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

\[
i := i - 1 \\
a[i] := 21 \\
assert a[i] \neq 0
\]

||

\[
a[j] := 7 \\
j := i + 1 \\
a[j] := 0
\]
i := i - 1
a[i] := 21
assert a[i] != 0

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

Control Flow Automaton $P$:

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

Goal: Prove $P$ is correct, i.e. show all accepted error traces $\tau$ are infeasible:

- $\{true\}$
- $\{false\}$

valid Hoare triple
Motivation

Control Flow Automaton $P$:

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

Goal: Prove $P$ is correct

i.e. show all accepted error traces $\tau$
are infeasible:

$\{\text{true}\} \tau \{\text{false}\}$
valid Hoare triple
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

\[ A = (Q, \Sigma, \text{true}, \Delta, \{\text{false}\}) \]
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

\[ A = (Q, \Sigma, \text{true}, \Delta, \{\text{false}\}) \]

logical formulae
over program variables

If \( P \subseteq A \), then \( P \) is correct.
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

\[ A = (Q, \Sigma, \text{true}, \Delta, \{\text{false}\}) \]

logical formulae over program variables

program statements taken from \( P \)

If \( P \subseteq A \), then \( P \) is correct.
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

\[ A = (Q, \Sigma, \text{true}, \Delta, \{ \text{false} \}) \]

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

\[ a[i] \neq 0 \land i < j \]
\[ a[i] = 21 \]
\[ a[j] = 0 \]
\[ i := i - 1 \]
\[ j := i + 1 \]
iteratively build Floyd-Hoare automaton

\[ A = (Q, \Sigma, \text{true}, \Delta, \{\text{false}\}) \]

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

\[ \Rightarrow \text{for all } \tau \in A, \text{ Hoare triple} \]
\[ \{\text{true}\} \tau \{\text{false}\} \text{ is valid} \]
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

\[ A = (Q, \Sigma, \text{true}, \Delta, \{\text{false}\}) \]

logical formulae over program variables

program statements taken from \( P \)

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

\[ \Rightarrow \] for all \( \tau \in A \), Hoare triple
\(\{\text{true}\} \ \tau \ \{\text{false}\}\) is valid

If \( P \subseteq A \), then \( P \) is correct.
Trace Abstraction Refinement

iteratively build Floyd-Hoare automaton

\[ A = (Q, \Sigma, \text{true}, \Delta, \{\text{false}\}) \]

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.

\[ \Rightarrow \text{for all } \tau \in A, \text{ Hoare triple } \{\text{true}\} \tau \{\text{false}\} \text{ is valid} \]

If \( P \subseteq A \), then \( P \) is correct.
Trace Abstraction Refinement

program $P$

$A := \emptyset$

$P \subseteq A$ ?

construct $A_\tau$ with $\tau \in A_\tau$

set $A := A \cup A_\tau$

{true} $\tau$ {false} valid?

yes

no

pick error trace $\tau \in P \setminus A$

yes

no

“$P$ is correct”

“$P$ is incorrect”
Partial Order Reduction

error traces:

\[ \tau_1: \ i:=i-1 \ a[j]:=7 \quad j:=i+1 \quad a[i]:=21 \quad a[j]:=0 \quad a[i]==0 \]

\[ \tau_2: \ i:=i-1 \ a[j]:=7 \quad a[i]:=21 \quad j:=i+1 \quad a[j]:=0 \quad a[i]==0 \]

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 \( \tau_1, \tau_2 \) are equivalent (\( \tau_1 \sim_I \tau_2 \)) iff \( \tau_1 = \tau_2 \) or \( \tau_1 = \rho_{ab} \sigma, \tau_2 = \rho_{ba} \sigma \)

where \((a, b) \in I\) or \( \exists \tau' . \tau_1 \sim \tau' \sim \tau_2 \)

Goal: Only analyse one representative of each equivalence class!
error traces:

\[ \tau_1: \quad i:=i-1 \quad a[j]:=7 \quad j:=i+1 \quad a[i]:=21 \quad a[j]:=0 \quad a[i]==0 \]

\[ \tau_2: \quad i:=i-1 \quad a[j]:=7 \quad a[i]:=21 \quad j:=i+1 \quad a[j]:=0 \quad a[i]==0 \]

- 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 \( \tau_1, \tau_2 \) are equivalent (\( \tau_1 \sim_I \tau_2 \)) iff \( \tau_1 = \tau_2 \) or \( \tau_1 = \rho_{ab} \sigma, \tau_2 = \rho_{ba} \sigma \) where \( (a, b) \in I \) or \( \exists \tau' \cdot \tau_1 \sim \tau' \sim \tau_2 \).

