

The Ghosts of Empires: Extracting Modularity from Interleaving-Based Proofs

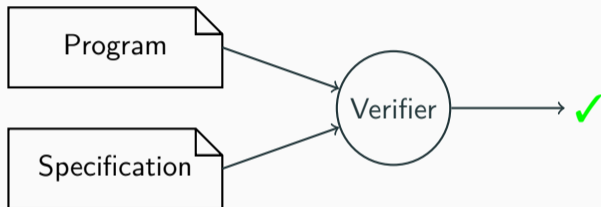
POPL 2026 – 16th January 2026

Frank Schüssele¹ Matthias Zumkeller¹ Miriam Lagunes-Rochin¹ Dominik Klumpp²

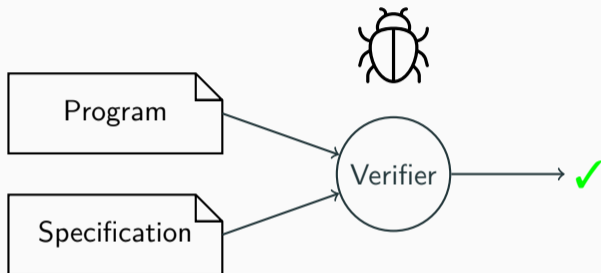
¹University of Freiburg, Freiburg im Breisgau, Germany

²LIX – CNRS – École Polytechnique, Palaiseau, France

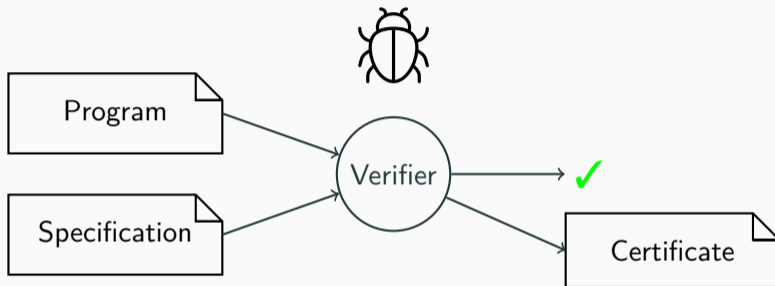
Motivation



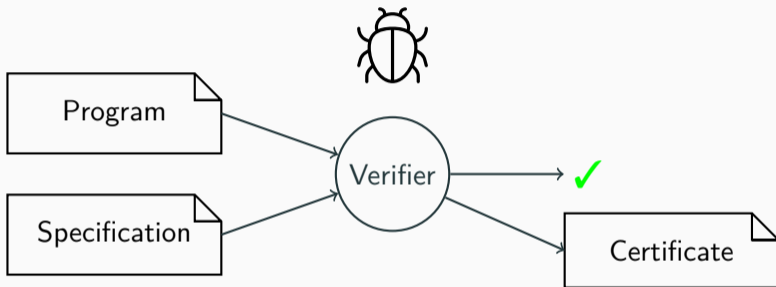
Motivation



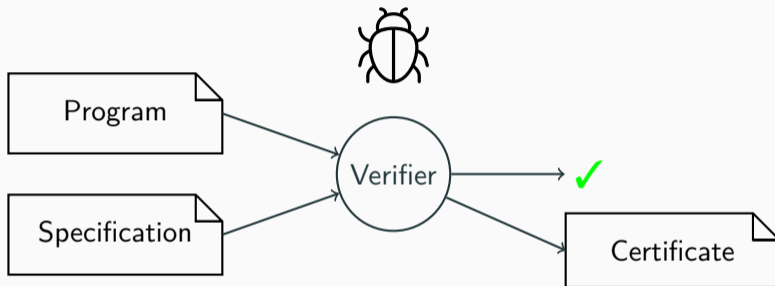
Motivation



Motivation

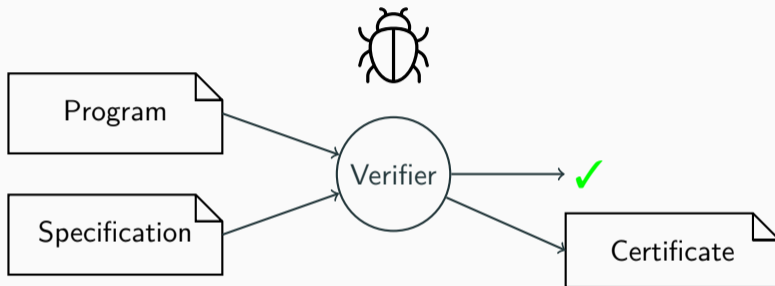


Goal: certificates for concurrent programs



Goal: certificates for concurrent programs

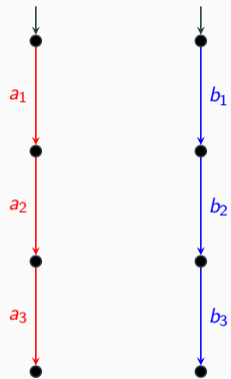
- compact



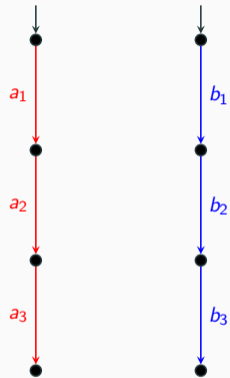
Goal: certificates for concurrent programs

- compact
- efficiently checkable

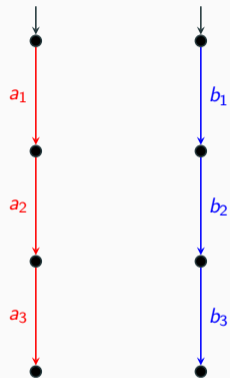
Interleaving-based proofs



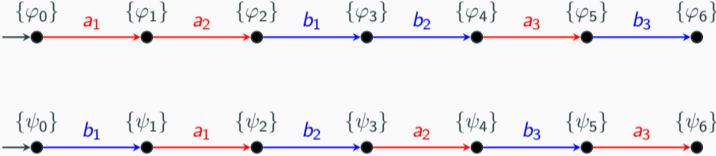
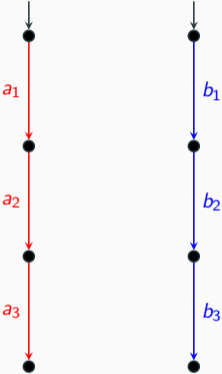
Interleaving-based proofs



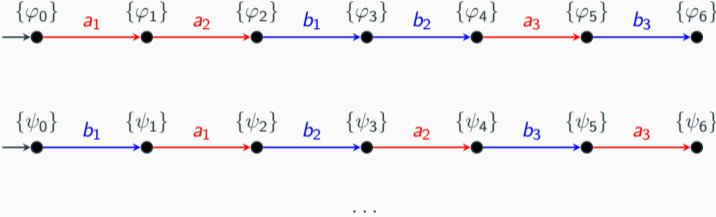
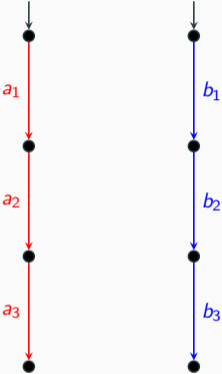
Interleaving-based proofs



Interleaving-based proofs

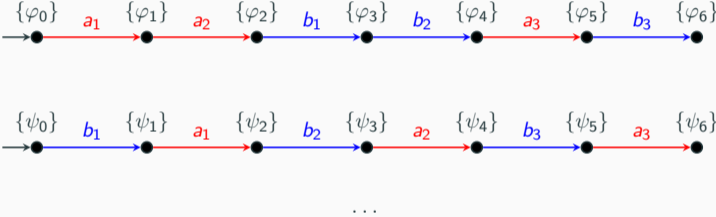
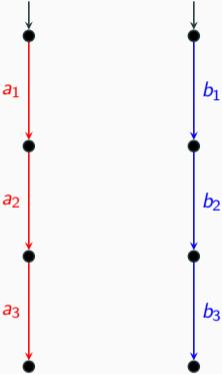


