Verification of Concurrent Algorithms: an Abstraction Refinement Approach

Giorgio Delzanno[1] Frédéric Haziza[2] Ahmed Rezine[2]

[1]Università di Genova

[2]Uppsala Universitet

MCC Uppsala 2009

Outline

Introduction

Concurrent Algorithms

Symbolic Representation and Abstraction for Verification

Counter Example Guided Abstraction Refinement

Conclusion

Outline

Introduction

Concurrent Algorithms

Symbolic Representation and Abstraction for Verification

Counter Example Guided Abstraction Refinement

Conclusion

Introduction: parallelization

Exponential increase in processors speed has reached its technological limits

- Increase the parallelization

Introduction: parallelization

Exponential increase in processors speed has reached its technological limits

- Increase the parallelization
- Divide the workload on parallel threads

Introduction: parallelization

Exponential increase in processors speed has reached its technological limits

- Increase the parallelization
- Divide the workload on parallel threads
- Sustain the performance growth

Introduction: parallelization

Exponential increase in processors speed has reached its technological limits

- Increase the parallelization
- Divide the workload on parallel threads
- Sustain the performance growth
- Concurrent algorithms are already a reality, e.g. the java.util.concurrent

Introduction:price

Parallelization comes at a price:

- Code extremely difficult to get right and to debug

Introduction:price

Parallelization comes at a price:

- Code extremely difficult to get right and to debug
- Correctness need to cover all possible interleavings, and all possible numbers of threads

Introduction:price

Parallelization comes at a price:

- Code extremely difficult to get right and to debug
- Correctness need to cover all possible interleavings, and all possible numbers of threads
- Requires expertise and experience

Introduction:price

Parallelization comes at a price:

- Code extremely difficult to get right and to debug
- Correctness need to cover all possible interleavings, and all possible numbers of threads
- Requires expertise and experience
- Built on intuition, often with hand-waving correctness arguments

Introduction:price

Parallelization comes at a price:

- Code extremely difficult to get right and to debug
- Correctness need to cover all possible interleavings, and all possible numbers of threads
- Requires expertise and experience
- Built on intuition, often with hand-waving correctness arguments

Goal: Develop automatic tools that analyze source code and check correctness for all possible interleavings and all possible numbers of threads

Verification Approach

Tools are to implement verification techniques:

Verification Approach

Tools are to implement verification techniques:

- Testing and simulation
  are good for finding
  errors

Verification Approach

Tools are to implement verification techniques:

- Testing and simulation are good for finding errors
- Model checking is good for showing correctness and for debugging

Verification Approach

Tools are to implement verification techniques:

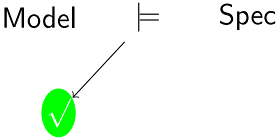
- Testing and simulation are good for finding errors
- Model checking is good for showing correctness and for debugging

$$\text{Model} \quad \models \quad \text{Spec}$$

Verification Approach

Tools are to implement verification techniques:

- Testing and simulation are good for finding errors
- Model checking is good for showing correctness and for debugging



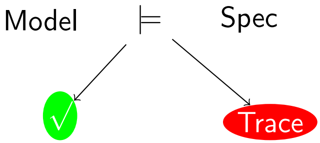
Model $\models$ Spec

✓

Verification Approach

Tools are to implement verification techniques:

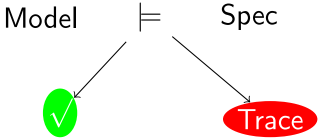
- Testing and simulation are good for finding errors
- Model checking is good for showing correctness and for debugging



Verification Approach

Tools are to implement verification techniques:

- Testing and simulation are good for finding errors
- Model checking is good for showing correctness and for debugging



Goal: Extend Model Checking to analyze concurrent algorithms

Outline

Introduction

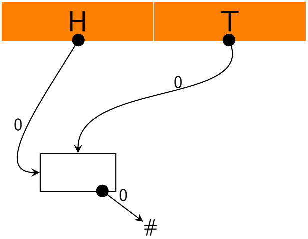
Concurrent Algorithms

Symbolic Representation and Abstraction for Verification

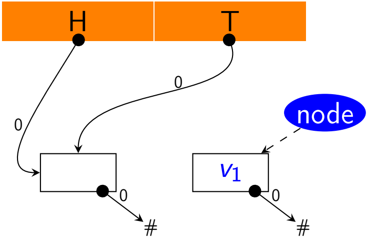
Counter Example Guided Abstraction Refinement

Conclusion

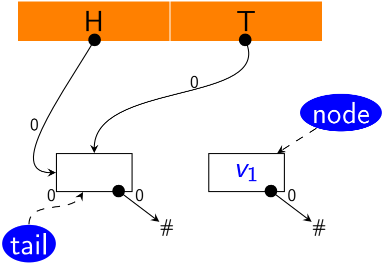
Concurrent Algorithms: A non-bloquing queue