Goal: Only analyse one representative of each equivalence class!
error traces:

\[ \tau_1: \begin{array}{llllll} i := & i-1 & a[j] := & 7 & j := & i+1 \end{array} \begin{array}{llllll} a[i] := & 21 & a[j] := & 0 & a[i] := & 0 \end{array} \]
\[ \tau_2: \begin{array}{llllll} i := & i-1 & a[j] := & 7 & a[i] := & 21 \end{array} \begin{array}{llllll} j := & i+1 & a[j] := & 0 & a[i] := & 0 \end{array} \]

- 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
error traces:

\[ \tau_1: \ i:=i-1 \ a[j]:=7 \quad j:=i+1 \quad a[i]:=21 \quad a[j]:=0 \quad a[i]==0 \]
\[ \tau_2: \ i:=i-1 \ a[j]:=7 \quad a[i]:=21 \quad j:=i+1 \quad a[j]:=0 \quad a[i]==0 \]

- 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 \( \tau_1, \tau_2 \) are equivalent \((\tau_1 \sim_I \tau_2)\) iff

\[ \tau_1 = \tau_2 \quad \text{or} \quad \tau_1 = \rho_ab\sigma, \tau_2 = \rho_ba\sigma \quad \text{where} \quad (a, b) \in I \quad \text{or} \quad \exists \tau'. \tau_1 \sim \tau' \sim \tau_2 \]
error traces:

\[ \tau_1: \quad i:=i-1 \quad a[j]:=7 \quad j:=i+1 \quad a[i]:=21 \quad a[j]:=0 \quad a[i]==0 \]
\[ \tau_2: \quad i:=i-1 \quad a[j]:=7 \quad a[i]:=21 \quad j:=i+1 \quad a[j]:=0 \quad a[i]==0 \]

- 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 \( \tau_1, \tau_2 \) are equivalent \((\tau_1 \sim_I \tau_2)\) iff

\[ \tau_1 = \tau_2 \quad \text{or} \quad \tau_1 = \rho a b \sigma, \tau_2 = \rho b a \sigma \quad \text{where } (a, b) \in I \quad \text{or} \quad \exists \tau'. \tau_1 \sim \tau' \sim \tau_2 \]

- 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

New proof criterion:

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

(for suitable commutativity relation $I$)

(All traces equivalent to some $\tau \in A$)

Algorithmic Check:

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

$P' \subseteq A$

Hence: Compute a (regular) reduction $P'$ and check $P'$ sufficient but (necessarily) incomplete more general than checking $P \subseteq A$.
Partial Order Reduction

New proof criterion:

If $P \subseteq cl_1(A)$, then $P$ is correct.

(for suitable commutativity relation $I$)

Algorithmic Check:

$P \subseteq cl_1(A) \iff \exists$ reduction $P'$ of $P$ s.t. $P' \subseteq A$
Partial Order Reduction

New proof criterion:

If \( P \subseteq \text{cl}_I(A) \), then \( P \) is correct.

(for suitable commutativity relation \( I \))

Algorithmic Check:

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

Closure of \( A \)

All traces equivalent to some \( \tau \in A \)
Partial Order Reduction

New proof criterion:

If \( P \subseteq \text{cl}_I(A) \), then \( P \) is correct.

(for suitable commutativity relation \( I \))

Algorithmic Check:

\[ P \subseteq \text{cl}_I(A) \iff \exists \text{ reduction } P' \text{ of } P \text{ s.t. } P' \subseteq A \]

i.e. \( \text{cl}_I(P') = P \)

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

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

Definition:

\( a \text{ and } b \text{ commute iff } Ja \circ Kb = Kb \circ Ja \)

Semantics of statement relation over program states

For example:

\( Ja[i] := 21 K = \{ \langle s, s' \rangle \mid s' = s \{ a \mapsto \text{store}(s(a), s(i), 21) \} \} \)

\( j := i + 1 K = \{ \langle s, s' \rangle \mid s' = s \{ j \mapsto s(i) + 1 \} \} \)

Therefore

\( Ja[i] := 21 K \circ j := i + 1 K = j := i + 1 K \circ Ja[i] := 21 K \)

Combined with Trace Abstraction Refinement by Cassez et al.
Concrete Commutativity

Definition:

\[ a \text{ and } b \text{ commute } \iff [a] \circ [b] = [b] \circ [a] \]
Concrete Commutativity

Definition:

$a$ and $b$ commute iff $[a] \circ [b] = [b] \circ [a]$

semantics of statement $a$ relation over program states

For example:

$J_a[i] := 21$  
$J_j := i + 1$

Therefore $J_a[i] := 21 \circ J_j := i + 1 = J_j := i + 1 \circ J_a[i] := 21$
Concrete Commutativity

Definition:

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

For example:

\[ [a[i]:=21] = \{(s, s') | s' = s\{a \mapsto \text{store}(s(a), s(i), 21)\}\} \]
\[ [j:=i+1] = \{(s, s') | s' = s\{j \mapsto s(i) + 1\}\} \]

Therefore

\[ [a[i]:=21] \circ [j:=i+1] = [j:=i+1] \circ [a[i]:=21] \]
Concrete Commutativity

Definition:

\( a \) and \( b \) commute \( \iff [a] \circ [b] = [b] \circ [a] \)

