Causality-Based Verification of Multi-threaded Programs

Andrey Kupriyanov and Bernd Finkbeiner

Saarland University
Reactive Systems Group

August 28, 2013

Introduction

Causality

The relation between two events (the *cause* and the *effect*), where the second event is understood as a (necessary) consequence of the first.

Introduction

Causality

The relation between two events (the *cause* and the *effect*), where the second event is understood as a (necessary) consequence of the first.

In this talk

- Capturing causality by concurrent traces and their transformations
- Verification of concurrent programs based on causality
- How causality-based verification can bring exponential savings for some classes of multi-threaded programs

Introduction

Verification of Safety Properties

$$\boxed{\text{System } S} \models \mathbf{G}\, \mathit{safe}\ ?$$

Introduction

Verification of Safety Properties for **Concurrent** Systems

$$S = \boxed{P_1} \parallel \boxed{P_2} \parallel \ldots \parallel \boxed{P_N} \models \mathbf{G}\ \mathit{safe}\ ?$$

Introduction

Verification of Safety Properties for **Concurrent** Systems

$$S = \boxed{P_1} \parallel \boxed{P_2} \parallel \ldots \parallel \boxed{P_N} \models \mathbf{G}\ \mathit{safe}\ ?$$

Different flavors:

- Synchronized product of finite automata
- Communicating processes
- Multi-threaded programs

Introduction

Verification of Safety Properties for **Concurrent** Systems

$$S = \boxed{P_1} \parallel \boxed{P_2} \parallel \dots \parallel \boxed{P_N} \models \mathbf{G}\ \mathit{safe}\ ?$$

Different flavors:

- Synchronized product of finite automata
- Communicating processes
- Multi-threaded programs

Complexity

The problem is **PSPACE-complete**

Introduction

Verification of Safety Properties for **Concurrent** Systems

$$S = \boxed{P_1} \parallel \boxed{P_2} \parallel \ldots \parallel \boxed{P_N} \models \mathbf{G}\, \mathit{safe}\ ?$$

Different flavors:

- Synchronized product of finite automata
- Communicating processes
- Multi-threaded programs

Complexity

The problem is **PSPACE-complete**

Problem complexity is **robust**

- varying communication models (global/binary/shared vars)
- different sizes of the alphabet

Motivation

- Unless $P = PSPACE$, there is no scalable algorithm for the general-case concurrent verification problem
- It is easy to manually prove/disprove the correctness of many concurrent programs

Motivation

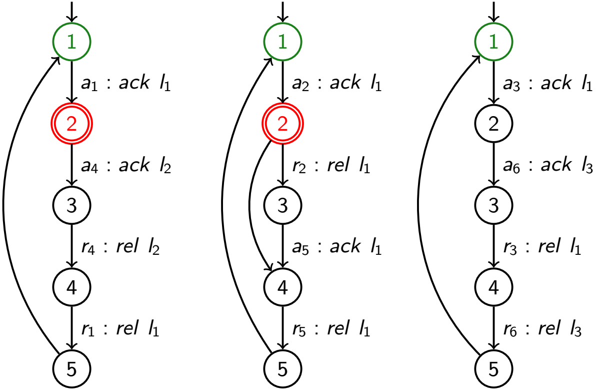
- Unless $P = PSPACE$, there is no scalable algorithm for the general-case concurrent verification problem
- It is easy to manually prove/disprove the correctness of many concurrent programs

$\Rightarrow$ Investigate:

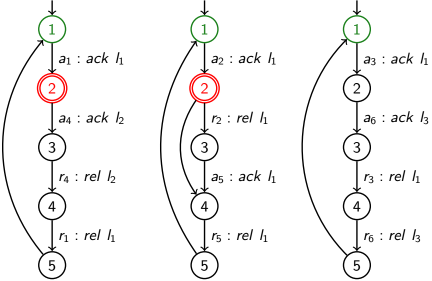
- Efficient (**polynomial**) proof techniques
- Classes of efficiently verifiable concurrent programs

Multi-threaded Programs with Locks

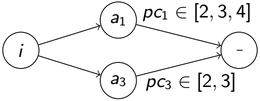
Syntax	**Semantics**
acquire l_i	$l_i = 0 \wedge l_i' = 1 \ \wedge pc' = pc + 1$
release l_i	$l_i' = 0 \ \wedge pc' = pc + 1$
if (φ) *goto* j	$(\varphi \wedge pc' = j) \ \vee (\neg\varphi \wedge pc' = pc + 1)$

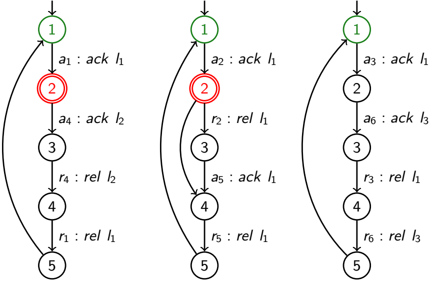


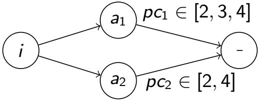
A Polynomial Proof

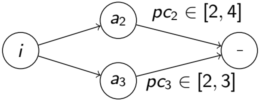


A Polynomial Proof

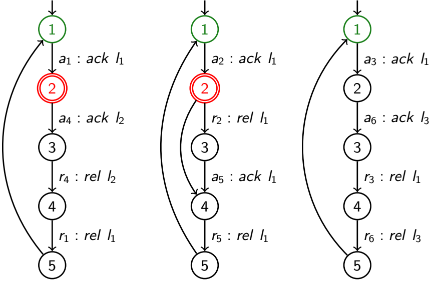


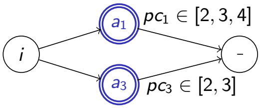




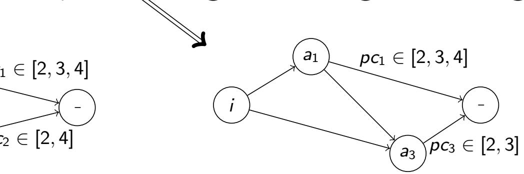


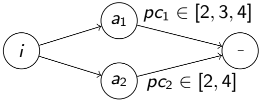
A Polynomial Proof

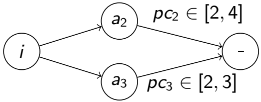




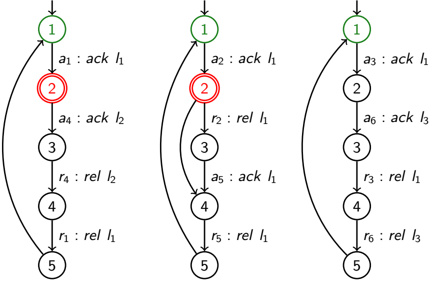
OrderSplit

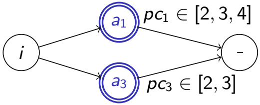






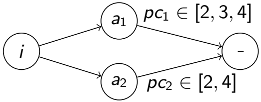
A Polynomial Proof

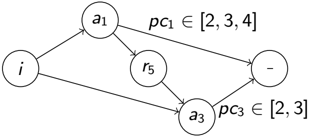


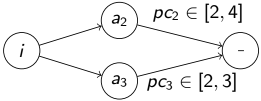


OrderSplit

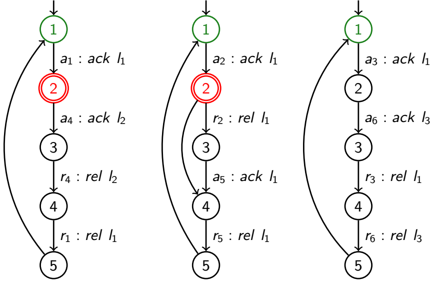
NecessaryAction $(l' = 1 \lightning l = 0)$

