Partial Order Reduction for Trace Abstraction Refinement

Dominik Klumpp
University of Freiburg

Andreas Podelski
University of Freiburg

Azadeh Farzan
University of Toronto

MOVEP 2020

Motivation

```
i := i-1
a[i] := 21
assert a[i] != 0
```

||

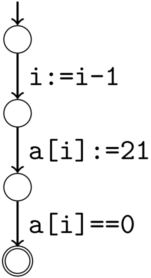
```
a[j] := 7
j := i+1
a[j] := 0
```

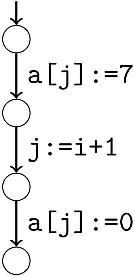
Motivation

```
i := i-1
a[i] := 21
assert a[i] != 0
```

||

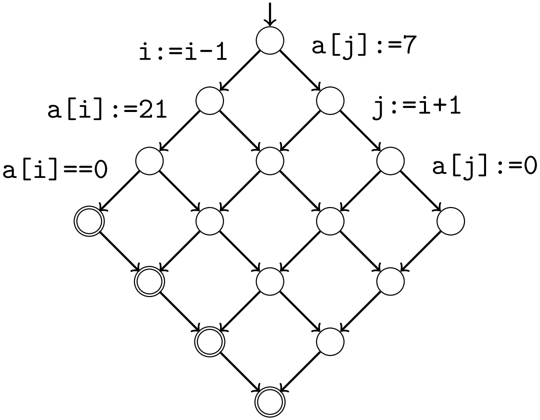
```
a[j] := 7
j := i+1
a[j] := 0
```





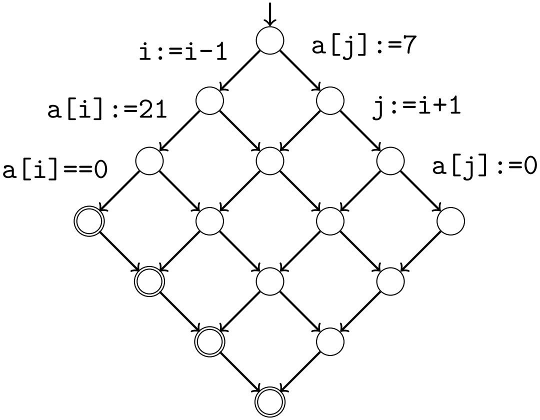
Motivation

Control Flow Automaton P:



Motivation

Control Flow Automaton P:



Goal: Prove P is *correct*

i.e. show all accepted *error traces* τ are *infeasible*:

$\{\textbf{true}\}\ \tau\ \{\textbf{false}\}$
valid Hoare triple

Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

$$A = (Q, \Sigma, \textbf{true}, \Delta, \{\textbf{false}\})$$

Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

$$A = (Q, \Sigma, \textbf{true}, \Delta, \{\textbf{false}\})$$

logical formulae
over program variables

Trace Abstraction Refinement

iterativelyy build Floyd-Hoare automaton

$$A = (\underbrace{Q}_{\text{logical formulae over program variables}}, \underbrace{\Sigma}_{\text{program statements taken from } P}, \textbf{true}, \Delta, \{\textbf{false}\})$$

Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

$$A = (Q, \Sigma, \mathbf{true}, \Delta, \{\mathbf{false}\})$$

logical formulae
over program variables

program statements
taken from P

such that for all $(\varphi, st, \psi) \in \Delta$,
Hoare triple $\{\varphi\}\, st\, \{\psi\}$ is valid.

Trace Abstraction Refinement

iterativelyly build Floyd-Hoare automaton

$$A = (Q, \Sigma, \textbf{true}, \Delta, \{\textbf{false}\})$$

logical formulae
over program variables

program statements
taken from P

such that for all $(\varphi, st, \psi) \in \Delta$,
Hoare triple $\{\varphi\}\, st\, \{\psi\}$ is valid.

$\Rightarrow$ for all $\tau \in A$, Hoare triple
$\{\textbf{true}\}\, \tau\, \{\textbf{false}\}$ is valid

Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

$$A = (Q, \Sigma, \mathbf{true}, \Delta, \{\mathbf{false}\})$$

logical formulae
over program variables

program statements
taken from P

such that for all $(\varphi, st, \psi) \in \Delta$,
Hoare triple $\{\varphi\}\, st\, \{\psi\}$ is valid.

$\Rightarrow$ for all $\tau \in A$, Hoare triple
$\{\mathbf{true}\}\, \tau\, \{\mathbf{false}\}$ is valid

If $P \subseteq A$, then P is correct.