Interleaving-based proofs



Interleaving-based proof: finite set $\{\varphi_0, \dots, \varphi_6, \psi_0, \dots, \psi_6, \dots\}$

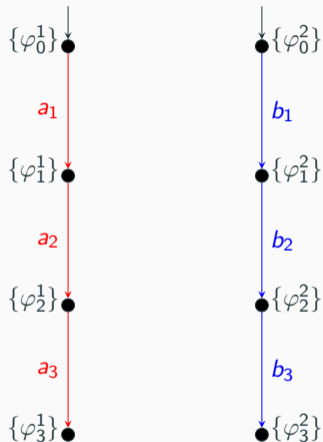
Interleaving-based proofs



Interleaving-based proof: finite set $\{\varphi_0, \dots, \varphi_6, \psi_0, \dots, \psi_6, \dots\}$

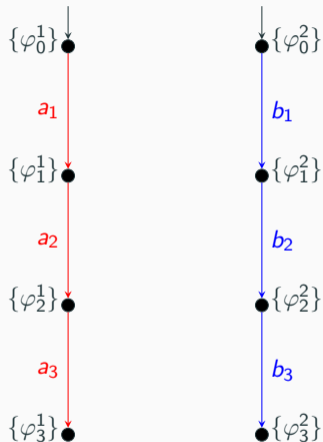
Problem: Validation requires exponentially many Hoare triple checks

Thread-modular proofs¹



¹Susan S. Owicki and David Gries. “**Verifying Properties of Parallel Programs: An Axiomatic Approach**”. In: *Commun. ACM* 19.5 (1976), pp. 279–285. DOI: 10.1145/360051.360224.

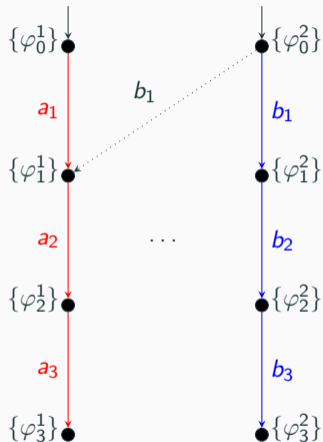
Thread-modular proofs¹



For validation only quadratically many Hoare triple checks

¹Susan S. Owicki and David Gries. “**Verifying Properties of Parallel Programs: An Axiomatic Approach**”. In: *Commun. ACM* 19.5 (1976), pp. 279–285. DOI: 10.1145/360051.360224.

Thread-modular proofs¹

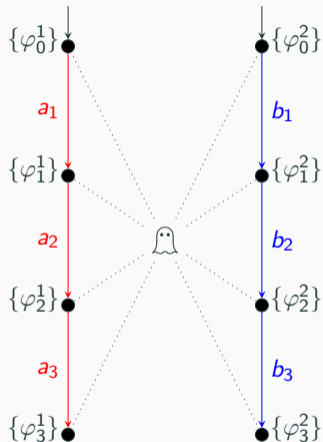


For validation only quadratically many Hoare triple checks

- non-interference, e.g., $\{\varphi_1^1 \wedge \varphi_0^2\} b_1 \{\varphi_1^1\}$

¹Susan S. Owicki and David Gries. “**Verifying Properties of Parallel Programs: An Axiomatic Approach**”. In: *Commun. ACM* 19.5 (1976), pp. 279–285. DOI: 10.1145/360051.360224.

Thread-modular proofs¹



For validation only quadratically many Hoare triple checks

- non-interference, e.g., $\{\varphi_1^1 \wedge \varphi_0^2\} b_1 \{\varphi_1^1\}$

For completeness: ghost variables to encode interleaving information

¹Susan S. Owicki and David Gries. “**Verifying Properties of Parallel Programs: An Axiomatic Approach**”. In: *Commun. ACM* 19.5 (1976), pp. 279–285. DOI: 10.1145/360051.360224.

Example

```

                                 $l_0$ : assume  $x > 0$ 
 $l_1$ :   y := x
 $l_2$ :   while (*) {
 $l_3$ :     assume  $y > 0$ 
 $l_4$ :     y := y - 1
 $l_5$ :   }
 $l_6$ :   x := x + 1
 $l_7$ :   assert  $y < x$ 
      ||
 $l_8$ :   while (*) {
 $l_9$ :     assume  $x \neq 0$ 
 $l_{10}$ :    z := z / x
 $l_{11}$ :   }
 $l_{12}$ :   x := x + 1
                                 $l_{13}$ : assert  $x > 2$ 

```

Example

```
l0: assume x > 0  $\wedge$   $\neg$   $\text{ghost}_1$   $\wedge$   $\neg$   $\text{ghost}_2$ 

l1:   y := x
l2:   while (*) {
l3:     assume y > 0
l4:     y := y - 1
l5:   }
l6:   x := x + 1;  $\text{ghost}_1 := \top$ 
l7:   assert y < x

l8:   while (*) {
l9:     assume x != 0
l10:    z := z / x
l11:  }
l12:  x := x + 1;  $\text{ghost}_2 := \top$ 

l13: assert x > 2
```

Example

```
l0: assume x > 0 ∧ ¬ $\phi_1$  ∧ ¬ $\phi_2$ 
l1:   y := x
l2:   while (*) {
l3:     assume y > 0
l4:     y := y - 1
l5:   }
l6:   x := x + 1;  $\phi_1$  := T
l7:   assert y < x
l8:   while (*) {
l9:     assume x != 0
l10:    z := z / x
l11:  }
l12:  x := x + 1;  $\phi_2$  := T
l13:  assert x > 2
```

$\neg\phi_1 \wedge (\neg\phi_2 \rightarrow x > 0) \wedge (\phi_2 \rightarrow x > 1) \wedge y \leq x$

Example

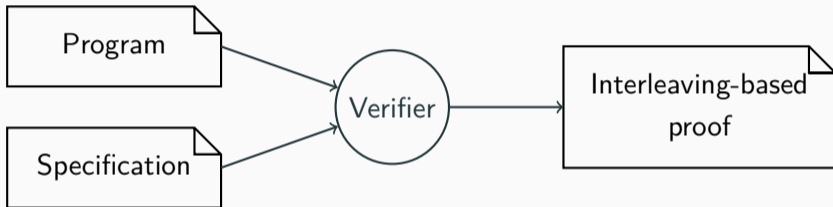
```
l0: assume x > 0 ∧ ¬ $\phi_1$  ∧ ¬ $\phi_2$ 
l1:   y := x
l2:   while (*) {
l3:     assume y > 0
l4:     y := y - 1
l5:   }
l6:   x := x + 1;  $\phi_1$  := T
l7:   assert y < x
```

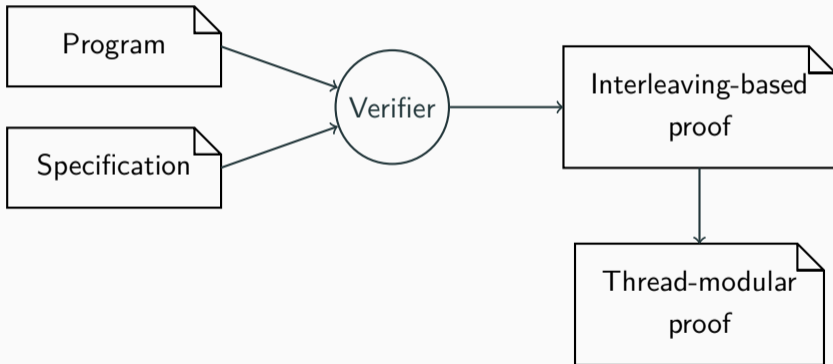
```
l8:   while (*) {
l9:     assume x != 0
l10:    z := z / x
l11:  }
l12:  x := x + 1;  $\phi_2$  := T
```

