Control-flow Guided Property Directed Reachability for Imperative Synchronous Programs

Xian Li

Supervisor: Prof. Dr. Klaus Schneider

Embedded Systems Chair
Department of Computer Science
University of Kaiserslautern, Germany

Synchron 2016 at Bamberg, December 5 – 9, 2016
1. Motivation

2. Property Directed Reachability

3. Control-flow Guided PDR for Imperative Synchronous Programs
Outline

1. Motivation

2. Property Directed Reachability

3. Control-flow Guided PDR for Imperative Synchronous Programs
Formal Verification of Synchronous Hardware Circuits

- PDR: a very efficient verification method based on induction
Formal Verification of Synchronous Programs

- PDR: a very efficient verification method based on induction

**Synchronous Programs**
```plaintext
module M(event bool ?a, ?b, o1, o2) {
    loop {
        l1: pause;
        if(o1 & (a | b)) {
            emit(o2);
            l2: await(a);
        }
    }
}
```

**Synchronous Circuits**

![Synchronous Circuits Diagram]
Imperative Synchronous Programs

Imperative Synchronous Languages: e.g. Quartz

- macro steps: consumption of one logical time unit
- micro steps: no logical time consumption

⇒ synchronous reactive model of computation

Control-flow Information

- not needed for synthesis
- useful for formal verification
**Goals**

**Target:** Safety Property Verification of Imperative Synchronous Programs

- PDR: relies on good estimation of the reachable states

**Our Heuristic:** Improve it by Exploiting Control-flow Information

- modify transition relation to generate less counterexamples to induction (CTIs) by reachable control-flow states computation
  - linear-time static analysis
  - symbolic reachability analysis
- identify CTIs in $\mathcal{K}$
  simpler unreachability tests in $\mathcal{K}^{cf}$
- generalize CTIs to narrow the reachable state approximations
  if $\mathcal{C}$ is unreachable, then generalize $\neg C'$ instead of $\neg C$:
  $C' := C|_{\neg cf}$ obtained from omitting the dataflow literals in $C$
<table>
<thead>
<tr>
<th>Outline</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Motivation</td>
</tr>
<tr>
<td></td>
<td>2. Property Directed Reachability</td>
</tr>
<tr>
<td></td>
<td>3. Control-flow Guided PDR for Imperative Synchronous Programs</td>
</tr>
</tbody>
</table>
Safety Property Verification

Target: Prove $\Phi$ is valid w.r.t. $\mathcal{K}$

- a state transition system: $\mathcal{K} := (\mathcal{V}, \mathcal{I}, \mathcal{T})$
- a safety property: $\Phi$
- $\Phi$ holds on all reachable states of $\mathcal{K}$

$\Phi$ is inductive w.r.t. $\mathcal{K}$

- induction base: $\Phi$ holds in all initial states
- induction step: $\Phi$-states have no successor violating $\Phi$

module CfSeq(){
    p1: pause;
    p2: pause;
}

\[
\begin{align*}
\mathcal{V} & := \{\text{run}, \text{p1}, \text{p2}\} \\
\mathcal{I} & := \neg(\text{run} \lor \text{p1} \lor \text{p2}) \\
\mathcal{T} & := \text{next}(\text{run}) \leftrightarrow \text{true} \\
& \quad \land (\text{next}(\text{p1}) \leftrightarrow \neg \text{run}) \\
& \quad \land (\text{next}(\text{p2}) \leftrightarrow \text{p1})
\end{align*}
\]

$\Phi := \neg(\text{p1} \land \text{p2})$
Safety Property Verification by Induction

Target: Prove $\Phi$ is valid w.r.t. $\mathcal{K}$

- a state transition system: $\mathcal{K} := (\mathcal{V}, \mathcal{I}, \mathcal{T})$
- a safety property: $\Phi$
- $\Phi$ holds on all reachable states of $\mathcal{K}$

$\Phi$ is inductive w.r.t. $\mathcal{K}$

- induction base: $\Phi$ holds in all initial states
- induction step: $\Phi$-states have no successor violating $\Phi$
Safety Property Verification by Induction