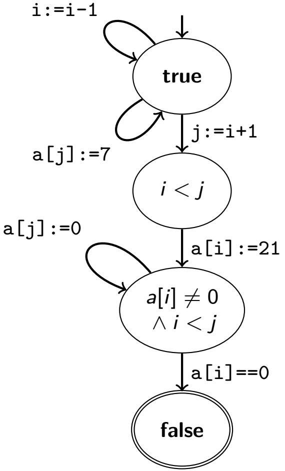
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

$$A = (Q, \Sigma, \textbf{true}, \Delta, \{\textbf{false}\})$$

logical formulae over program variables

program statements taken from P

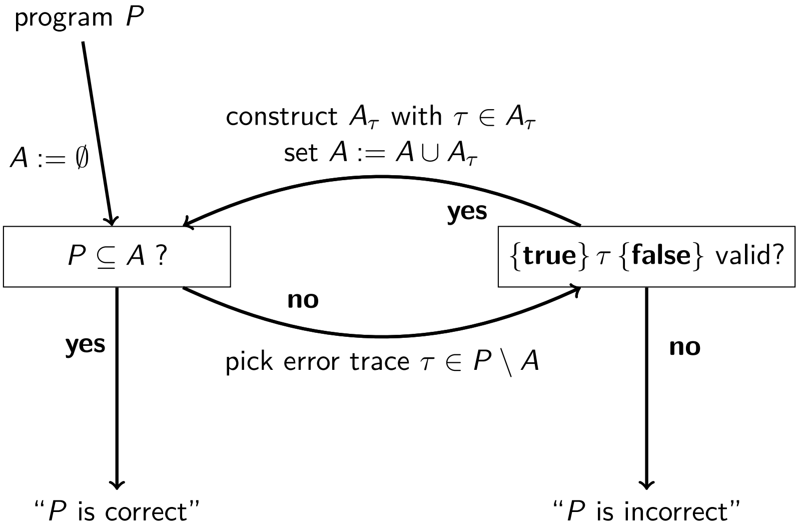


such that for all $(\varphi, st, \psi) \in \Delta$,
Hoare triple $\{\varphi\}\, st\, \{\psi\}$ is valid.

$\Rightarrow$ for all $\tau \in A$, Hoare triple
$\{\textbf{true}\}\, \tau\, \{\textbf{false}\}$ is valid

If $P \subseteq A$, then P is correct.

Trace Abstraction Refinement



Partial Order Reduction

error traces:

τ_1 :	`i:=i-1`	`a[j]:=7`	`j:=i+1`	`a[i]:=21`	`a[j]:=0`	`a[i]==0`
τ_2 :	`i:=i-1`	`a[j]:=7`	`a[i]:=21`	`j:=i+1`	`a[j]:=0`	`a[i]==0`

Partial Order Reduction

error traces:

τ_1 : `i:=i-1` `a[j]:=7` `j:=i+1` `a[i]:=21` `a[j]:=0` `a[i]==0`

τ_2 : `i:=i-1` `a[j]:=7` `a[i]:=21` `j:=i+1` `a[j]:=0` `a[i]==0`

- Idea: order of `a[i]:=21` and `j:=i+1` irrelevant!

Partial Order Reduction

error traces:

τ_1 :	`i:=i-1`	`a[j]:=7`		`j:=i+1`	`a[i]:=21`	`a[j]:=0`	`a[i]==0`
τ_2 :	`i:=i-1`	`a[j]:=7`	`a[i]:=21`		`j:=i+1`	`a[j]:=0`	`a[i]==0`

- Idea: order of `a[i]:=21` and `j:=i+1` irrelevant!
- Define *(partial) commutativity relation I* over program statements
  here: `a[i]:=21` and `j:=i+1` *commute*

Partial Order Reduction

error traces:

τ_1 :	`i:=i-1`	`a[j]:=7`	`j:=i+1`	`a[i]:=21`	`a[j]:=0`	`a[i]==0`
τ_2 :	`i:=i-1`	`a[j]:=7`	`a[i]:=21`	`j:=i+1`	`a[j]:=0`	`a[i]==0`

- Idea: order of `a[i]:=21` and `j:=i+1` irrelevant!
- Define *(partial) commutativity relation I* over program statements
  here: `a[i]:=21` and `j:=i+1` *commute*
- Traces τ_1, τ_2 are *equivalent* ($\tau_1 \sim_I \tau_2$) iff

$$\tau_1 = \tau_2 \quad \textbf{or} \quad \begin{array}{c} \tau_1 = \rho a b \sigma, \tau_2 = \rho b a \sigma \\ \text{where } (a, b) \in I \end{array} \quad \textbf{or} \quad \exists \tau' \, . \, \tau_1 \sim \tau' \sim \tau_2$$