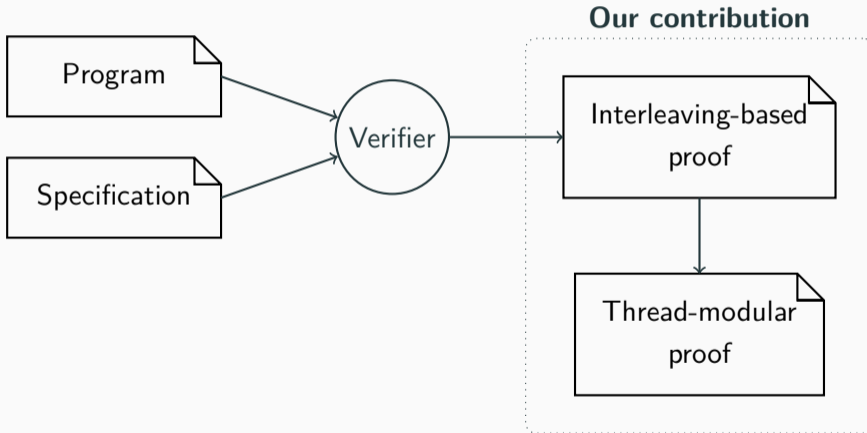
$\neg\phi_1 \wedge (\neg\phi_2 \rightarrow x > 0) \wedge (\phi_2 \rightarrow x > 1) \wedge y \leq x$