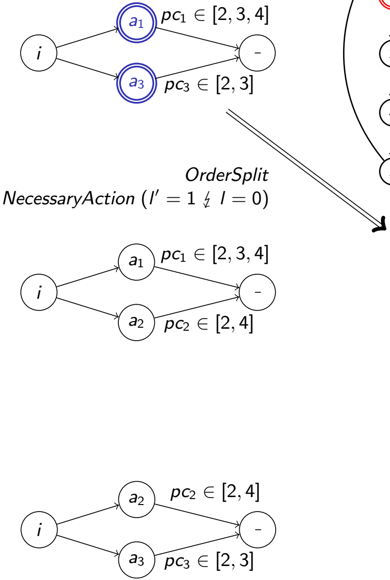


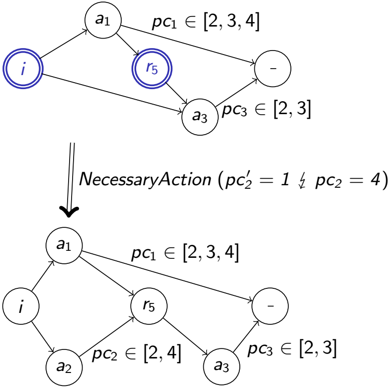




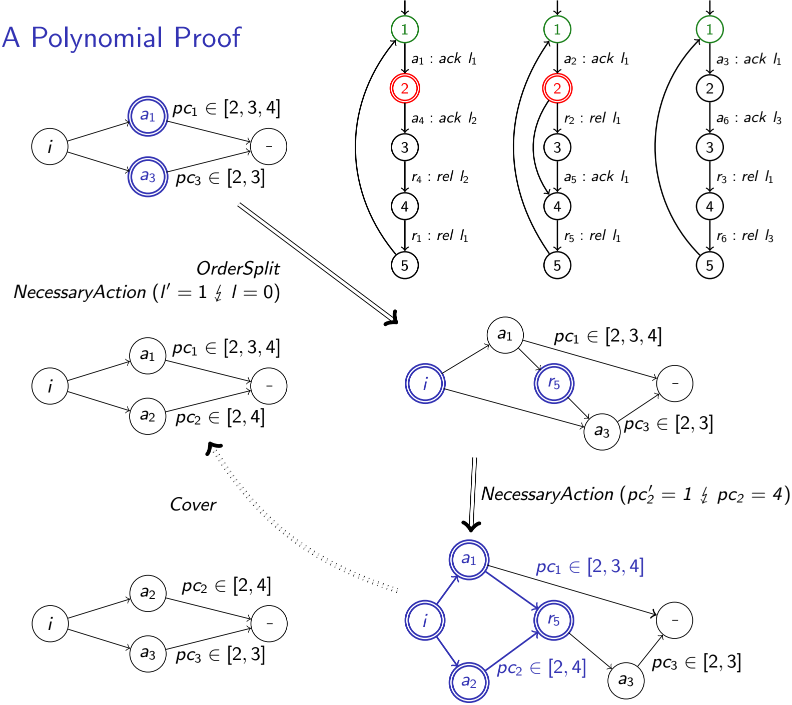
A Polynomial Proof







A Polynomial Proof



Proof Object

State

A tuple of state components $s = \langle p_1, p_2, \ldots, p_N \rangle \in |P_1| \times |P_2| \times \ldots \times |P_N|$

Proof Object

State

A tuple of state components $s = \langle p_1, p_2, \ldots, p_N \rangle \in |P_1| \times |P_2| \times \ldots \times |P_N|$

State Inclusion

$s = \langle p_1, \ldots, p_N \rangle \ \subseteq \ s' = \langle p'_1, \ldots, p'_N \rangle$ iff $\forall i \, . \, p_i \subseteq p'_i$

Proof Object

State

A tuple of state components $s = \langle p_1, p_2, \ldots, p_N \rangle \in |P_1| \times |P_2| \times \ldots \times |P_N|$

State Inclusion

$s = \langle p_1, \ldots, p_N \rangle \ \subseteq \ s' = \langle p'_1, \ldots, p'_N \rangle$ iff $\forall i \,.\, p_i \subseteq p'_i$

Trace (implicitely defined, for forward search)

For a state s, an equivalence class of all traces, ending in s:

$$s_1, t_1, \ldots, s_k, t_k, \boldsymbol{s} \ \equiv \ s'_1, t'_1, \ldots, s'_m, t'_m, \boldsymbol{s}$$

Proof Object

Concurrent Trace

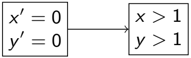
A labeled, directed, acyclic graph $A = \langle N, E, \nu, \eta \rangle$:

- $\langle N, E \rangle$ is a graph with *actions* N and edges E
- $\nu : N \to \Phi(V \cup V')$
- $\eta : E \to \Phi(V \cup V')$
  labelings of actions/edges with transition predicates

Trace Inclusion

$A = \langle N, E, \nu, \eta \rangle \ \subseteq_\lambda \ A' = \langle N', E', \nu', \eta' \rangle$ iff

- $\exists\, \lambda = \langle \lambda_N : N' \to N, \lambda_E : E' \to E \rangle$.
- for all $n' \in N'$. $\nu(\lambda_N(n')) \implies \nu'(n')$.
- for all $e' \in E'$. $\eta(\lambda_E(e')) \implies \eta'(e')$.