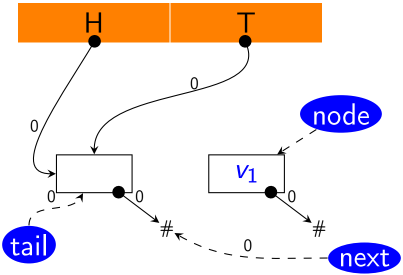
Concurrent Algorithms: A non-bloquing queue



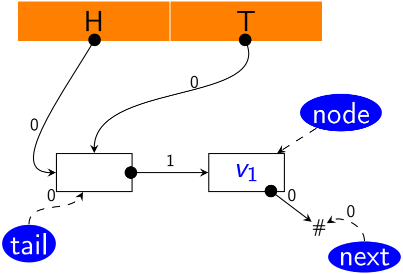
Concurrent Algorithms: A non-bloquing queue



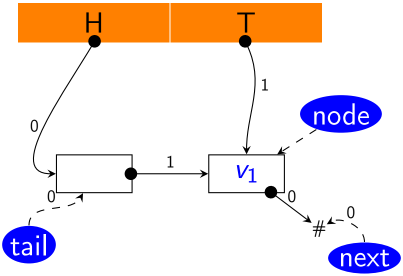
Concurrent Algorithms: A non-bloquing queue



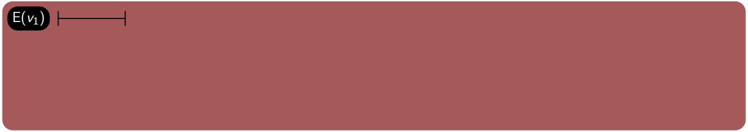
Concurrent Algorithms: A non-bloquing queue

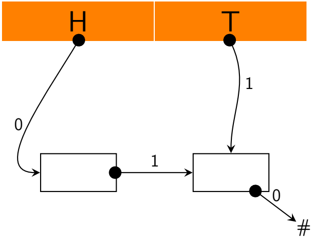


Concurrent Algorithms: A non-bloquing queue

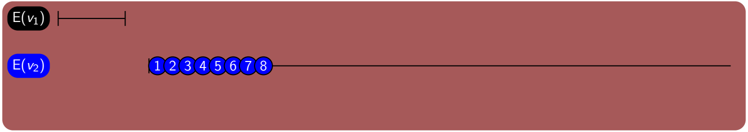


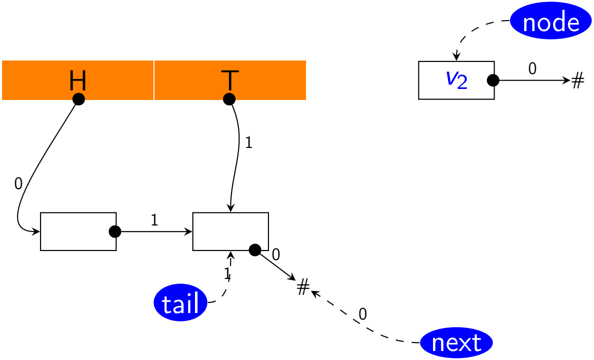
Concurrent Algorithms: A non-bloquing queue



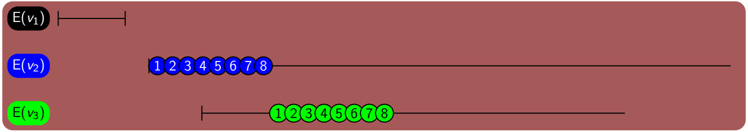


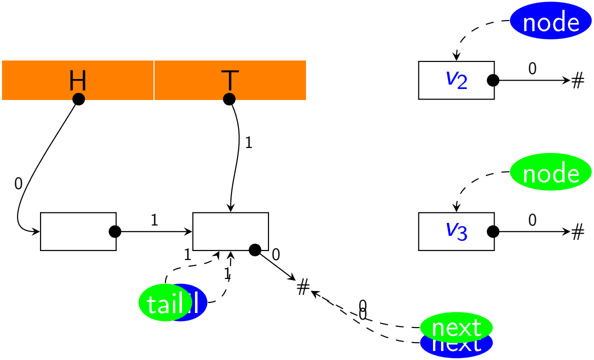
Concurrent Algorithms: A non-bloquing queue