$\phi_2 \wedge (\neg\phi_1 \rightarrow x > 1) \wedge (\phi_1 \rightarrow x > 2)$

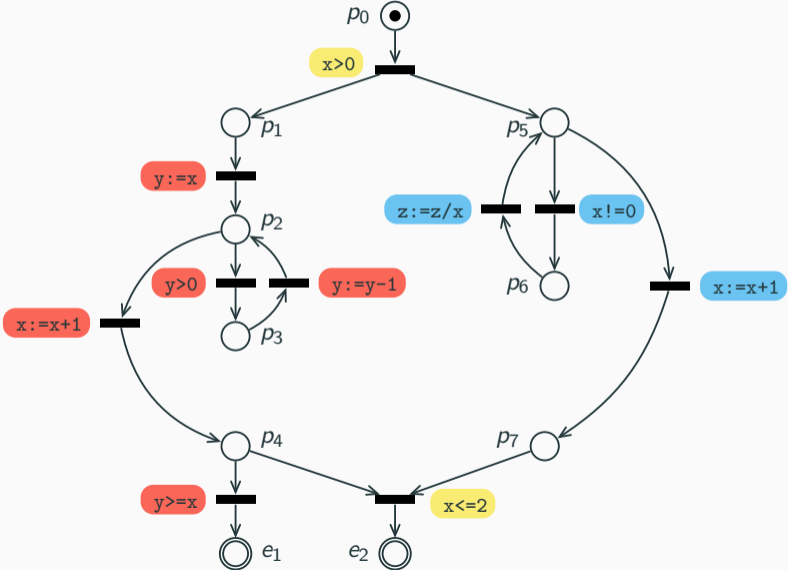
```
l13: assert x > 2
```



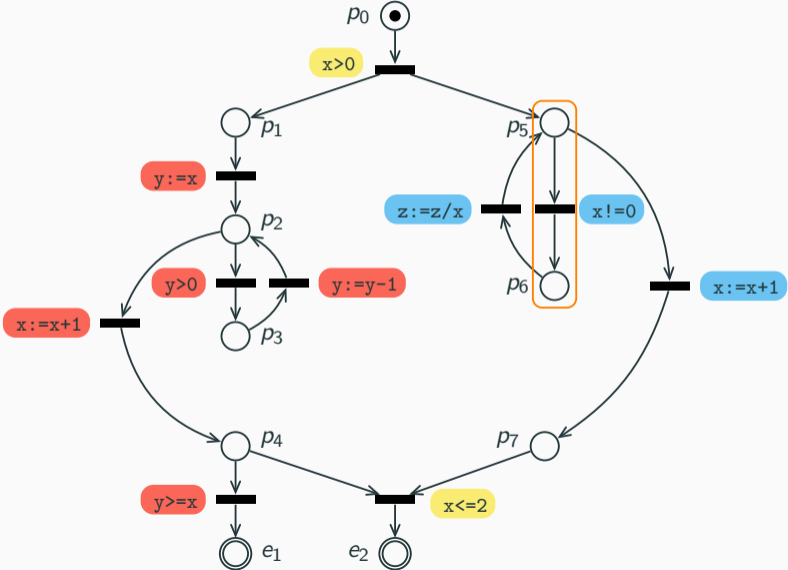




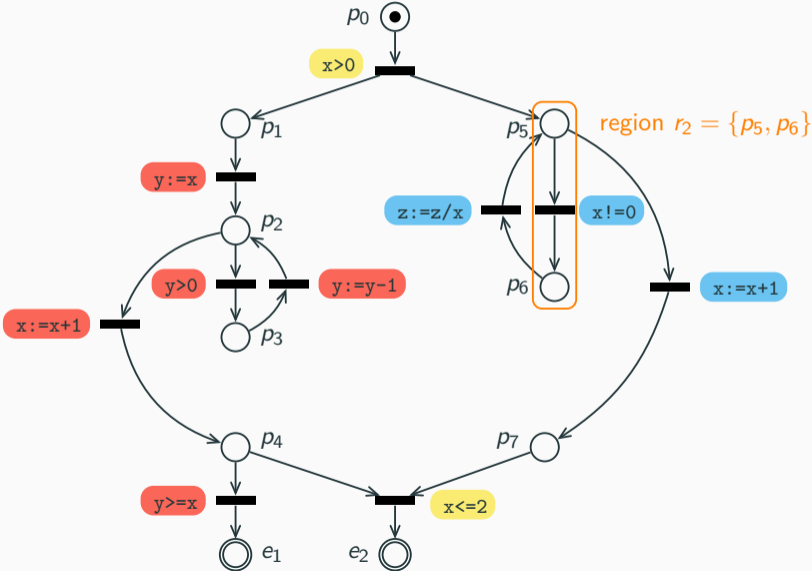
Petri Program



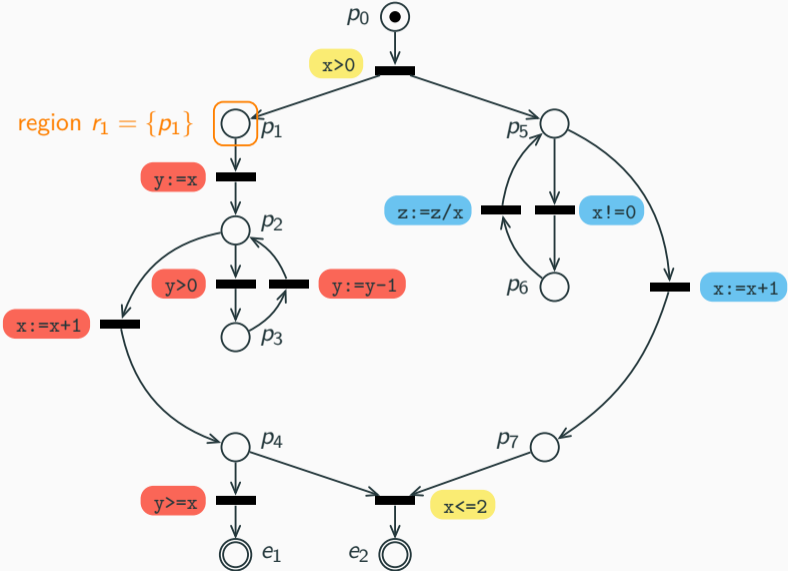
Petri Program



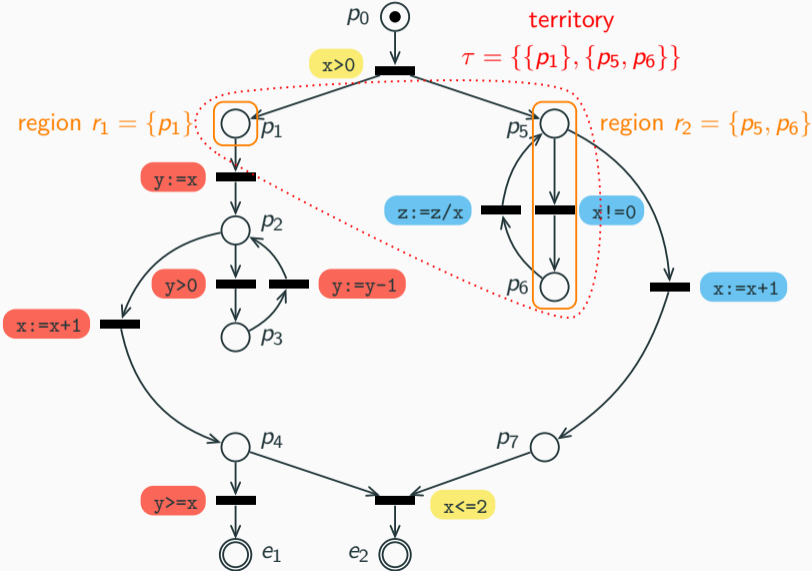
Petri Program – Regions



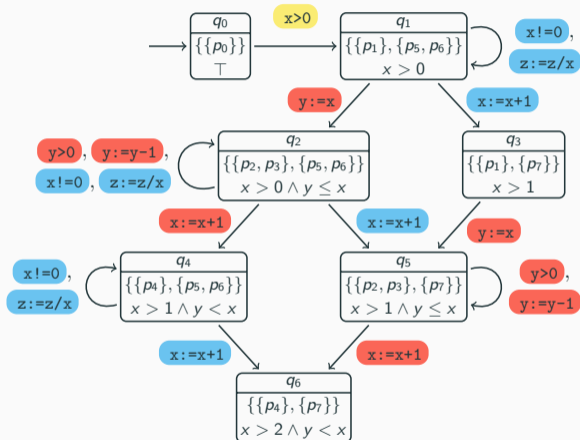
Petri Program – Regions



Petri Program – Regions & Territories

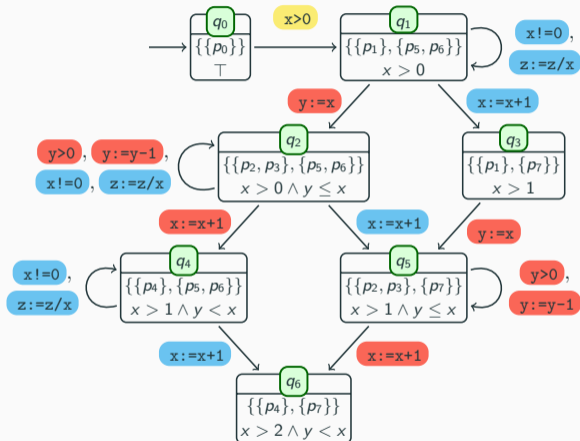


Empires: An Abstract Representation of Interleaving Information



Empire $E = (Q, q_0, \delta, terr, law)$

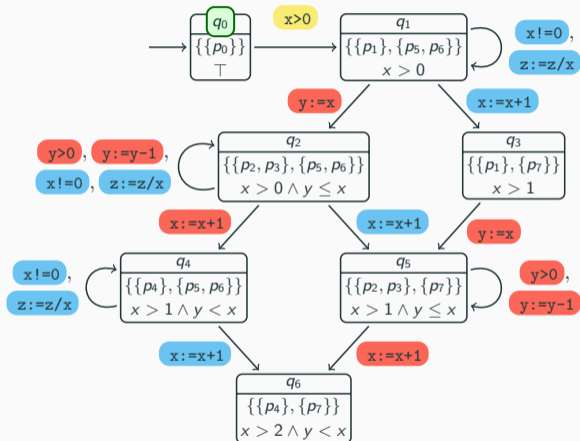
Empires: An Abstract Representation of Interleaving Information



Empire $E = (Q, q_0, \delta, terr, law)$

- finite set of states Q

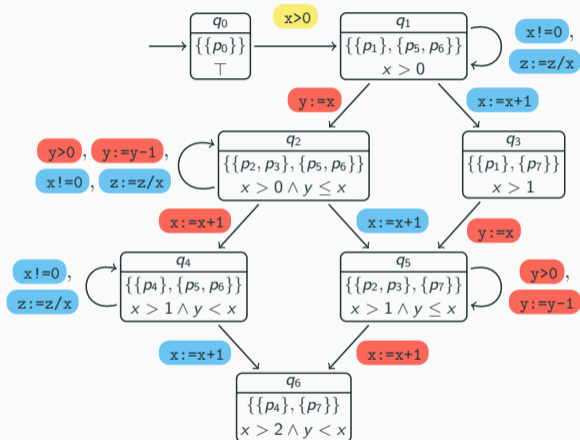
Empires: An Abstract Representation of Interleaving Information



Empire $E = (Q, q_0, \delta, terr, law)$

- finite set of states Q
- initial state $q_0 \in Q$

Empires: An Abstract Representation of Interleaving Information

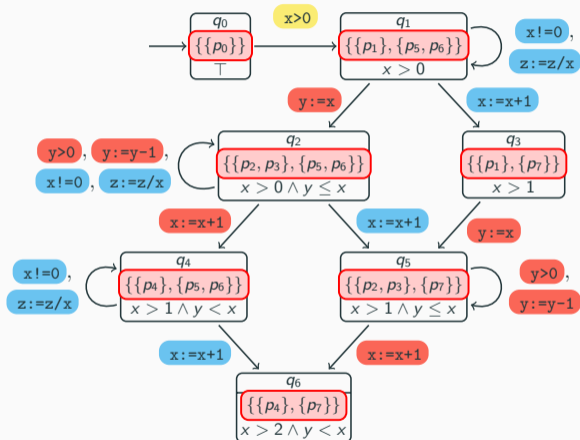


Empire $E = (Q, q_0, \delta, terr, law)$

- finite set of states Q
- initial state $q_0 \in Q$
- transition function

$$\delta : Q \times T \rightarrow Q$$

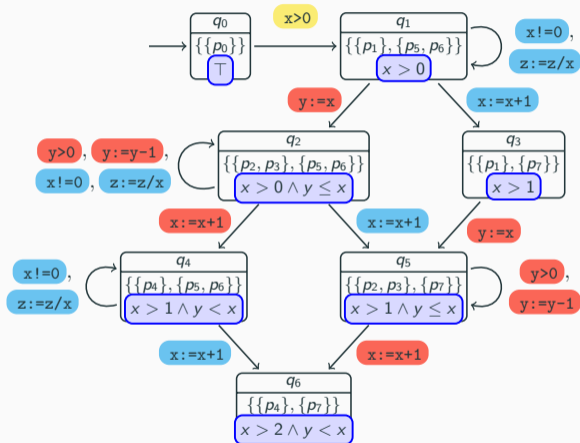
Empires: An Abstract Representation of Interleaving Information