$x' = 0$
$y' = 0$

$x > 1$
$y > 1$

Proof Object

Concurrent Trace

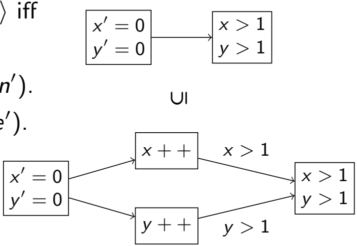
A labeled, directed, acyclic graph $A = \langle N, E, \nu, \eta \rangle$:

- $\langle N, E \rangle$ is a graph with *actions* N and edges E
- $\nu : N \to \Phi(V \cup V')$
- $\eta : E \to \Phi(V \cup V')$
  labelings of actions/edges with transition predicates

Trace Inclusion

$A = \langle N, E, \nu, \eta \rangle \ \subseteq_\lambda \ A' = \langle N', E', \nu', \eta' \rangle$ iff

- $\exists\, \lambda = \langle \lambda_N : N' \to N, \lambda_E : E' \to E \rangle$.
- for all $n' \in N'$. $\nu(\lambda_N(n')) \implies \nu'(n')$.
- for all $e' \in E'$. $\eta(\lambda_E(e')) \implies \eta'(e')$.



Proof Object

Concurrent Trace

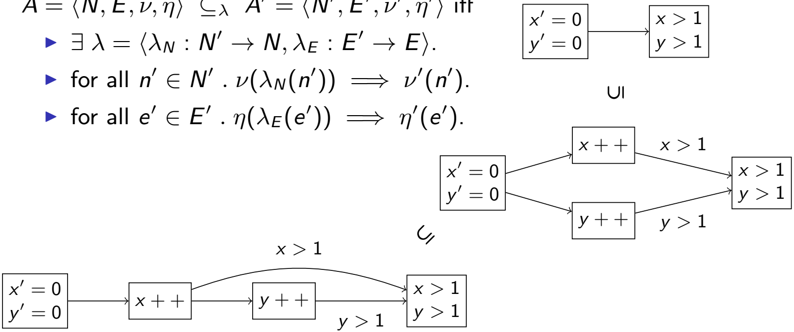
A labeled, directed, acyclic graph $A = \langle N, E, \nu, \eta \rangle$:

- $\langle N, E \rangle$ is a graph with *actions* N and edges E
- $\nu : N \rightarrow \Phi(V \cup V')$
- $\eta : E \rightarrow \Phi(V \cup V')$
  labelings of actions/edges with transition predicates

Trace Inclusion

$A = \langle N, E, \nu, \eta \rangle \ \subseteq_\lambda \ A' = \langle N', E', \nu', \eta' \rangle$ iff

- $\exists \, \lambda = \langle \lambda_N : N' \rightarrow N, \lambda_E : E' \rightarrow E \rangle$.
- for all $n' \in N'$. $\nu(\lambda_N(n')) \implies \nu'(n')$.
- for all $e' \in E'$. $\eta(\lambda_E(e')) \implies \eta'(e')$.



Proof Rules

State Transition

For a state s: $\{t^1, \ldots, t^n\}$, where $s \xrightarrow{t^i} s'_i$ are transitions, enabled in s.

Proof Rules

State Transition

For a state s: $\{t^1, \dots, t^n\}$, where $s \xrightarrow{t^i} s'_i$ are transitions, enabled in s.

Seen as Trace Transformations

$$\begin{tikzcd}
& s_1, t_1, \dots, \boldsymbol{s} \arrow[dl, Rightarrow] \arrow[d, Rightarrow] \arrow[dr, Rightarrow] & \\
s_1, t_1, \dots, s, t^1, \boldsymbol{s'_1} & \dots & s_1, t_1, \dots, s, t^n, \boldsymbol{s'_n}
\end{tikzcd}$$

Proof Rules

Causal Transition

$\tau : \{\tau_1, \ldots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big(\tau_i^m(A)\big)$$

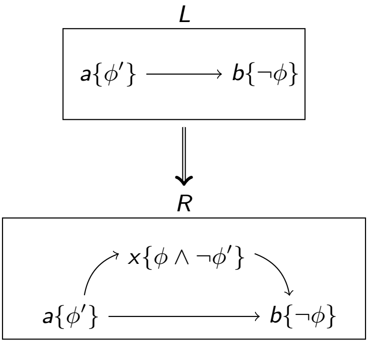
Proof Rules

Causal Transition

$\tau : \{\tau_1, \dots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big(\tau_i^m(A)\big)$$

Necessary Action



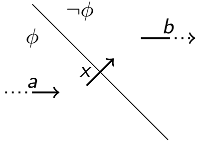
L

$$\begin{tikzcd} a\{\phi'\} \arrow[r] & b\{\neg\phi\} \end{tikzcd}$$

$\Downarrow$

R

$$\begin{tikzcd} & x\{\phi \wedge \neg\phi'\} \arrow[dr, bend left] & \\ a\{\phi'\} \arrow[ur, bend left] \arrow[rr] & & b\{\neg\phi\} \end{tikzcd}$$



Proof Rules

Causal Transition

$\tau : \{\tau_1, \ldots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big(\tau_i^m(A)\big)$$

Necessary Action

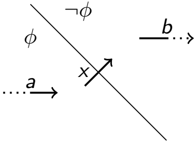
L

$$\begin{tikzcd} a\{\phi'\} \arrow[rr] & & b\{\neg\phi\} \end{tikzcd}$$

$\Downarrow$

R_{first}

$$\begin{tikzcd} & x\{\phi \wedge \neg\phi'\} \arrow[dr, bend left] & \\ a\{\phi'\} \arrow[ur, bend left, "\{\phi \wedge \phi'\}"] \arrow[rr] & & b\{\neg\phi\} \end{tikzcd}$$



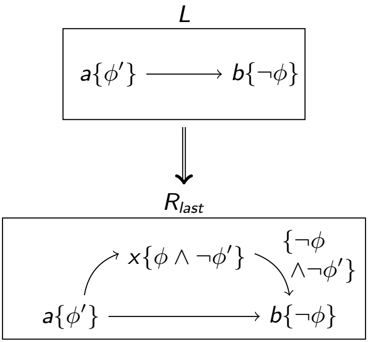
Proof Rules

Causal Transition

$\tau : \{\tau_1, \ldots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big(\tau_i^{\,m}(A)\big)$$

Necessary Action



L

$a\{\phi'\} \longrightarrow b\{\neg\phi\}$

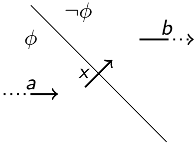
$\Downarrow$

R_{last}

$a\{\phi'\} \longrightarrow x\{\phi \wedge \neg\phi'\}$

$x\{\phi \wedge \neg\phi'\} \xrightarrow{\{\neg\phi \wedge \neg\phi'\}} b\{\neg\phi\}$

$a\{\phi'\} \longrightarrow b\{\neg\phi\}$



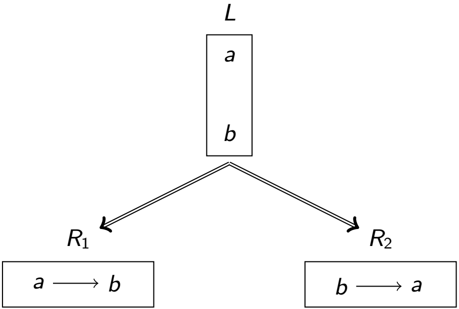
Proof Rules

Causal Transition

$\tau : \{\tau_1, \ldots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big(\tau_i^m(A)\big)$$