Target: Prove $\Phi$ is valid w.r.t. $\mathcal{K}$

- a state transition system: $\mathcal{K} := (\mathcal{V}, \mathcal{I}, \mathcal{T})$
- a safety property: $\Phi$
- $\Phi$ holds on all reachable states of $\mathcal{K}$

$\Phi$ is inductive w.r.t. $\mathcal{K}$

- induction base: $\Phi$ holds in all initial states
- induction step: $\Phi$-states have no successor violating $\Phi$
Motivation

Property Directed Reachability

Control-flow Guided PDR for Imperative Synchronous Programs

Property Directed Reachability

PDR method constructs a sequence of clause sets $\Psi_0, \ldots, \Psi_k$ that overapproximate the states reachable in $0, \ldots, k$ steps.

- incremental induction: extend the sequence $\Psi_0, \ldots, \Psi_k$
- unreachability checking: CTI identification and generalization

### Reachable States

<table>
<thead>
<tr>
<th>State</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>{}</td>
</tr>
<tr>
<td>$s_1$</td>
<td>{p2}</td>
</tr>
<tr>
<td>$s_2$</td>
<td>{p1}</td>
</tr>
<tr>
<td>$s_3$</td>
<td>{p1,p2}</td>
</tr>
<tr>
<td>$s_4$</td>
<td>{run}</td>
</tr>
<tr>
<td>$s_5$</td>
<td>{run,p2}</td>
</tr>
<tr>
<td>$s_6$</td>
<td>{run,p1}</td>
</tr>
<tr>
<td>$s_7$</td>
<td>{run,p1,p2}</td>
</tr>
</tbody>
</table>

$\Phi$ holds $\wedge$ $\Phi$ doesn't hold

Reachable States
Property Directed Reachability

PDR method constructs a sequence of clause sets $\Psi_0, \ldots, \Psi_k$ that overapproximate the states reachable in $0, \ldots, k$ steps.

- Incremental induction: extend the sequence $\Psi_0, \ldots, \Psi_k$
- Unreachability checking: CTI identification and generalization
Property Directed Reachability

PDR method constructs a sequence of clause sets $\Psi_0, \ldots, \Psi_k$ that overapproximate the states reachable in $0, \ldots, k$ steps.

- incremental induction: extend the sequence $\Psi_0, \ldots, \Psi_k$
- unreachability checking: CTI identification and generalization
Outline

1. Motivation

2. Property Directed Reachability

3. Control-flow Guided PDR for Imperative Synchronous Programs
Main Idea I: Modify Transition Relation to generate less CTIs

Original Transition Relation:

\[ \psi_{k-1} \]

\[ \psi_\phi \]

s0: \{\}

s1: \{p2\}

s6: \{run,p1\}

s2: \{p1\}

s5: \{run,p2\}

s7: \{run,p1,p2\}

s4: \{run\}

s3: \{p1,p2\}

Reachable States

\[ \phi \text{ holds} \]

\[ \phi \text{ doesn't hold} \]

s2 has successor s7 violating \( \phi \)

Enhanced Transition Relation:

\[ \psi_{k-1} \]

\[ \psi_\phi \]

s0: \{\}

s1: \{p2\}

s6: \{run,p1\}

s2: \{p1\}

s5: \{run,p2\}

s7: \{run,p1,p2\}

s4: \{run\}

s3: \{p1,p2\}

Reachable States

\[ \phi \text{ holds} \]

\[ \phi \text{ doesn't hold} \]

s2 has no successor

\( \Rightarrow \) remove transitions from unreachable states by control-flow invariants
Control-flow Invariants by **static** Analysis

Control-flow can never be active at both substatements of sequences and conditional statements:

```plaintext
module CfSeq(){
    p1: pause;
    p2: pause;
}
¬(p1 ∧ p2)
```
Control-flow can never be active at both substatements of sequences and conditional statements:

```plaintext
module Ite(){
    mem bool i;
    if (i) {
        p1: pause;
    } else {
        q1: pause;
    }
}

¬(p1 ∧ q1)
```
Control-flow Invariants by static Analysis