Empire $E = (Q, q_0, \delta, terr, law)$

- finite set of states Q
- initial state $q_0 \in Q$
- transition function $\delta : Q \times T \rightarrow Q$
- territory mapping $terr : Q \rightarrow \mathbf{Terr}$

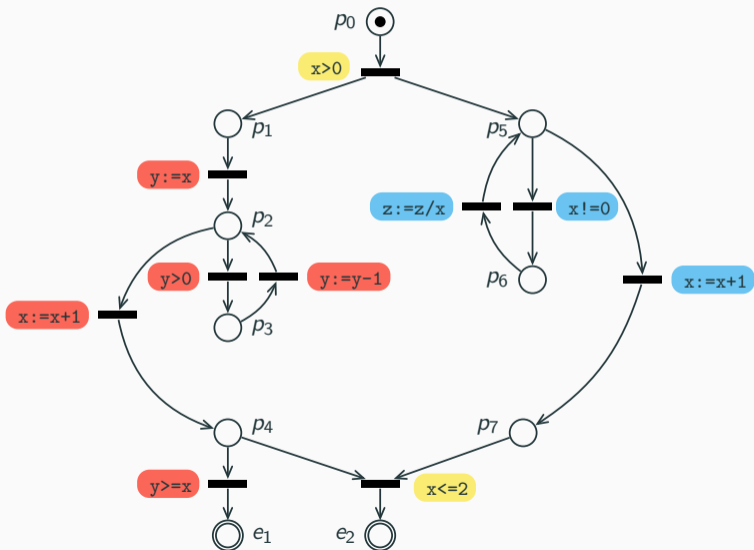
Empires: An Abstract Representation of Interleaving Information



Empire $E = (Q, q_0, \delta, terr, law)$

- finite set of states Q
- initial state $q_0 \in Q$
- transition function $\delta : Q \times T \rightarrow Q$
- territory mapping $terr : Q \rightarrow \mathbf{Terr}$
- law mapping $law : Q \rightarrow \mathbf{Fm}(V_P)$

Empire algorithm



Interleaving-based proof:

\top

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

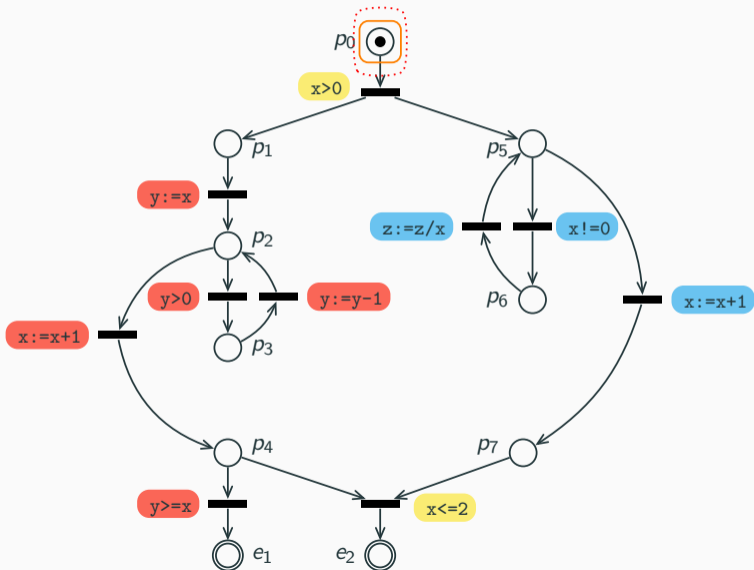
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

\perp

Empire algorithm



Interleaving-based proof:

\top

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

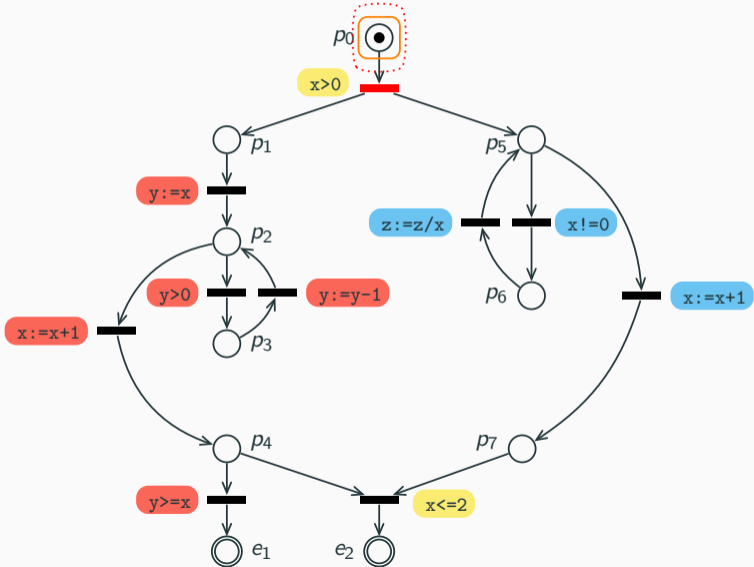
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

\perp

Empire algorithm



Interleaving-based proof:

\top

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

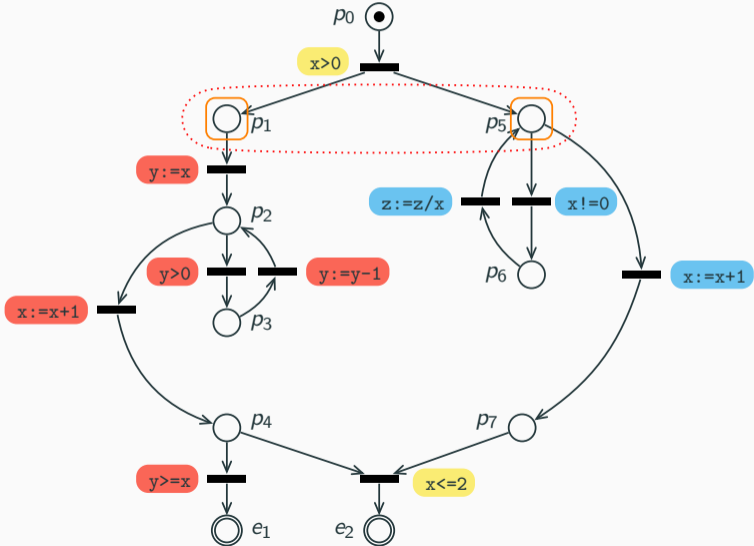
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

\perp

Empire algorithm



Interleaving-based proof:

⊤

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

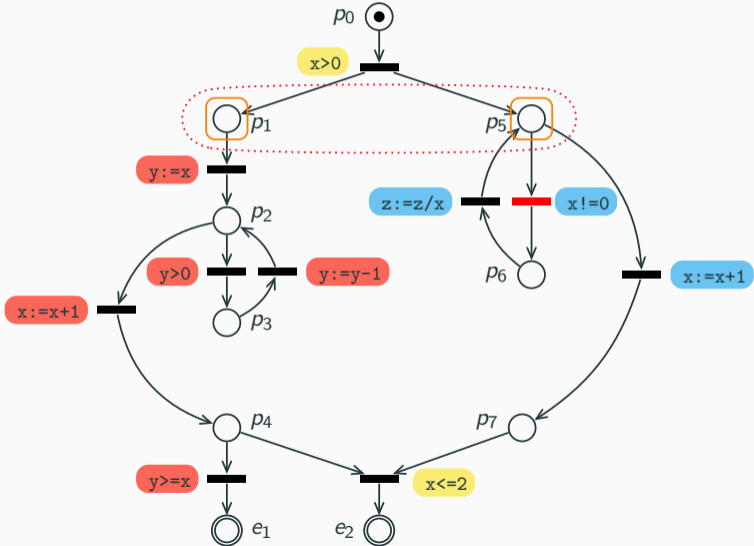
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