Order Split



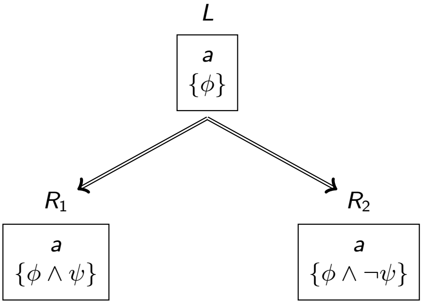
Proof Rules

Causal Transition

$\tau : \{\tau_1, \ldots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big(\tau_i^m(A)\big)$$

Action Split



L

a
$\{\phi\}$

R_1

a
$\{\phi \wedge \psi\}$

R_2

a
$\{\phi \wedge \neg\psi\}$

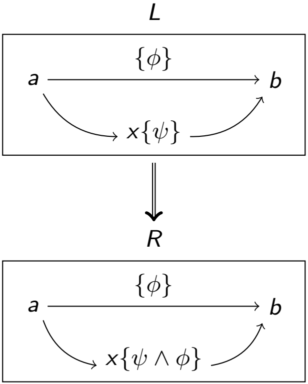
Proof Rules

Causal Transition

$\tau : \{\tau_1, \ldots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big({\tau_i}^m(A)\big)$$

Action Restriction



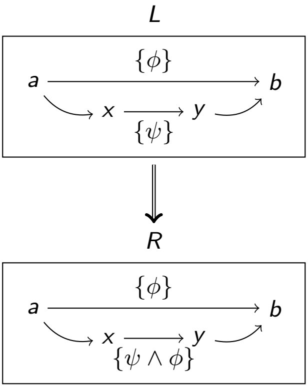
Proof Rules

Causal Transition

$\tau : \{\tau_1, \ldots, \tau_n\}$ where $\tau_i : (L \xrightarrow{r_i} R_i)$, are trace productions sharing the same left-hand side L. τ is *sound* if the following holds:

$$\forall A \,.\, A \subseteq_m L \implies \mathcal{L}(A) \subseteq \bigcup_{\tau_i \in \tau} \mathcal{L}\big({\tau_i}^m(A)\big)$$

Edge Restriction



Verification as a Search Problem: States vs Traces

```
begin
    set $Q \longleftarrow InitialAbstraction(S)$
    while not $FixedPoint(Q)$ do
        take some $q$ from $Q$
        if $IsError(q)$ then
            return unsafe
        else
            $Q := Q \cup Successors(q)$
    return safe
```

Verification as a Search Problem: States vs Traces

```
begin
    set $Q \longleftarrow InitialAbstraction(S)$
    while not $FixedPoint(Q)$ do
        take some $q$ from $Q$
        if $IsError(q)$ then
            return unsafe
        else
            $Q := Q \cup Successors(q)$
    return safe
```

Search Object:	State	Concurrent Trace

Verification as a Search Problem: States vs Traces

```
begin
    set $Q \longleftarrow \mathit{InitialAbstraction}(S)$
    while not $\mathit{FixedPoint}(Q)$ do
        take some $q$ from $Q$
        if $\mathit{IsError}(q)$ then
            return unsafe
        else
            $Q := Q \cup \mathit{Successors}(q)$
    return safe
```

Search Object:	State	Concurrent Trace
$\mathit{InitialAbstraction}(S)$:	I	$I \longrightarrow E$

Verification as a Search Problem: States vs Traces

```
begin
    set $Q \longleftarrow InitialAbstraction(S)$
    while not $FixedPoint(Q)$ do
        take some $q$ from $Q$
        if $IsError(q)$ then
            return unsafe
        else
            $Q := Q \cup Successors(q)$
    return safe
```

Search Object:	State	Concurrent Trace
$InitialAbstraction(S)$:	I	$I \longrightarrow E$
$IsError(q)$:	$q \cap E \neq \emptyset$	$Linearizable(q)$

Verification as a Search Problem: States vs Traces

```
begin
    set $Q \longleftarrow InitialAbstraction(S)$
    while not $FixedPoint(Q)$ do
        take some $q$ from $Q$
        if $IsError(q)$ then
            return unsafe
        else
            $Q := Q \cup Successors(q)$
    return safe
```

Search Object:	State	Concurrent Trace
$InitialAbstraction(S)$:	I	$I \longrightarrow E$
$IsError(q)$:	$q \cap E \neq \emptyset$	$Linearizable(q)$
$Successors(q)$:	$StateTransition(q)$	$CausalTransition(q)$

Verification as a Search Problem: States vs Traces

```
begin
    set $Q \longleftarrow InitialAbstraction(S)$
    while not $FixedPoint(Q)$ do
        take some $q$ from $Q$
        if $IsError(q)$ then
            return unsafe
        else
            $Q := Q \cup Successors(q)$
    return safe
```

Search Object:	State	Concurrent Trace
$InitialAbstraction(S)$:	I	$I \longrightarrow E$
$IsError(q)$:	$q \cap E \neq \emptyset$	$Linearizable(q)$
$Successors(q)$:	$StateTransition(q)$	$CausalTransition(q)$
$FixedPoint(Q)$:	$\forall q \in Q \,.$ $Succesors(q) \subseteq Q$	?

Looking Closer into State Fixed-point...

State Fixed-point

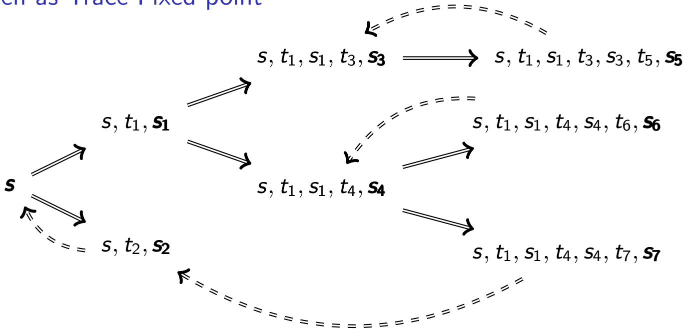
$\forall q \in Q \,.\, Succesors(q) \subseteq Q$

Looking Closer into State Fixed-point...

