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

Table of Contents

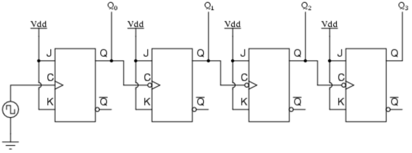
Outline

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

Synchronous Circuits



$\models \varphi$

Formal Verification of Synchronous Programs

- PDR: a very efficient verification method based on induction

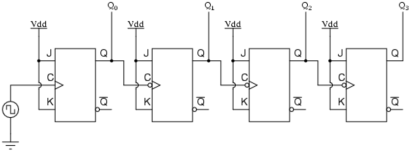
Synchronous Programs

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

$\models \varphi'$

synthesize

Synchronous Circuits



$\models \varphi$

Imperative Synchronous Programs

Imperative Synchronous Languages: e.g. Quartz

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

$\Rightarrow$ 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 Expoiting 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
- indentify CTIs in $\mathcal{K}$
  simpler unreachability tests in $\mathcal{K}^{\mathsf{cf}}$
- generalize CTIs to narrow the reachable state approximations
  if $\mathcal{C}$ is unreachable, then generalize $\neg\mathcal{C}'$ instead of $\neg\mathcal{C}$:
  $\mathcal{C}' := \mathcal{C}_{|\mathcal{V}^{\mathsf{cf}}}$ obtained from omitting the dataflow literals in $\mathcal{C}$

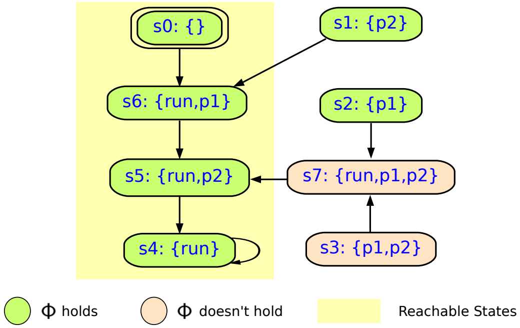
Outline

1. Motivation
2. Property Directed Reachability
3. Control-flow Guided PDR for Imperative Synchronous Programs

Safety Property Verification

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

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




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

$$
\begin{aligned}
\mathcal{V} &:= \{\mathrm{run}, \mathrm{p1}, \mathrm{p2}\} \\
\mathcal{I} &:= \neg(\mathrm{run} \lor \mathrm{p1} \lor \mathrm{p2}) \\
\mathcal{T} &:= \quad \mathbf{next}(\mathrm{run}) \leftrightarrow \mathbf{true} \\
&\quad \land (\mathbf{next}(\mathrm{p1}) \leftrightarrow \neg\mathrm{run}) \\
&\quad \land (\mathbf{next}(\mathrm{p2}) \leftrightarrow \mathrm{p1})
\end{aligned}
$$

$$\Phi := \neg(\mathrm{p1} \land \mathrm{p2})$$

Safety Property Verification by Induction

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

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

Φ is inductive w.r.t. $\mathcal{K}$

- induction base: Φ holds in all initial states
- induction step: Φ-states have no successor violating Φ

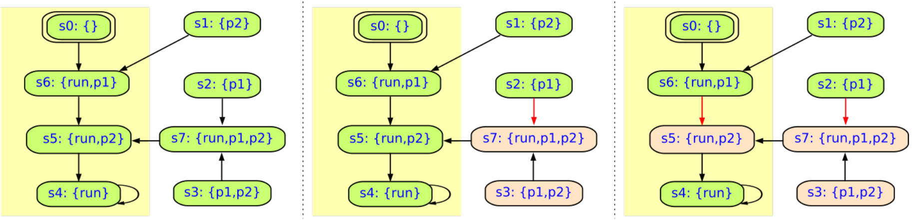
Safety Property Verification by Induction

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

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

Φ is inductive w.r.t. $\mathcal{K}$

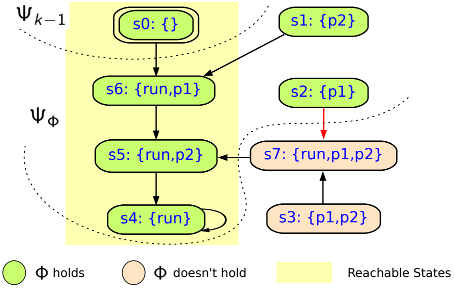
- induction base: Φ holds in all initial states
- induction step: Φ-states have no successor violating Φ



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.

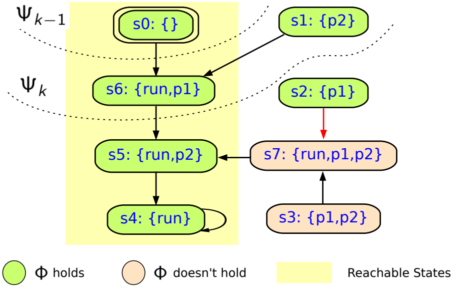
- incremental induction: extend the sequence $\Psi_0, \ldots, \Psi_k$
- unreachability checking: CTI indentification and generalization



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.