Control-flow can never be active at both substatements of sequences and conditional statements:

\[ \neg(p_1 \land p_2) \land \neg(q_1 \land q_2) \land \neg((p_1 \lor p_2) \land (q_1 \lor q_2)) \]

module CfIte(){
    mem bool i;
    if (i) {
        p1: pause;
        p2: pause;
    } else {
        q1: pause;
        q2: pause;
    }
}
module CfIte() {
    mem bool i;
    if (i) {
        p1: pause;
        p2: pause;
    } else {
        q1: pause;
        q2: pause;
    }
}
module CfIte()
{
    mem bool i;
    if (i) {
        p1: pause;
        p2: pause;
    } else {
        q1: pause;
        q2: pause;
    }
}

Enhanced Transition Relation:

with control-flow invariant by static analysis:
\(\neg(p1 \land p2) \land \neg(q1 \land q2) \land \neg((p1 \lor p2) \land (q1 \lor q2))\)
Control-flow Invariants by **symbolic** Analysis

```java
module CfPar(){
    {
        p1: pause;
        p2: pause;
    } ||
    {
        q1: pause;
        q2: pause;
    }
}
```

Original Transition Relation:
Control-flow Invariants by **symbolic** Analysis

module CfPar()
{
    p1: pause;
    p2: pause;
} ||
{
    q1: pause;
    q2: pause;
}

Enhanced Transition Relation:

\[-(p_1 \land p_2) \land -(q_1 \land q_2)\]
Control-flow Invariants by symbolic Analysis

Symbolic traversal of the state space of the control-flow system:

```plaintext
module CfPar(){
{
    p1: pause;
    p2: pause;
} ||
{
    q1: pause;
    q2: pause;
}
}

¬(p1 ∧ p2) ∧ ¬(q1 ∧ q2) ∧ ¬((p1 ∧ q2) ∨ (p2 ∧ q1))
```
Control-flow Invariants by **symbolic** Analysis

**module** CfPar(){
    {
        p1: pause;
p2: pause;
    } ||
    {
        q1: pause;
q2: pause;
    }
}

**Enhanced Transition Relation:**

\[
\neg (p_1 \land p_2) \land \neg (q_1 \land q_2) \land \neg ((p_1 \land q_2) \lor (p_2 \land q_1))
\]
Main Idea II: CTI Indentification and Generalization by Control-flows

- reachability of CTIs in $K$
- simpler unreachability tests in $K^{cf}$

- generalize CTIs to narrow the reachable state approximations
  if $C$ is unreachable, then generalize $\neg C'$ instead of $\neg C$:
  $C' := C\mid_{\forall cf}$ obtained from omitting the dataflow literals in $C$
Let $\mathcal{V} := \mathcal{V}^{\text{cf}} \cup \mathcal{V}^{\text{df}}$ and $\mathcal{K} := \mathcal{K}^{\text{cf}} \times \mathcal{K}^{\text{df}}$, with

- $\mathcal{K} = (\mathcal{V}, \mathcal{I}, \mathcal{T})$
- $\mathcal{K}^{\text{cf}} = (\mathcal{V}, \mathcal{I}^{\text{cf}}, \mathcal{T}^{\text{cf}})$
- $\mathcal{K}^{\text{df}} = (\mathcal{V}, \mathcal{I}^{\text{df}}, \mathcal{T}^{\text{df}})$
Transition Systems of a Synchronous Program

Let \( V := V^{cf} \cup V^{df} \) and \( \mathcal{K} := \mathcal{K}^{cf} \times \mathcal{K}^{df} \), with

- \( \mathcal{K} = (V, I, \mathcal{T}) \)
- \( \mathcal{K}^{cf} = (V, I^{cf}, \mathcal{T}^{cf}) \)
- \( \mathcal{K}^{df} = (V, I^{df}, \mathcal{T}^{df}) \)

unreachability of CTIs in \( \mathcal{K} \) can be proved by unreachability in \( \mathcal{K}^{cf} \)
CTI Indentification by Control-flows