State Fixed-point

$\forall q \in Q \,.\, Succesors(q) \subseteq Q$

Seen as Trace Fixed-point



$\boldsymbol{s}$

$s, t_1, \boldsymbol{s_1}$

$s, t_2, \boldsymbol{s_2}$

$s, t_1, s_1, t_3, \boldsymbol{s_3}$

$s, t_1, s_1, t_4, \boldsymbol{s_4}$

$s, t_1, s_1, t_3, s_3, t_5, \boldsymbol{s_5}$

$s, t_1, s_1, t_4, s_4, t_6, \boldsymbol{s_6}$

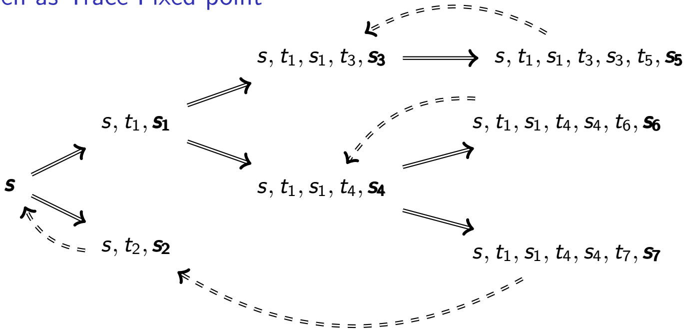
$s, t_1, s_1, t_4, s_4, t_7, \boldsymbol{s_7}$

Looking Closer into State Fixed-point...

State Fixed-point

$\forall q \in Q \, . \, \mathit{Succesors}(q) \subseteq Q$

Seen as Trace Fixed-point



$\boldsymbol{s}$

$s, t_1, \boldsymbol{s_1}$

$s, t_2, \boldsymbol{s_2}$

$s, t_1, s_1, t_3, \boldsymbol{s_3}$

$s, t_1, s_1, t_4, \boldsymbol{s_4}$

$s, t_1, s_1, t_3, s_3, t_5, \boldsymbol{s_5}$

$s, t_1, s_1, t_4, s_4, t_6, \boldsymbol{s_6}$

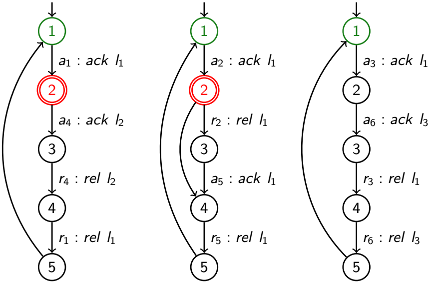
$s, t_1, s_1, t_4, s_4, t_7, \boldsymbol{s_7}$

Trace Fixed-point

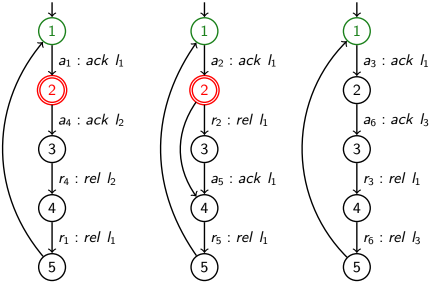
There is no **finite** trace between I and E.

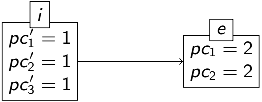
Alternatively: any trace between I and E should have **infinite** length!

Causal Trace Unwindings



Causal Trace Unwindings





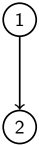
i

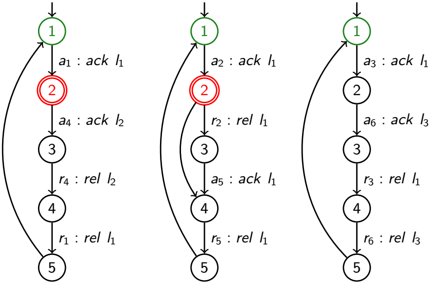
$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

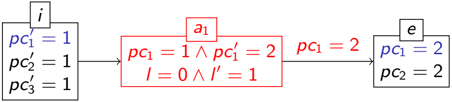
e

$pc_1 = 2$
$pc_2 = 2$

Causal Trace Unwindings







i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

a_1

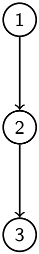
$pc_1 = 1 \wedge pc_1' = 2$
$l = 0 \wedge l' = 1$

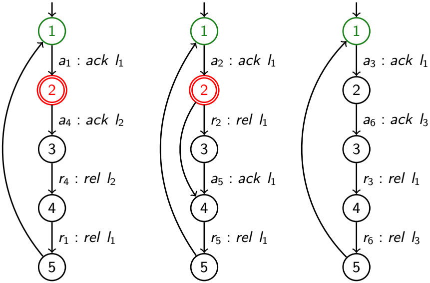
$pc_1 = 2$

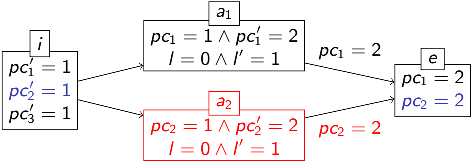
e

$pc_1 = 2$
$pc_2 = 2$

Causal Trace Unwindings







i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

a_1

$pc_1 = 1 \land pc_1' = 2$
$l = 0 \land l' = 1$

a_2

$pc_2 = 1 \land pc_2' = 2$
$l = 0 \land l' = 1$

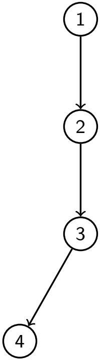
$pc_1 = 2$

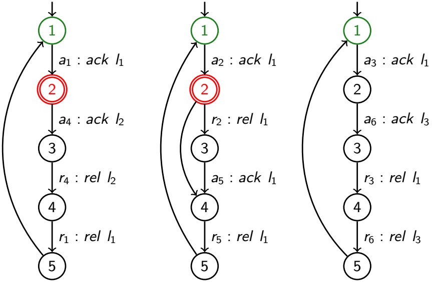
$pc_2 = 2$

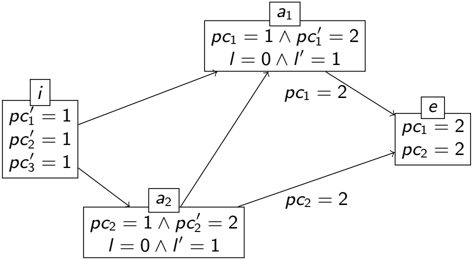
e

$pc_1 = 2$
$pc_2 = 2$

Causal Trace Unwindings







a_1

$pc_1 = 1 \wedge pc_1' = 2$
$l = 0 \wedge l' = 1$

i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

$pc_1 = 2$

e

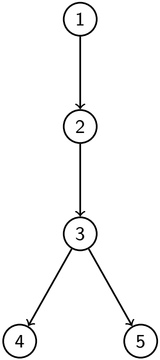
$pc_1 = 2$
$pc_2 = 2$

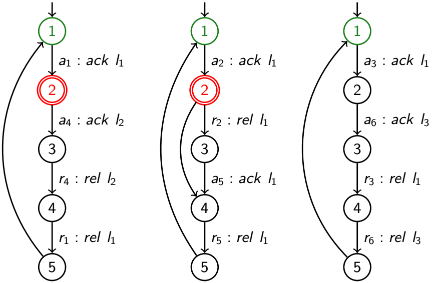
a_2

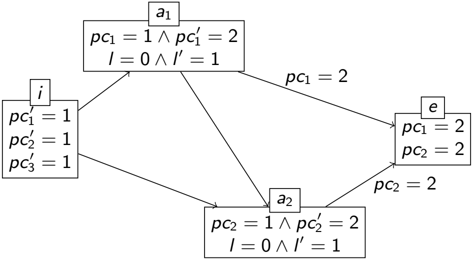
$pc_2 = 1 \wedge pc_2' = 2$
$l = 0 \wedge l' = 1$