⊥

Empire algorithm



Interleaving-based proof:

⊤

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

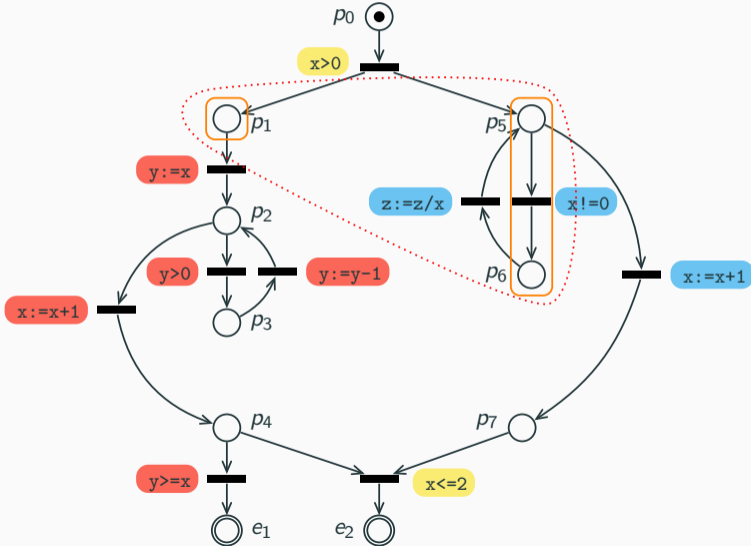
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

⊥

Empire algorithm



Interleaving-based proof:

⊤

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

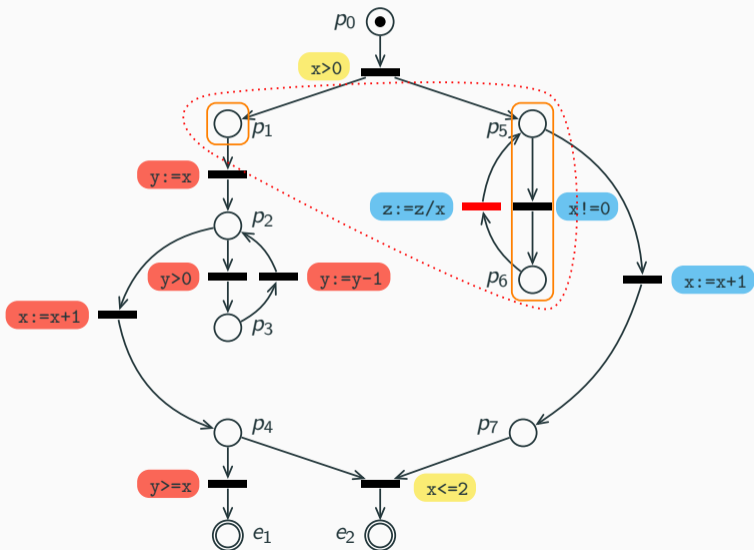
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

⊥

Empire algorithm



Interleaving-based proof:

\top

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

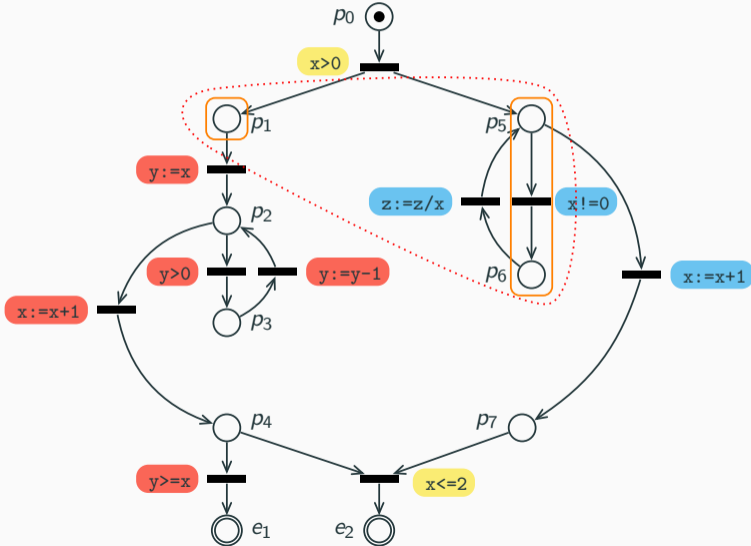
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

\perp

Empire algorithm



Interleaving-based proof:

⊤

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

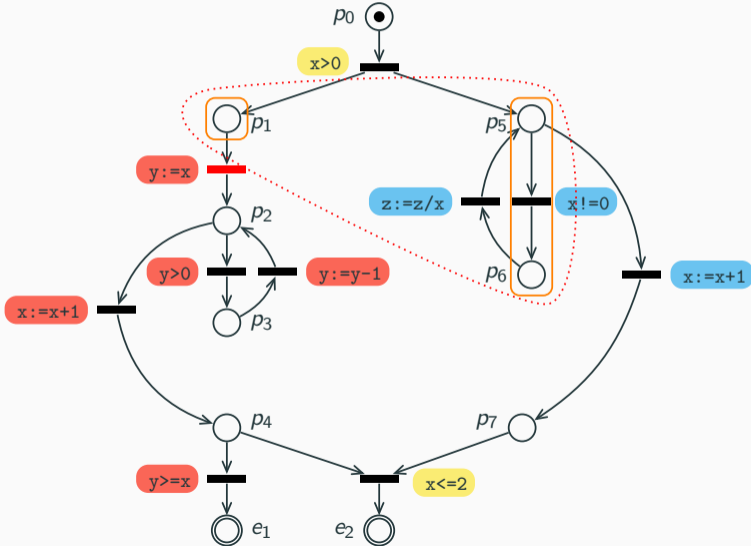
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

⊥

Empire algorithm



Interleaving-based proof:

⊤

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

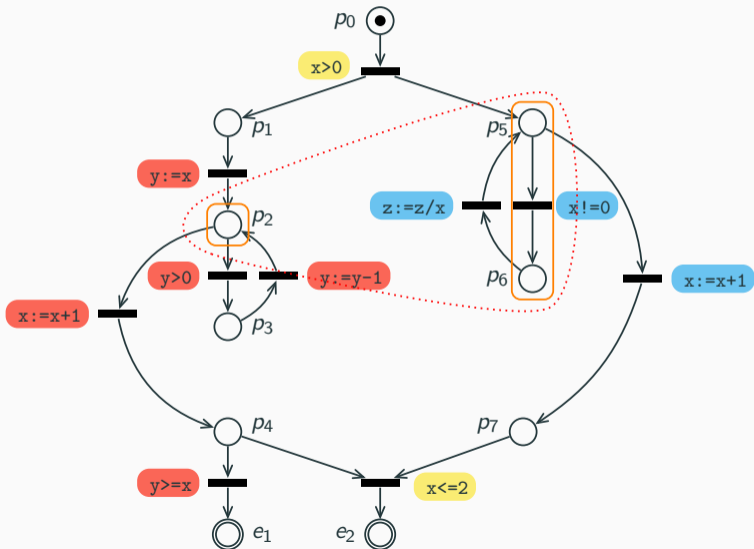
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

