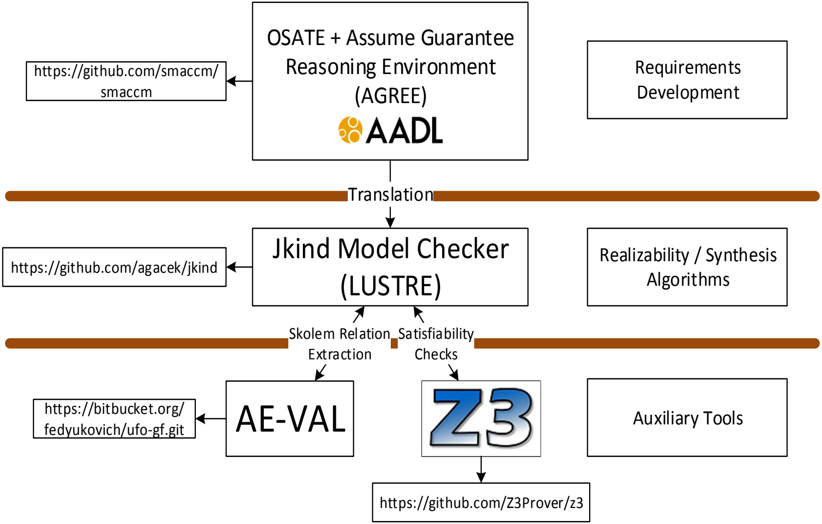
1. Construct history arrays for variables in I and S.
2. Initialize variable values (0th element of array) using G_I
3. Initialize history of length k using BaseCheck' Skolem relations
4. Use ExtendCheck's solution in a recurrence loop to define the next-step values

Skolem relation example

```
ite([&&
    $defs__rising_edge~1.Mode_Control_Impl_Instance__signal$0
    !($Mode_Control_Impl_Instance__seconds_to_cook$0>=0)
    !$defs__initially_true~0.Mode_Control_Impl_Instance__result$0
  ], [&&
    $Mode_Control_Impl_Instance__is_setup$0
    $defs__rising_edge~1.Mode_Control_Impl_Instance__re$0
    !$Mode_Control_Impl_Instance__is_cooking$0
    $defs__rising_edge~1.Mode_Control_Impl_Instance__signal$0
    !$_TOTAL_COMP_HIST$0
    !$_SYSTEM_ASSUMP_HIST$0
    !$Mode_Control_Impl_Instance__is_suspended$0
    !$Mode_Control_Impl_Instance__is_running$0
    !$defs__rising_edge~0.Mode_Control_Impl_Instance__re$0
    !$defs__initially_true~0.Mode_Control_Impl_Instance__b$0
    !$defs__initially_true~0.Mode_Control_Impl_Instance__result$0
    !$defs__rising_edge~2.Mode_Control_Impl_Instance__re$0
    !$defs__rising_edge~2.Mode_Control_Impl_Instance__signal$0
  ], ite([&&
      %init
      $_SYS_GUARANTEE_2$0
      !($Mode_Control_Impl_Instance__seconds_to_cook$0>=0)
      !$defs__rising_edge~1.Mode_Control_Impl_Instance__signal$0
      !$defs__initially_true~0.Mode_Control_Impl_Instance__b$0
    ], ...))
```

- This is only one of the necessary solutions to construct the implementation
- 900 lines of code
- A good intermediate representation to retranslate into any target language

Implementation



Future Work

- Extend work to Linear Real Arithmetic
- Improve transition relation representation
- Efficient translation of Lustre data-flow programs to non-minimal FSMs
- Formal verification of algorithm
- Improve realizability algorithm using an inductive invariant generation approach (Property Directed Reachability)
- Possible obstacle + Research subject : Mapping infinite to equivalent finite implementations

Thank You!

Definition (Reachable with respect to assumptions)

A state of (I, T) is reachable with respect to A if there exists a path starting in an initial state and eventually reaching s such that all transitions are satisfying the assumptions

$$Reachable_A(s) = I(s) \lor \exists s_{prev}, i.\ Reachable_A(s_{prev}) \land A(s_{prev}, i) \land T(s_{prev}, i, s)$$

Definition (Realization)

A transition system (I, T) is a realization of the contract $(A, (G_I, G_T))$ when the following conditions hold

- $\forall s.\ I(s) \Rightarrow G_I(s)$
- $\forall s, i, s'.\ \mathrm{Reachable}_A(s) \land A(s, i) \land T(s, i, s') \Rightarrow G_T(s, i, s')$
- $\exists s.\ I(s)$
- $\forall s, i.\ \mathrm{Reachable}_A(s) \land A(s, i) \Rightarrow \exists s'.\ T(s, i, s')$