$pc_2 = 2$

Causal Trace Unwindings







a_1

$pc_1 = 1 \wedge pc_1' = 2$
$l = 0 \wedge l' = 1$

i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

$pc_1 = 2$

e

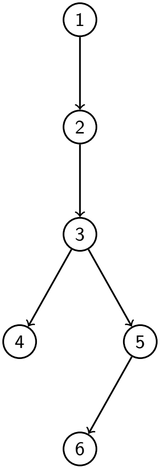
$pc_1 = 2$
$pc_2 = 2$

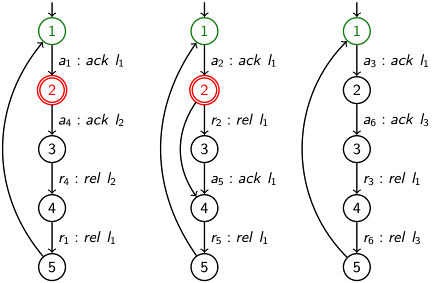
$pc_2 = 2$

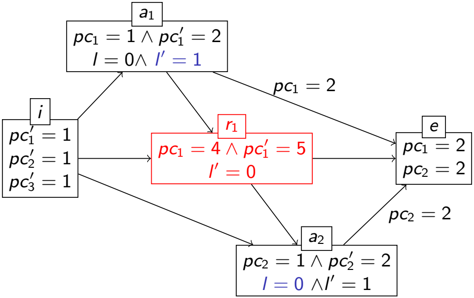
a_2

$pc_2 = 1 \wedge pc_2' = 2$
$l = 0 \wedge l' = 1$

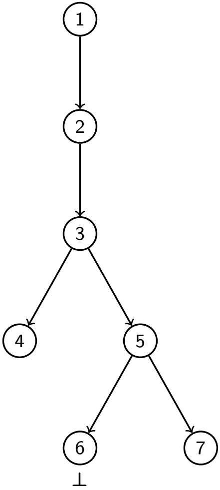
Causal Trace Unwindings

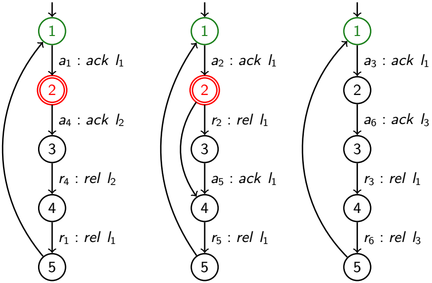


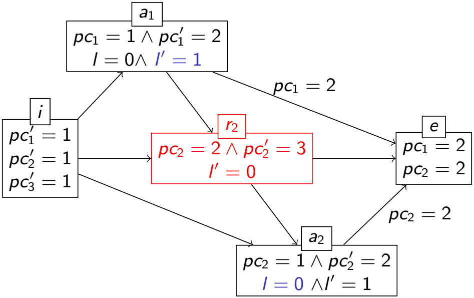




Causal Trace Unwindings







a_1

$pc_1 = 1 \wedge pc_1' = 2$
$l = 0 \wedge\ l' = 1$

i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

r_2

$pc_2 = 2 \wedge pc_2' = 3$
$l' = 0$

e

$pc_1 = 2$
$pc_2 = 2$

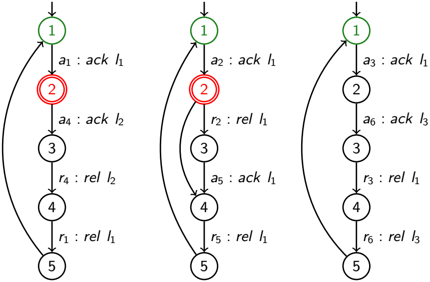
a_2

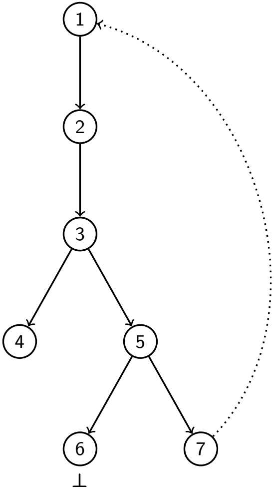
$pc_2 = 1 \wedge pc_2' = 2$
$l = 0\ \wedge l' = 1$

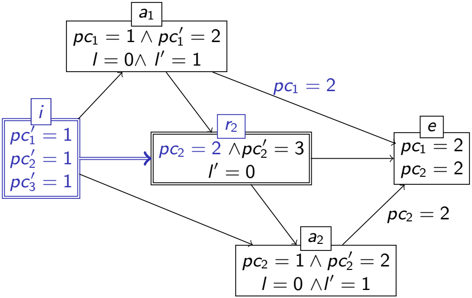
$pc_1 = 2$

$pc_2 = 2$

Causal Trace Unwindings







a_1

$pc_1 = 1 \wedge pc_1' = 2$
$l = 0 \wedge l' = 1$

i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

r_2

$pc_2 = 2 \wedge pc_2' = 3$
$l' = 0$

$pc_1 = 2$

e

$pc_1 = 2$
$pc_2 = 2$

$pc_2 = 2$

a_2

$pc_2 = 1 \wedge pc_2' = 2$
$l = 0 \wedge l' = 1$

Causal Trace Unwindings

Theorem (Soundness of Trace Unwinding)

If there exists a correct causal trace unwinding for $\mathcal{P}$, where every causal path is either contradictory or unbounded, then $\mathcal{P}$ is safe.

Causal Trace Unwindings

Theorem (Soundness of Trace Unwinding)

If there exists a correct causal trace unwinding for $\mathcal{P}$, where every causal path is either contradictory or unbounded, then $\mathcal{P}$ is safe.