Let $\mathcal{V} := \mathcal{V}^{\text{cf}} \cup \mathcal{V}^{\text{df}}$ and $\mathcal{K} := \mathcal{K}^{\text{cf}} \times \mathcal{K}^{\text{df}}$, with

$\mathcal{K} = (\mathcal{V}, \mathcal{I}, \mathcal{T})$

$\mathcal{K}^{\text{cf}} = (\mathcal{V}, \mathcal{I}^{\text{cf}}, \mathcal{T}^{\text{cf}})$

$\mathcal{K}^{\text{df}} = (\mathcal{V}, \mathcal{I}^{\text{df}}, \mathcal{T}^{\text{df}})$

unreachability of CTIs in $\mathcal{K}$ can be proved by unreachability in $\mathcal{K}^{\text{cf}}$

reachability of CTIs in $\mathcal{K}$

simpler unreachability tests in $\mathcal{K}^{\text{cf}}$
CTI Generalization by Control-flows

Let $\mathcal{V} := \mathcal{V}^{\text{cf}} \cup \mathcal{V}^{\text{df}}$ and $\mathcal{K} := \mathcal{K}^{\text{cf}} \times \mathcal{K}^{\text{df}}$, with

- $\mathcal{K} = (\mathcal{V}, \mathcal{I}, \mathcal{T})$
- $\mathcal{K}^{\text{cf}} = (\mathcal{V}, \mathcal{I}^{\text{cf}}, \mathcal{T}^{\text{cf}})$
- $\mathcal{K}^{\text{df}} = (\mathcal{V}, \mathcal{I}^{\text{df}}, \mathcal{T}^{\text{df}})$

unreachability in $\mathcal{K}^{\text{cf}}$ is independent on the dataflows
CTI Generalization by Control-flows

Let $\mathcal{V} := \mathcal{V}^{\text{cf}} \cup \mathcal{V}^{\text{df}}$ and $\mathcal{K} := \mathcal{K}^{\text{cf}} \times \mathcal{K}^{\text{df}}$, with

- $\mathcal{K} = (\mathcal{V}, \mathcal{I}, \mathcal{T})$
- $\mathcal{K}^{\text{cf}} = (\mathcal{V}, \mathcal{I}^{\text{cf}}, \mathcal{T}^{\text{cf}})$
- $\mathcal{K}^{\text{df}} = (\mathcal{V}, \mathcal{I}^{\text{df}}, \mathcal{T}^{\text{df}})$

unreachability in $\mathcal{K}^{\text{cf}}$ is independent on the dataflows

- generalize CTIs to narrow the reachable state approximations
  if $C$ is unreachable, then generalize $\neg C'$ instead of $\neg C$:
  $C' := C|_{\mathcal{V}^{\text{cf}}}$ obtained from omitting the dataflow literals in $C$
module ITELoop() {
  [N]bool i;
  i[0] = true;
  if (!i[0]) {
    loop{
      p1: pause;
      i[0] = false;
      p2: pause;
    }
  }
}

The set of boolean variables of module ITELoop

\[ \mathcal{V}_N := \{i[0], \ldots, i[N-1]\} \cup \{p1, p2, run\} \]

⇒ reduce at most \(2^{N+3}\) to \(2^3\) times relative inductiveness reasoning
Summary

Control-flow Guided PDR for Imperative Synchronous Programs

- modify transition relation to generate less CTIs by reachable control-flow states computation
  - linear-time static analysis
  - symbolic reachability analysis

- identify CTIs in $K$
  simpler unreachability tests in $K^\text{cf}$

- generalize CTIs to narrow the reachable state approximations
  if $C$ is unreachable, then generalize $\neg C'$ instead of $\neg C$:
  $C' := C|_{\forall \text{ cf}}$ obtained from omitting the dataflow literals in $C$