Partial Order Reduction

error traces:

τ_1 : `i:=i-1` `a[j]:=7` `j:=i+1` `a[i]:=21` `a[j]:=0` `a[i]==0`

τ_2 : `i:=i-1` `a[j]:=7` `a[i]:=21` `j:=i+1` `a[j]:=0` `a[i]==0`

- Idea: order of `a[i]:=21` and `j:=i+1` irrelevant!
- Define *(partial) commutativity relation I* over program statements here: `a[i]:=21` and `j:=i+1` *commute*
- Traces τ_1, τ_2 are *equivalent* ($\tau_1 \sim_I \tau_2$) iff

$$\tau_1 = \tau_2 \quad \textbf{or} \quad \begin{array}{c} \tau_1 = \rho a b \sigma, \tau_2 = \rho b a \sigma \\ \text{where } (a, b) \in I \end{array} \quad \textbf{or} \quad \exists \tau' \,.\, \tau_1 \sim \tau' \sim \tau_2$$

- Goal: Only analyse one representative of each equivalence class!

Partial Order Reduction

New proof criterion:

If $P \subseteq cl_I(A)$, then P is correct.

(for suitable commutativity relation I)

Partial Order Reduction

closure of A
all traces equivalent
to some $\tau \in A$

New proof criterion:

If $P \subseteq cl_I(A)$, then P is correct.

(for suitable commutativity relation I)

Partial Order Reduction

closure of A

all traces equivalent
to some $\tau \in A$

New proof criterion:

If $P \subseteq cl_I(A)$, then P is correct.

(for suitable commutativity relation I)

Algorithmic Check:

$$P \subseteq cl_I(A) \iff \exists \text{ reduction } P' \text{ of } P \text{ s.t. } P' \subseteq A$$

Partial Order Reduction

closure of A
all traces equivalent
to some $\tau \in A$

New proof criterion:

If $P \subseteq cl_I(A)$, then P is correct.

(for suitable commutativity relation I)

Algorithmic Check:

i.e. $cl_I(P') = P$

$$P \subseteq cl_I(A) \iff \exists \text{ reduction } P' \text{ of } P \text{ s.t. } P' \subseteq A$$

Partial Order Reduction

closure of A
all traces equivalent
to some $\tau \in A$

New proof criterion:

If $P \subseteq cl_I(A)$, then P is correct.

(for suitable commutativity relation I)

---

i.e. $cl_I(P') = P$

Algorithmic Check:

$$P \subseteq cl_I(A) \iff \exists \text{ reduction } P' \text{ of } P \text{ s.t. } P' \subseteq A$$

Hence: Compute a (regular) reduction P' and check $P' \subseteq A$

- sufficient but (necessarily) incomplete
- more general than checking $P \subseteq A$

Concrete Commutativity

Concrete Commutativity

Definition:

$$a \text{ and } b \text{ commute} \qquad \text{iff} \qquad [\![a]\!] \circ [\![b]\!] = [\![b]\!] \circ [\![a]\!]$$

Concrete Commutativity

semantics of statement *a*
relation over program states

Definition:

$$a \text{ and } b \text{ commute} \qquad \text{iff} \qquad [\![a]\!] \circ [\![b]\!] = [\![b]\!] \circ [\![a]\!]$$

Concrete Commutativity

Definition:

semantics of statement a

relation over program states

$$a \text{ and } b \text{ commute} \qquad \text{iff} \qquad \boxed{[\![a]\!]} \circ [\![b]\!] = [\![b]\!] \circ [\![a]\!]$$

For example:

$$[\![\mathtt{a[i]:=21}]\!] = \{\, (s, s') \mid s' = s\{\mathtt{a} \mapsto \mathit{store}(s(\mathtt{a}), s(\mathtt{i}), 21)\} \,\}$$
$$[\![\mathtt{j:=i+1}]\!] = \{\, (s, s') \mid s' = s\{\mathtt{j} \mapsto s(\mathtt{i}) + 1\} \,\}$$

Therefore

$$[\![\mathtt{a[i]:=21}]\!] \circ [\![\mathtt{j:=i+1}]\!] = [\![\mathtt{j:=i+1}]\!] \circ [\![\mathtt{a[i]:=21}]\!]$$