⊥

Empire algorithm



Interleaving-based proof:

\top

$x > 0$

$x > 0 \wedge y \leq x$

$x > 1 \wedge y < x$

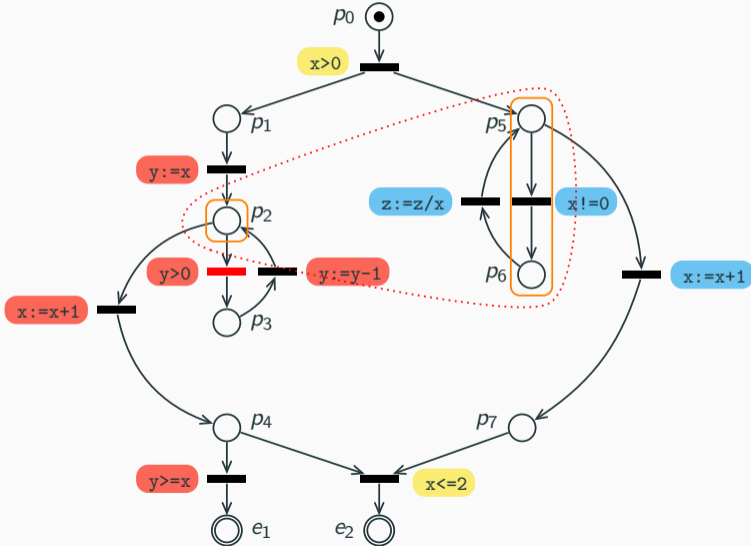
$x > 2 \wedge y < x$

$x > 1$

$x > 1 \wedge y \leq x$

\perp

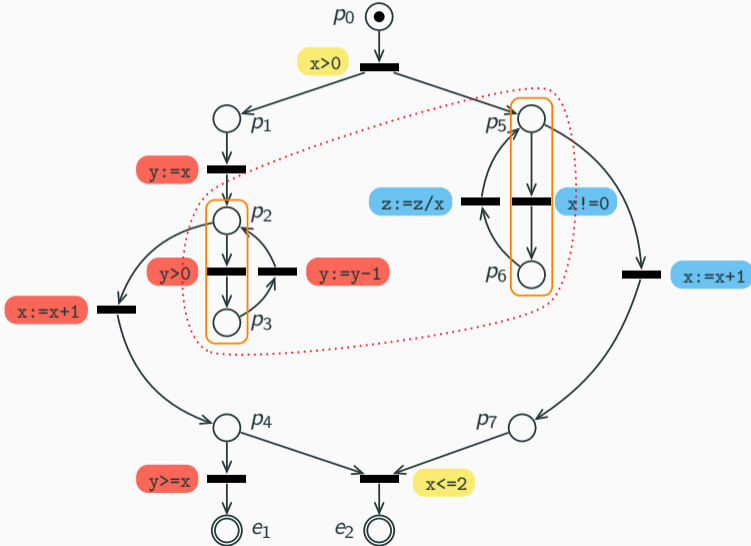
Empire algorithm



Interleaving-based proof:

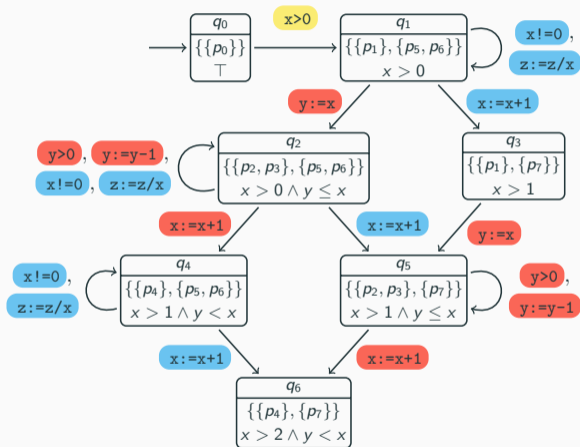
⊤
 $x > 0$
 $x > 0 \wedge y \leq x$
 $x > 1 \wedge y < x$
 $x > 2 \wedge y < x$
 $x > 1$
 $x > 1 \wedge y \leq x$
 ⊥

Empire algorithm

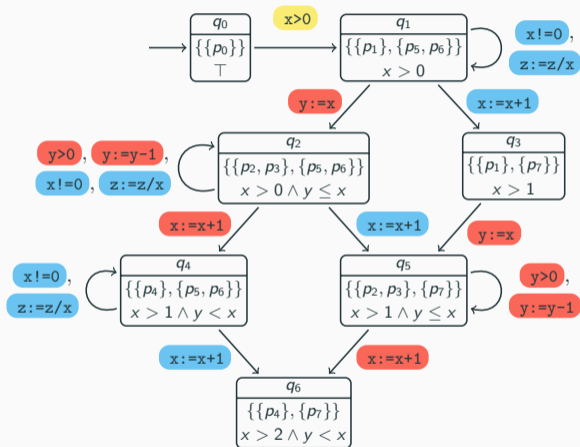


Interleaving-based proof:

⊤
 $x > 0$
 $x > 0 \wedge y \leq x$
 $x > 1 \wedge y < x$
 $x > 2 \wedge y < x$
 $x > 1$
 $x > 1 \wedge y \leq x$
 ⊥

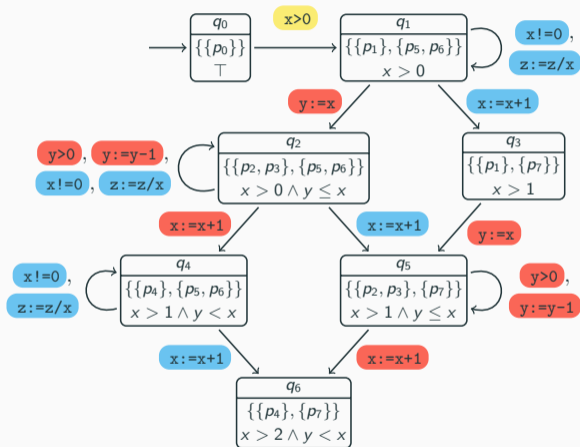


How to convert to thread-modular proof?



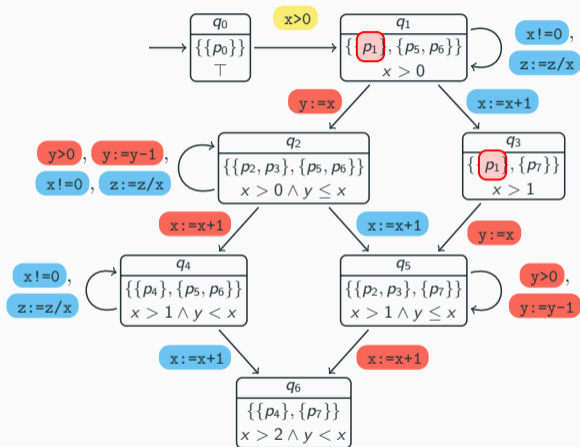
How to convert to thread-modular proof?

- $\text{ghost} \in Q$



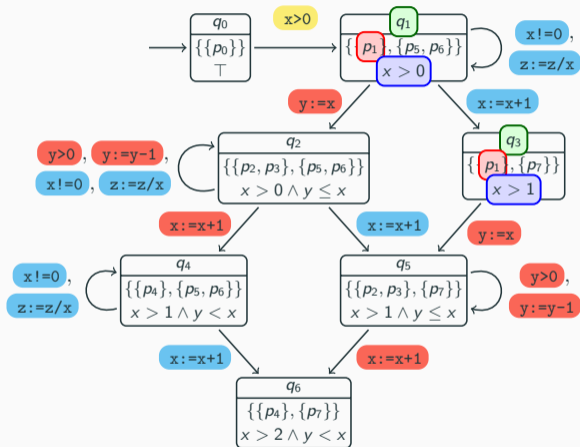
How to convert to thread-modular proof?