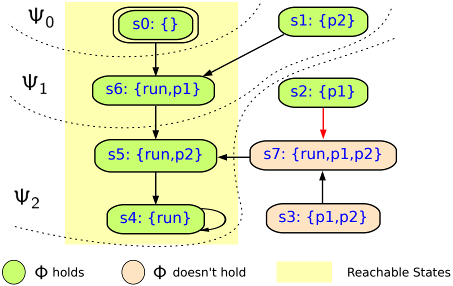
- incremental induction: extend the sequence $\Psi_0, \ldots, \Psi_k$
- unreachability checking: CTI indentification and generalization



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 indentification and generalization



Outline

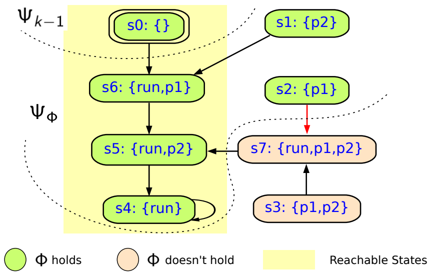
1. Motivation

2. Property Directed Reachability

3. Control-flow Guided PDR for Imperative Synchronous Programs

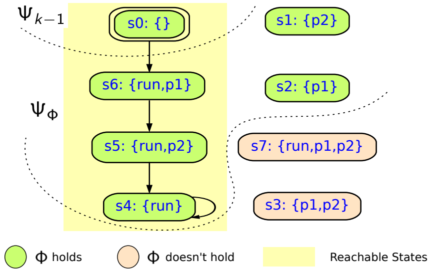
Main Idea I: Modify Transition Relation to generate less CTIs

Original Transition Relation:



s_2 has successor s_7 violating Φ

Enhanced Transition Relation:



s_2 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:

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

$$\neg(\mathrm{p1} \land \mathrm{p2})$$

Control-flow Invariants by static Analysis

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

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

$$\neg(p1 \wedge q1)$$

Control-flow Invariants by static Analysis

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

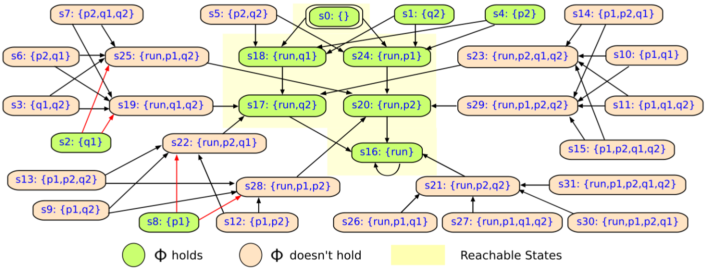
```
module CfIte(){
  mem bool i;
  if (i) {
    p1: pause;
    p2: pause;
  } else {
    q1: pause;
    q2: pause;
  }
}
```

$$\neg(\mathrm{p1} \wedge \mathrm{p2}) \wedge \neg(\mathrm{q1} \wedge \mathrm{q2}) \wedge \neg((\mathrm{p1} \vee \mathrm{p2}) \wedge (\mathrm{q1} \vee \mathrm{q2}))$$

Control-flow Invariants by static Analysis

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

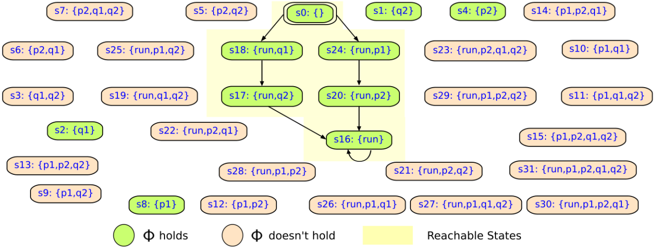
Original Transition Relation:



Control-flow Invariants by static Analysis

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

Enhanced Transition Relation:



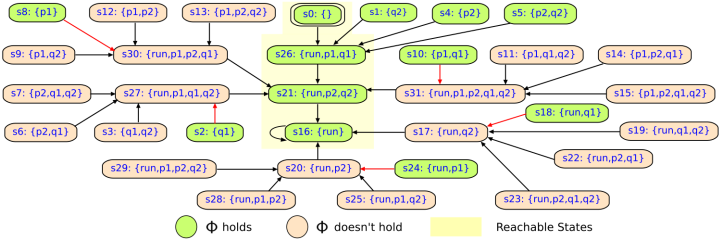
with control-flow invariant by static analysis:

$\neg(p1 \wedge p2) \wedge \neg(q1 \wedge q2) \wedge \neg((p1 \vee p2) \wedge (q1 \vee q2))$

Control-flow Invariants by symbolic Analysis

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

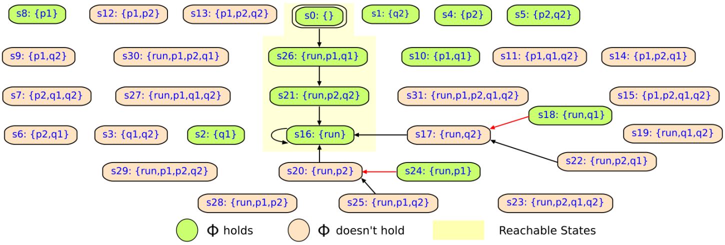
Original Transition Relation:



Control-flow Invariants by symbolic Analysis

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

Enhanced Transition Relation:



with control-flow invariant by static analysis:

$\neg(p1 \wedge p2) \wedge \neg(q1 \wedge q2)$

Control-flow Invariants by symbolic Analysis

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

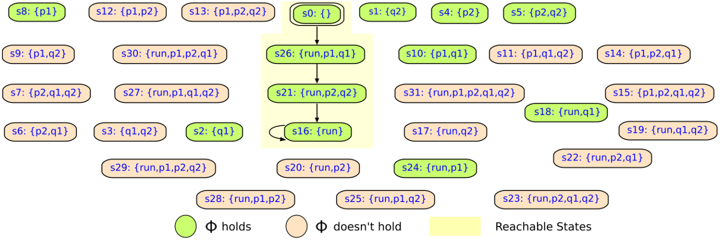
```
module CfPar(){
  {
    p1: pause;
    p2: pause;
  } ||
  {
    q1: pause;
    q2: pause;
  }
}
```

$$\neg(\mathrm{p1} \land \mathrm{p2}) \land \neg(\mathrm{q1} \land \mathrm{q2}) \land \neg((\mathrm{p1} \land \mathrm{q2}) \lor (\mathrm{p2} \land \mathrm{q1}))$$

Control-flow Invariants by symbolic Analysis

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

Enhanced Transition Relation:



with control-flow invariant by symbolic analysis:

$$\neg(p1 \wedge p2) \wedge \neg(q1 \wedge q2) \wedge \neg((p1 \wedge q2) \vee (p2 \wedge q1))$$

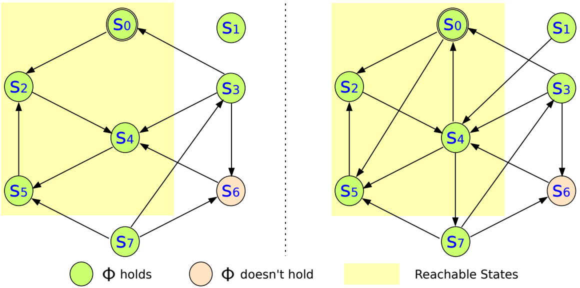
Main Idea II: CTI Indentification and Generalization by Control-flows

- reachability of CTIs in $\mathcal{K}$
  simpler unreachability tests in $\mathcal{K}^{\text{cf}}$
- generalize CTIs to narrow the reachable state approximations
  if $\mathcal{C}$ is unreachable, then generalize $\neg\mathcal{C}'$ instead of $\neg\mathcal{C}$:
  $\mathcal{C}' := \mathcal{C}_{|\mathcal{V}^{\text{cf}}}$ obtained from omitting the dataflow literals in $\mathcal{C}$

Transition Systems of a Synchronous Program

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

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

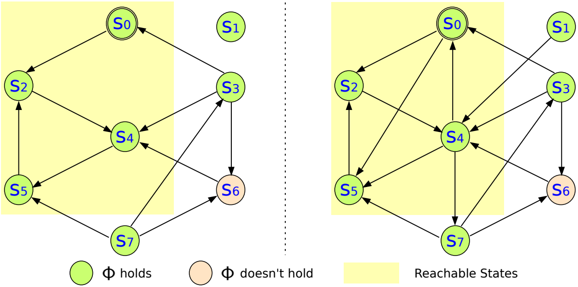


Transition Systems of a Synchronous Program

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

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

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



CTI Indentification by Control-flows

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

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

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

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

CTI Generalization by Control-flows

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

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

unreachability in $\mathcal{K}^{\mathsf{cf}}$ 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 $\mathcal{C}$ is unreachable, then generalize $\neg\mathcal{C}'$ instead of $\neg\mathcal{C}$:
  $\mathcal{C}' := \mathcal{C}_{|\mathcal{V}^{\text{cf}}}$ obtained from omitting the dataflow literals in $\mathcal{C}$

Example

```
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

$$\mathcal{V}_{\mathtt{N}} \quad := \quad \underbrace{\{\mathtt{i[0]}, \dots, \mathtt{i[N-1]}\}}_{\mathcal{V}^{\mathsf{df}}} \cup \underbrace{\{\mathtt{p1}, \mathtt{p2}, \mathtt{run}\}}_{\mathcal{V}^{\mathsf{cf}}}$$

$\Rightarrow$ reduce at most $2^{\mathtt{N}+3}$ 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 $\mathcal{K}$
  simpler unreachability tests in $\mathcal{K}^{\mathsf{cf}}$
- generalize CTIs to narrow the reachable state approximations
  if $\mathcal{C}$ is unreachable, then generalize $\neg\mathcal{C}'$ instead of $\neg\mathcal{C}$:
  $\mathcal{C}' := \mathcal{C}_{|\mathcal{V}^{\mathsf{cf}}}$ obtained from omitting the dataflow literals in $\mathcal{C}$