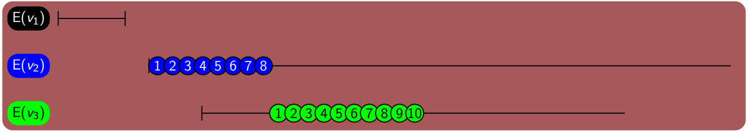


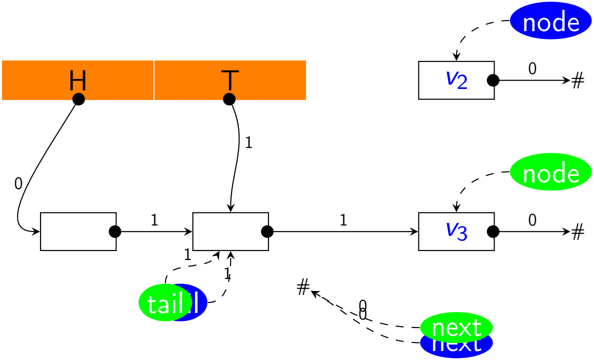
Concurrent Algorithms: A non-bloquing queue



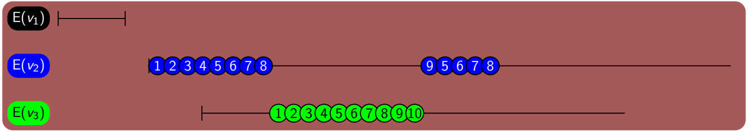


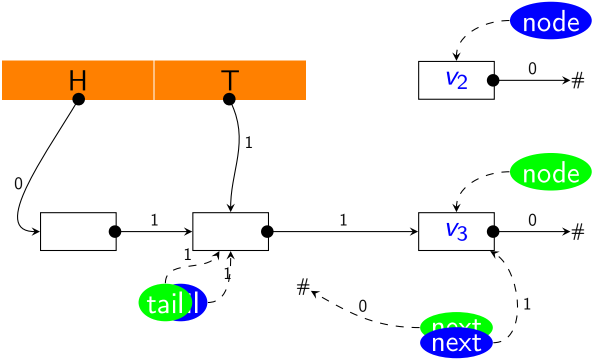
Concurrent Algorithms: A non-bloquing queue



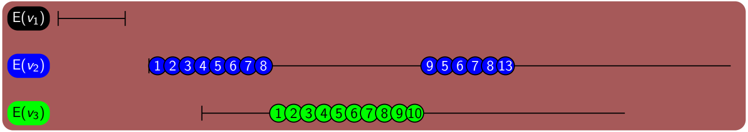


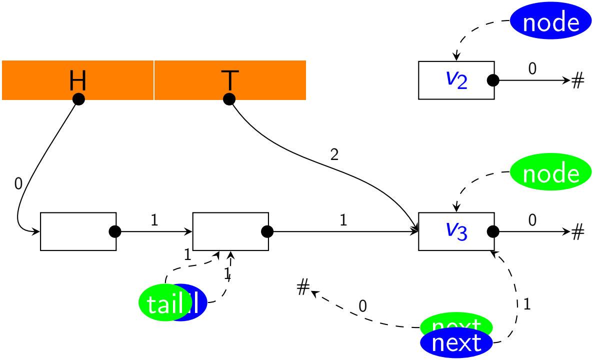
Concurrent Algorithms: A non-bloquing queue



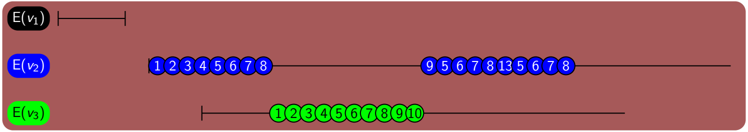


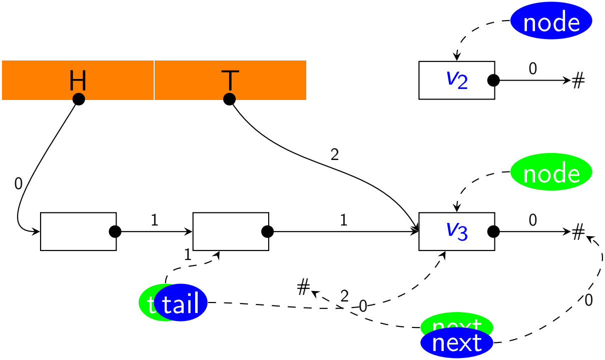
Concurrent Algorithms: A non-bloquing queue