Concrete Commutativity

semantics of statement a
relation over program states

Definition:

$$a \text{ and } b \text{ commute} \qquad \text{iff} \qquad \boxed{[\![a]\!]} \circ [\![b]\!] = [\![b]\!] \circ [\![a]\!]$$

For example:

$$[\![\texttt{a[i]:=21}]\!] = \{\, (s, s') \mid s' = s\{\texttt{a} \mapsto \mathit{store}(s(\texttt{a}), s(\texttt{i}), 21)\}\,\}$$
$$[\![\texttt{j:=i+1}]\!] = \{\, (s, s') \mid s' = s\{\texttt{j} \mapsto s(\texttt{i}) + 1\}\,\}$$

Therefore

$$[\![\texttt{a[i]:=21}]\!] \circ [\![\texttt{j:=i+1}]\!] = [\![\texttt{j:=i+1}]\!] \circ [\![\texttt{a[i]:=21}]\!]$$

Combined with Trace Abstraction Refinement by Cassez et al.

[1] Franck Cassez and Frowin Ziegler. "Verification of Concurrent Programs Using Trace Abstraction Refinement". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. LNCS. Springer, Nov. 2015, pp. 233–248.

Conditional Commutativity

Do `a[i]:=21` and `a[j]:=0` commute?

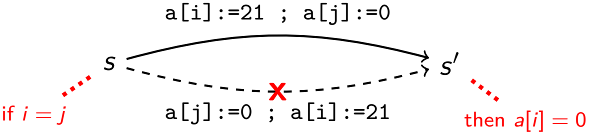
Conditional Commutativity

Do a[i]:=21 and a[j]:=0 commute?

$$\begin{tikzcd} s \arrow[r, bend left, "\texttt{a[i]:=21 ; a[j]:=0}"] \arrow[r, bend right, dashed, "\texttt{a[j]:=0 ; a[i]:=21}"'] & s' \end{tikzcd}$$

Conditional Commutativity

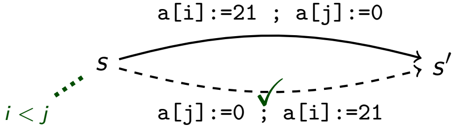
Do `a[i]:=21` and `a[j]:=0` commute?



- In general: No!

Conditional Commutativity

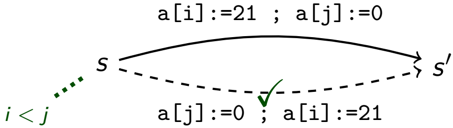
Do a[i]:=21 and a[j]:=0 commute?



- In general: No!
- In our program: $i < j$ due to assignments j:=i+1, i:=i-1

Conditional Commutativity

Do a[i]:=21 and a[j]:=0 commute?

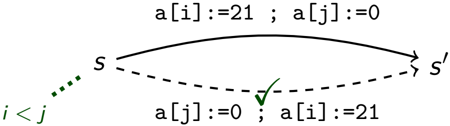


- In general: No!
- In our program: $i < j$ due to assignments j:=i+1, i:=i-1
- Godefroid 1996: *conditional* commutativity relation parametrized in state of transition system

[2] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Springer, 1996.

Conditional Commutativity

Do `a[i]:=21` and `a[j]:=0` commute?



- In general: No!
- In our program: $i < j$ due to assignments `j:=i+1`, `i:=i-1`
- Godefroid 1996: *conditional* commutativity relation parametrized in state of transition system
- Here: state of Floyd-Hoare automaton A (i.e. a formula φ):

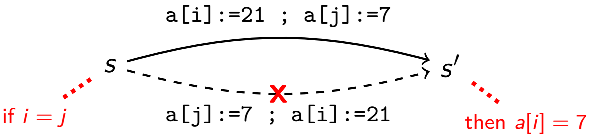
$$([\![a]\!] \circ [\![b]\!]) \cap (\varphi \times \textbf{true}) = ([\![b]\!] \circ [\![a]\!]) \cap (\varphi \times \textbf{true})$$

Abstract Commutativity

Do `a[i]:=21` and `a[j]:=7` commute?

Abstract Commutativity

Do `a[i]:=21` and `a[j]:=7` commute?



- No: order matters in case $i = j$ (which is possible)

Abstract Commutativity

Do `a[i]:=21` and `a[j]:=7` commute?