- $\text{ghost} \in Q$
- invariant ω : disjunction



How to convert to thread-modular proof?

- $\text{ghost} \in Q$
- invariant ω : disjunction

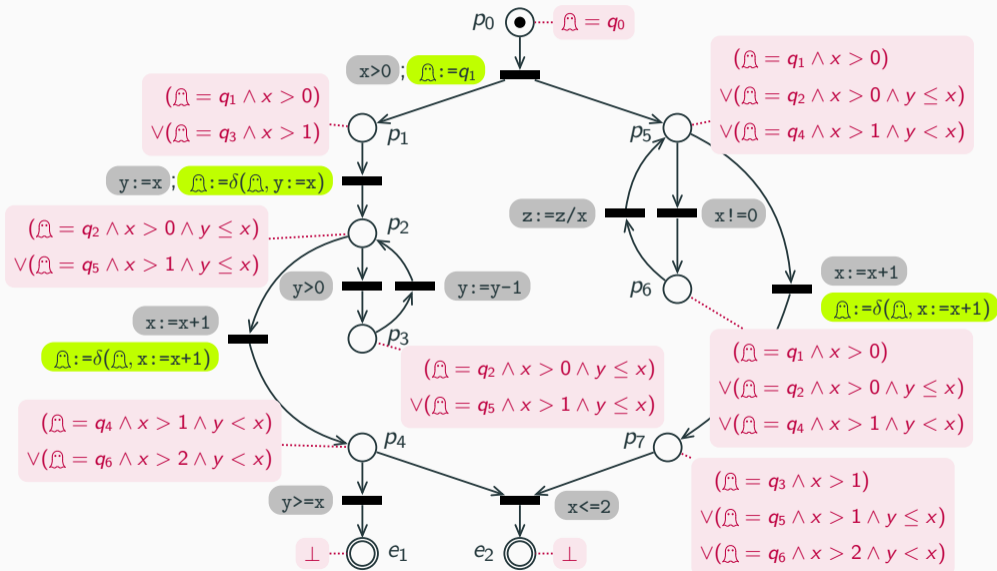


How to convert to thread-modular proof?

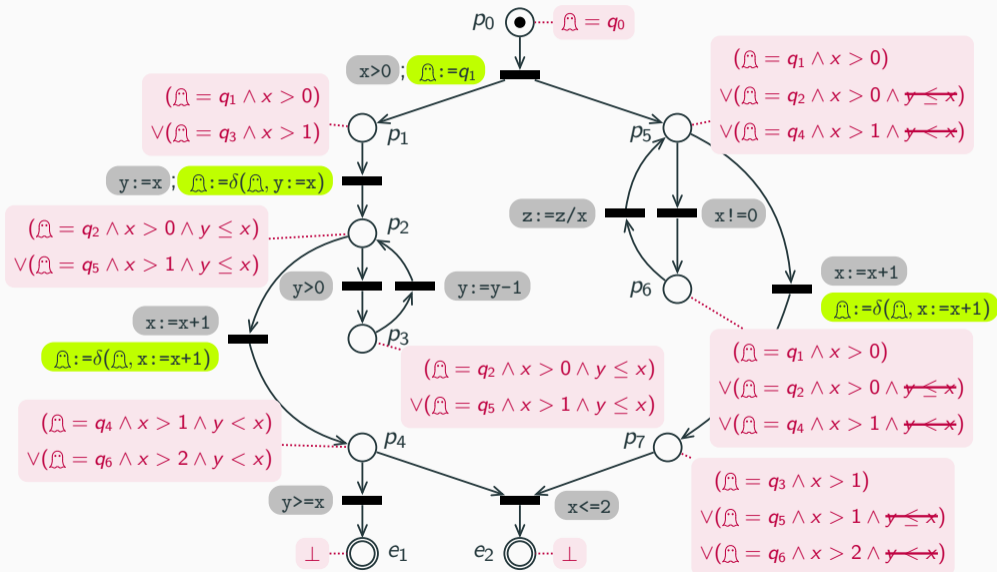
- $\text{ghost} \in Q$
- invariant ω : disjunction

$$\omega(p_1) = (\text{ghost} = q_1 \wedge x > 0) \vee (\text{ghost} = q_3 \wedge x > 1)$$

Imperial thread-modular proof



Imperial thread-modular proof with focus



- `ULTIMATE AUTOMIZER` as verifier to compute interleaving-based proofs

- `ULTIMATE AUTOMIZER` as verifier to compute interleaving-based proofs
- Approach applied to these proofs, compared to naïve approach as a baseline

- `ULTIMATE AUTOMIZER` as verifier to compute interleaving-based proofs
- Approach applied to these proofs, compared to naïve approach as a baseline
- Benchmarks: 1109 concurrent C programs from SV-COMP 2025

Highlights:

Highlights: (compared to naïve approach)

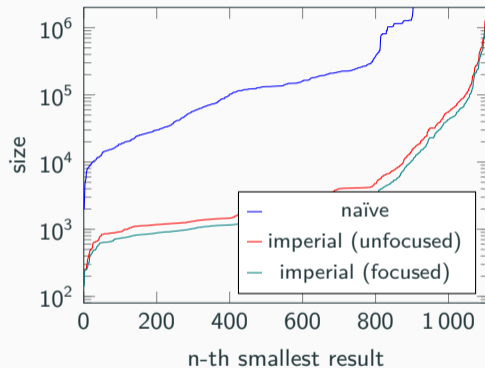
- computation time: 119 times faster

Highlights: (compared to naïve approach)

- computation time: 119 times faster
- size: 15 times smaller

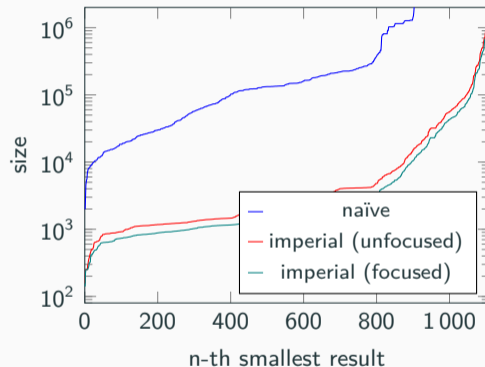
Highlights: (compared to naïve approach)

- computation time: 119 times faster
- size: 15 times smaller



Highlights: (compared to naïve approach)

- computation time: 119 times faster
- size: 15 times smaller
- validation time: 26 times faster



- Algorithm: interleaving-based proofs to thread-modular proofs

- Algorithm: interleaving-based proofs to thread-modular proofs
- ... to be used with existing verification techniques

- Algorithm: interleaving-based proofs to thread-modular proofs
- ... to be used with existing verification techniques
- Evaluation: feasible in practice (fast to generate and validate)

- Algorithm: interleaving-based proofs to thread-modular proofs
- ... to be used with existing verification techniques
- Evaluation: feasible in practice (fast to generate and validate)
- Future work: extend focus on ghost updates (for smaller proofs)

Evaluation

	naïve	imperial	
		unfocused	focused
generation			
# successful	904	1 109	1 109
# timeout	194	0	0
# out of memory	10	0	0
avg. generation time (s)	39.29	0.33	0.38
generation & validation			
# successful (valid)	316	941	949
# timeout	781	148	130
# out of memory	8	0	0
# unknown	0	19	28
avg. validation time (s)	55.20	2.09	1.54

Evaluation: Ghost Updates