Semantics of statement \( a \) relation over program states

For example:

\[
\begin{align*}
[a[i]:=21] &= \{(s, s') \mid s' = s\{a \mapsto \text{store}(s(a), s(i), 21)\}\} \\
[j:=i+1] &= \{(s, s') \mid s' = s\{j \mapsto s(i) + 1\}\}
\end{align*}
\]

Therefore

\[
[a[i]:=21] \circ [j:=i+1] = [j:=i+1] \circ [a[i]:=21]
\]

Combined with Trace Abstraction Refinement by Cassez et al.

\[\text{References}\]

Conditional Commutativity

Do \(a[i] := 21\) and \(a[j] := 0\) commute?
Conditional Commutativity

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

- $a[i] := 21$ ; $a[j] := 0$
- $a[j] := 0$ ; $a[i] := 21$

If $i = j$ then $a[i] = 0$

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 $\varphi$):

$$(J_a K \circ J_b K) \setminus (\varphi \times \text{true}) = (J_b K \circ J_a K) \setminus (\varphi \times \text{true})$$
Conditional Commutativity

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

- $a[i] := 21 ; a[j] := 0$
- $a[j] := 0 ; a[i] := 21$

If $i = j$ then $a[i] = 0$

- In general: No!
Conditional Commutativity

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

$\begin{align*}
& a[i] := 21 \quad ; \quad a[j] := 0 \\
& \text{if } i \neq j \text{ then } a[i] = 0 \\
& i < j \\
& \text{true} \quad \checkmark \\
& a[j] := 0 \quad ; \quad a[i] := 21 \\
\end{align*}$

- 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?

\[
\begin{align*}
\text{a[i]:=21 ; a[j]:=0} \\
\text{a[j]:=0 ; a[i]:=21}
\end{align*}
\]

- 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

Conditional Commutativity

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

\[
\begin{align*}
a[i] := 21 & ; a[j] := 0 \\
\quad s & \quad s' \\
i < j & \quad a[j] := 0 & ; a[i] := 21
\end{align*}
\]

- 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 \( \varphi \)):
  \[
  ([a] \circ [b]) \cap (\varphi \times \text{true}) = ([b] \circ [a]) \cap (\varphi \times \text{true})
  \]
Do $a[i] := 21$ and $a[j] := 7$ commute?

If $i = j$ then $a[i] \neq 0 \checkmark$

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

- $a[i] := 21$ ; $a[j] := 7$
- $s$ — $s'$
- if $i = j$ then $a[j] := 7$ ; $a[i] := 21$
- $a[j] := 7$ ; $a[i] := 21$ then $a[i] = 7$

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

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

$\begin{align*}
   & a[i] := * ; a[j] := * \\
   \Rightarrow & \text{No: order matters in case } i = j \text{ (which is possible)} \\
   \Rightarrow & \text{In our program: We only care that } a[i] \neq 0 \\
   \Rightarrow & \text{find abstractions}
\end{align*}$

Abstract Commutativity

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

- $a[i]:= \ast$ ; $a[j]:= \ast$

\[ s \] \[ s' \]

$a[j]:=\ast$ ; $a[i]:=\ast$

- $a[j]:=\ast$ ; $a[i]:=\ast$

- If $i=j$ then $a[i]=7$

- $a[i] \neq 0 \Rightarrow$ find abstractions

- 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 $A$
Do $a[i] := 21$ and $a[j] := 7$ commute?

- $a[i] := *$ with $a[i] \neq 0$ ; $a[j] := *$ with $a[j] \neq 0$
- $a[j] := *$ with $a[j] \neq 0$ ; $a[i] := *$ with $a[i] \neq 0$

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

Abstraction sometimes loses commutativity

Combine (conditional) concrete and abstract commutativity

New proof criterion: $P \subseteq cl_{concr}(cl_{abstr}(A))$, then $P$ is correct.

⇒ develop new partial order reduction algorithms for sufficient check

Very general criterion:

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

$P \subseteq cl_{abstr}(A) = \Rightarrow P \subseteq cl_{concr}(cl_{abstr}(A))$
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:

\[
\text{If } P \subseteq \text{cl}_{\text{concr}} (\text{cl}_{\text{abstr}}(A)), \text{ then } P \text{ is correct.}
\]

⇒ 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:

If \( P \subseteq \text{cl}_{\text{concr}}(\text{cl}_{\text{abstr}}(A)) \), then \( P \) is correct.

\[ \Rightarrow \text{develop new partial order reduction algorithms for sufficient check} \]

Very general criterion:

\[
\begin{align*}
P \subseteq \text{cl}_{\text{concr}}(A) & \Rightarrow P \subseteq \text{cl}_{\text{concr}}(\text{cl}_{\text{abstr}}(A)) \\
P \subseteq \text{cl}_{\text{abstr}}(A) & \Rightarrow P \subseteq \text{cl}_{\text{concr}}(\text{cl}_{\text{abstr}}(A))
\end{align*}
\]
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
investigate other partial order techniques
Empirical evaluation: effectiveness for verification
Theoretical complexity result
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 Owiciki-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
Thank you for your attention. Questions?