Causal Trace Unwindings

Theorem (Soundness of Trace Unwinding)

If there exists a correct causal trace unwinding for $\mathcal{P}$, where every causal path is either contradictory or unbounded, then $\mathcal{P}$ is safe.

$$\Downarrow$$

Trace Tableau = **Trace Unwinding**
+ abstract labels + covering relation

Causal Trace Tableaux

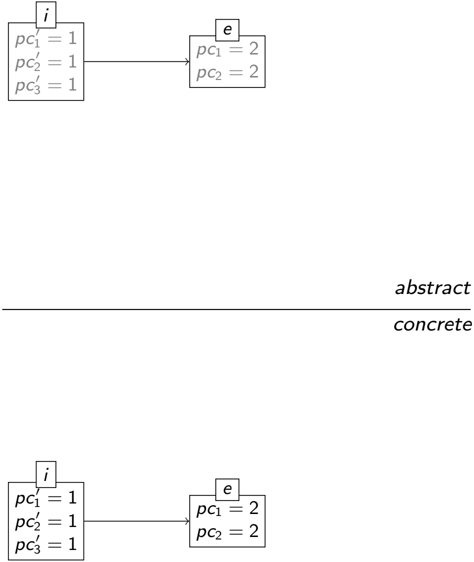
abstract

---

concrete

Causal Trace Tableaux

①



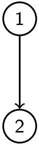
i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ $\rightarrow$ e: $pc_1 = 2$, $pc_2 = 2$

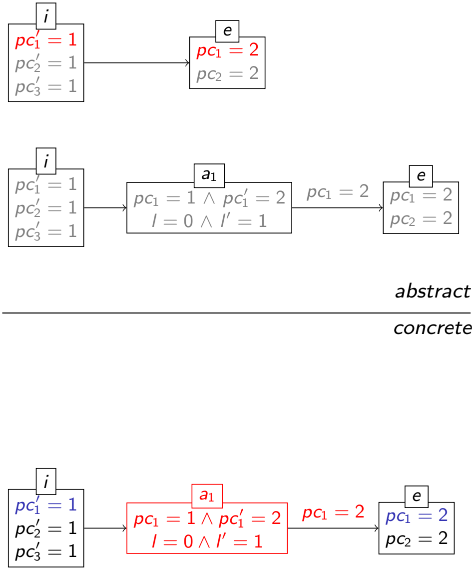
abstract

concrete

i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ $\rightarrow$ e: $pc_1 = 2$, $pc_2 = 2$

Causal Trace Tableaux





i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ → e: $pc_1 = 2$, $pc_2 = 2$

i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ → a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$ $\xrightarrow{pc_1 = 2}$ e: $pc_1 = 2$, $pc_2 = 2$

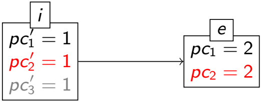
abstract

---

concrete

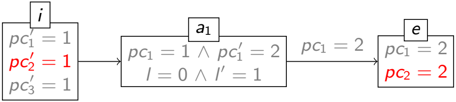
i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ → a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$ $\xrightarrow{pc_1 = 2}$ e: $pc_1 = 2$, $pc_2 = 2$

Causal Trace Tableaux



i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$

e: $pc_1 = 2$, $pc_2 = 2$



i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$

a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$

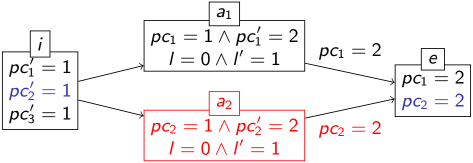
$pc_1 = 2$

e: $pc_1 = 2$, $pc_2 = 2$

abstract

---

concrete



i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$

a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$

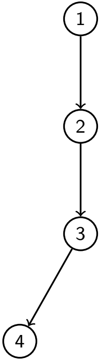
a_2: $pc_2 = 1 \wedge pc_2' = 2$, $l = 0 \wedge l' = 1$

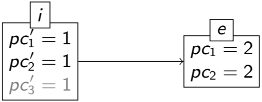
$pc_1 = 2$

$pc_2 = 2$

e: $pc_1 = 2$, $pc_2 = 2$

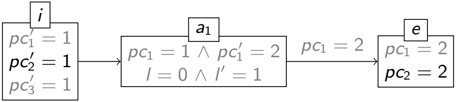
Causal Trace Tableaux





i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$

e: $pc_1 = 2$, $pc_2 = 2$



i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$

a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$

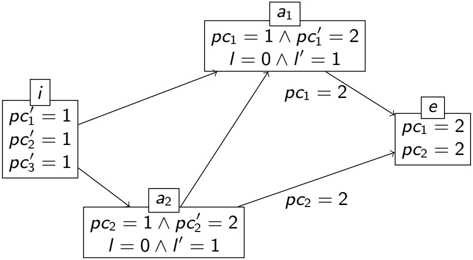
$pc_1 = 2$

e: $pc_1 = 2$, $pc_2 = 2$

abstract

---

concrete



a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$

i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$

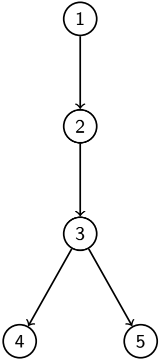
$pc_1 = 2$

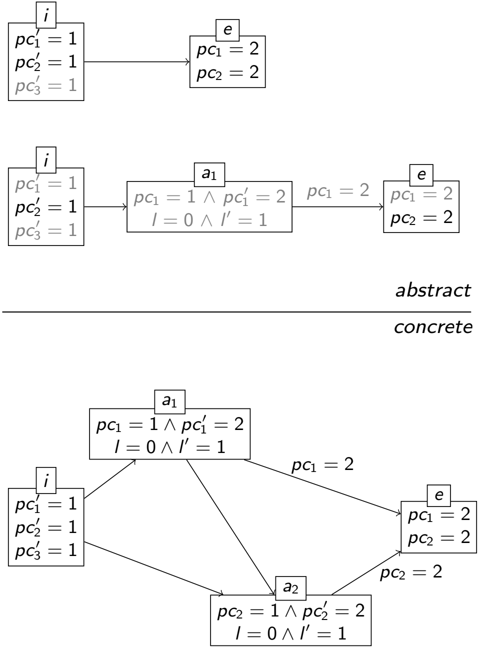
e: $pc_1 = 2$, $pc_2 = 2$

$pc_2 = 2$

a_2: $pc_2 = 1 \wedge pc_2' = 2$, $l = 0 \wedge l' = 1$

Causal Trace Tableaux





i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ → e: $pc_1 = 2$, $pc_2 = 2$

i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ → a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$ $\xrightarrow{pc_1 = 2}$ e: $pc_1 = 2$, $pc_2 = 2$

abstract

---

concrete

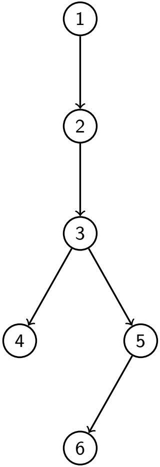
i: $pc_1' = 1$, $pc_2' = 1$, $pc_3' = 1$ → a_1, a_2