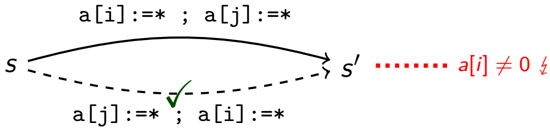
$$\begin{tikzcd} s \arrow[r, bend left, "\texttt{a[i]:=* ; a[j]:=*}"] \arrow[r, bend right, dashed, "\texttt{a[j]:=* ; a[i]:=*}"'] & s' \end{tikzcd}$$

- No: order matters in case $i = j$ (which is possible)
- In our program: We only care that $a[i] \neq 0$
  $\Rightarrow$ find abstractions

[3] Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran. "A Calculus of Atomic Actions". In: *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '09. New York, NY, USA: ACM, 2009, pp. 2–15.

Abstract Commutativity

Do `a[i]:=21` and `a[j]:=7` commute?



- No: order matters in case $i = j$ (which is possible)
- In our program: We only care that $a[i] \neq 0$
  $\Rightarrow$ find abstractions
- Abstracted program may be unsound!
  $\Rightarrow$ bound abstraction by proof candidate A

Abstract Commutativity

Do `a[i]:=21` and `a[j]:=7` commute?

`a[i]:=* with a[i]!=0 ; a[j]:=* with a[j]!=0`

$s \qquad s'$ ········ $a[i] \neq 0$ ✓

`a[j]:=* with a[j]!=0` ✓ `; a[i]:=* with a[i]!=0`

- No: order matters in case $i = j$ (which is possible)
- In our program: We only care that $a[i] \neq 0$
  $\Rightarrow$ find abstractions
- Abstracted program may be unsound!
  $\Rightarrow$ bound abstraction by proof candidate A

Abstract and Concrete Commutativity

Abstract and Concrete Commutativity

- Abstraction sometimes loses commutativity
- Combine (conditional) concrete and abstract commutativity

Abstract and Concrete Commutativity

- Abstraction sometimes loses commutativity
- Combine (conditional) concrete and abstract commutativity

New proof criterion:

> If $P \subseteq cl_{concr}(cl_{abstr}(A))$, then P is correct.

$\Rightarrow$ develop new partial order reduction algorithms for sufficient check

Abstract and Concrete Commutativity

- Abstraction sometimes loses commutativity
- Combine (conditional) concrete and abstract commutativity

New proof criterion:

$$\boxed{\text{If } P \subseteq cl_{concr}(cl_{abstr}(A)) \text{, then } P \text{ is correct.}}$$

$\Rightarrow$ develop new partial order reduction algorithms for sufficient check

Very general criterion:

$$P \subseteq cl_{concr}(A) \implies P \subseteq cl_{concr}(cl_{abstr}(A))$$
$$P \subseteq cl_{abstr}(A) \implies P \subseteq cl_{concr}(cl_{abstr}(A))$$

Future Work

Future Work

- Find suitable notion for abstract commutativity

Future Work

- Find suitable notion for abstract commutativity
  - currently: capture commutativity given by Owicki-Gries proofs

Future Work

- Find suitable notion for abstract commutativity
  - currently: capture commutativity given by Owicki-Gries proofs
  - provide theoretical guarantee for commutativity

Future Work

- Find suitable notion for abstract commutativity
  - currently: capture commutativity given by Owicki-Gries proofs
  - provide theoretical guarantee for commutativity
- Partial order reduction algorithms to check proof criterion

Future Work

- Find suitable notion for abstract commutativity
  - currently: capture commutativity given by Owicki-Gries proofs
  - provide theoretical guarantee for commutativity
- Partial order reduction algorithms to check proof criterion
  - so far: based on *sleep set* algorithm

Future Work

- Find suitable notion for abstract commutativity
  - currently: capture commutativity given by Owicki-Gries proofs
  - provide theoretical guarantee for commutativity
- Partial order reduction algorithms to check proof criterion
  - so far: based on *sleep set* algorithm
  - investigate other partial order techniques

Future Work

- Find suitable notion for abstract commutativity
  - currently: capture commutativity given by Owicki-Gries proofs
  - provide theoretical guarantee for commutativity
- Partial order reduction algorithms to check proof criterion
  - so far: based on *sleep set* algorithm
  - investigate other partial order techniques
- Empirical evaluation: effectiveness for verification

Future Work

- Find suitable notion for abstract commutativity
  - currently: capture commutativity given by Owicki-Gries proofs
  - provide theoretical guarantee for commutativity
- Partial order reduction algorithms to check proof criterion
  - so far: based on *sleep set* algorithm
  - investigate other partial order techniques
- Empirical evaluation: effectiveness for verification
- Theoretical complexity result

The End

Thank you for your attention.
Questions?