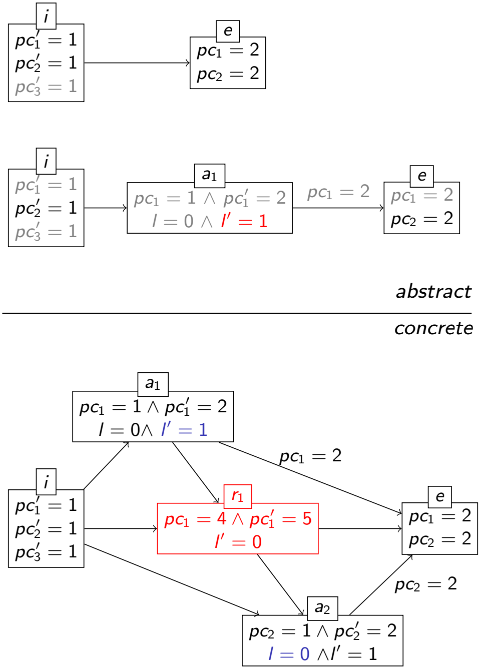
a_1: $pc_1 = 1 \wedge pc_1' = 2$, $l = 0 \wedge l' = 1$ → a_2; $a_1 \xrightarrow{pc_1 = 2} e$

a_2: $pc_2 = 1 \wedge pc_2' = 2$, $l = 0 \wedge l' = 1$ $\xrightarrow{pc_2 = 2}$ e

e: $pc_1 = 2$, $pc_2 = 2$

Causal Trace Tableaux





i

$pc_1' = 1$

$pc_2' = 1$

$pc_3' = 1$

e

$pc_1 = 2$

$pc_2 = 2$

i

$pc_1' = 1$

$pc_2' = 1$

$pc_3' = 1$

a_1

$pc_1 = 1 \wedge pc_1' = 2$

$l = 0 \wedge l' = 1$

$pc_1 = 2$

e

$pc_1 = 2$

$pc_2 = 2$

abstract

concrete

a_1

$pc_1 = 1 \wedge pc_1' = 2$

$l = 0 \wedge l' = 1$

$pc_1 = 2$

i

$pc_1' = 1$

$pc_2' = 1$

$pc_3' = 1$

r_1

$pc_1 = 4 \wedge pc_1' = 5$

$l' = 0$

e

$pc_1 = 2$

$pc_2 = 2$

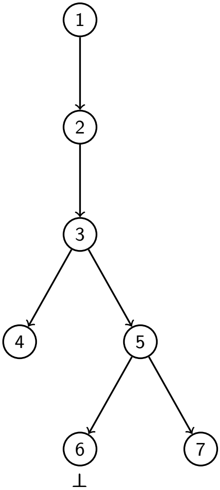
$pc_2 = 2$

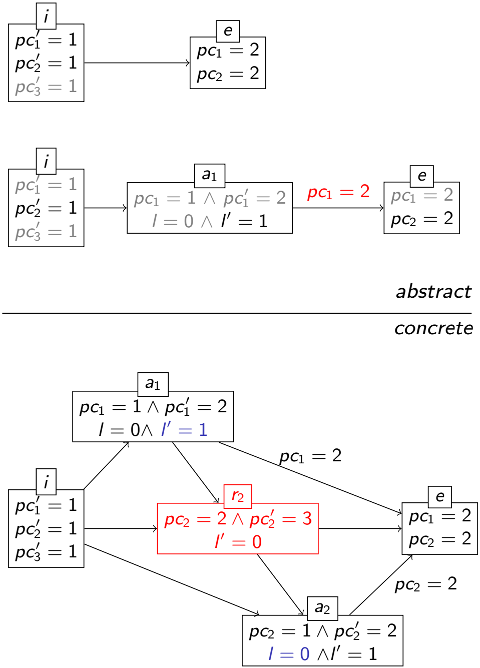
a_2

$pc_2 = 1 \wedge pc_2' = 2$

$l = 0 \wedge l' = 1$

Causal Trace Tableaux





i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

e

$pc_1 = 2$
$pc_2 = 2$

i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

a_1

$pc_1 = 1 \wedge pc_1' = 2$
$l = 0 \wedge l' = 1$

$pc_1 = 2$

e

$pc_1 = 2$
$pc_2 = 2$

abstract

concrete

a_1

$pc_1 = 1 \wedge pc_1' = 2$
$l = 0 \wedge l' = 1$

$pc_1 = 2$

i

$pc_1' = 1$
$pc_2' = 1$
$pc_3' = 1$

r_2

$pc_2 = 2 \wedge pc_2' = 3$
$l' = 0$

e

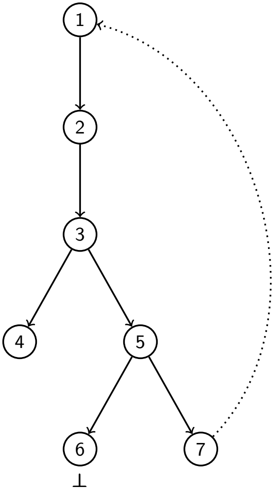
$pc_1 = 2$
$pc_2 = 2$

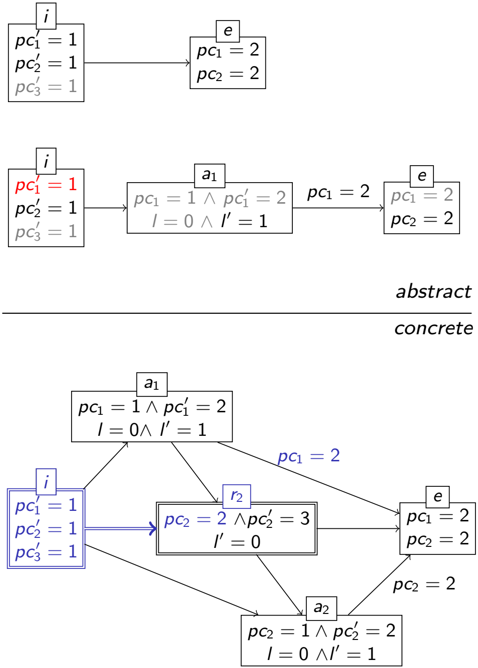
$pc_2 = 2$

a_2

$pc_2 = 1 \wedge pc_2' = 2$
$l = 0 \wedge l' = 1$

Causal Trace Tableaux





Causal Trace Tableaux

Theorem (Soundness)

If there exists a correct and complete causal trace tableau for a parallel program $\mathcal{P}$, then $\mathcal{P}$ is safe.

Theorem (Completeness)

If a parallel program $\mathcal{P}$ with finite-state quotient is safe, then there exists a correct and complete causal trace tableau for $\mathcal{P}$.

Causality-based Verification: Conclusion

We propose to shift emphasis from state space exploration to causality-based proof search:

- We capture causality by concurrent traces and their transformations
- More powerful proof object allows to better exhibit causal relationships
- More powerful proof rules lead to substantially shorter proofs

Causality-based Verification: Conclusion

We propose to shift emphasis from state space exploration to causality-based proof search:

- We capture causality by concurrent traces and their transformations
- More powerful proof object allows to better exhibit causal relationships
- More powerful proof rules lead to substantially shorter proofs

Reduces the complexity from exponential to polynomial for the important class of multi-threaded programs.