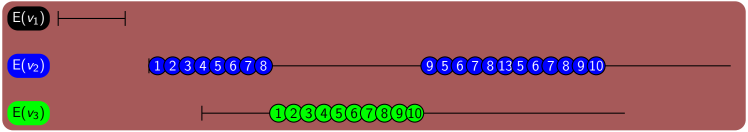


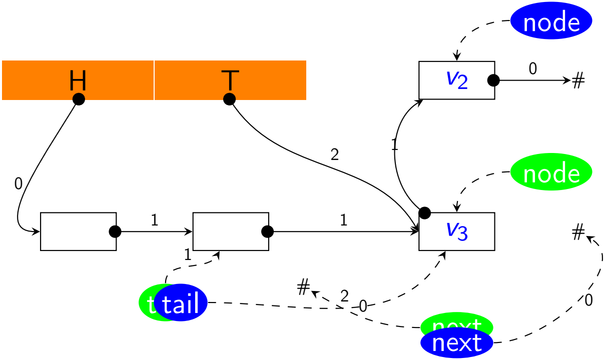
Concurrent Algorithms: A non-bloquing queue



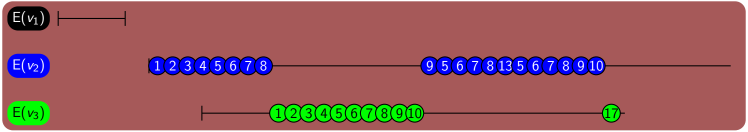


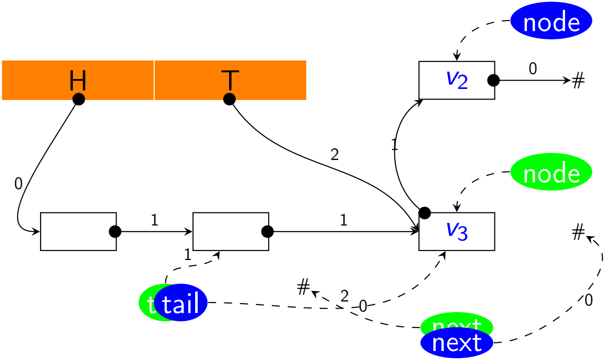
Concurrent Algorithms: A non-bloquing queue



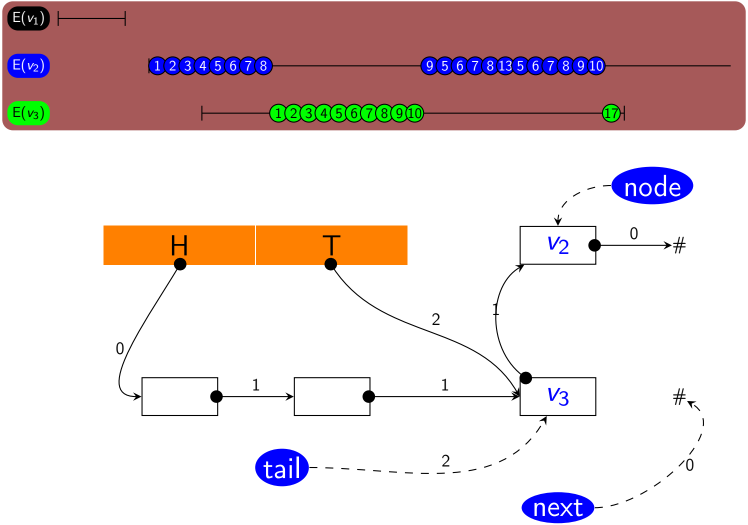


Concurrent Algorithms: A non-bloquing queue

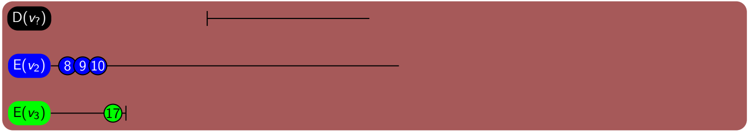


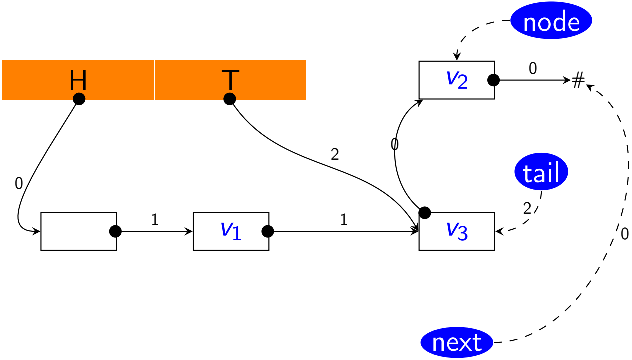


Concurrent Algorithms: A non-bloquing queue

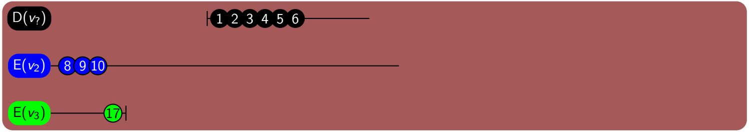


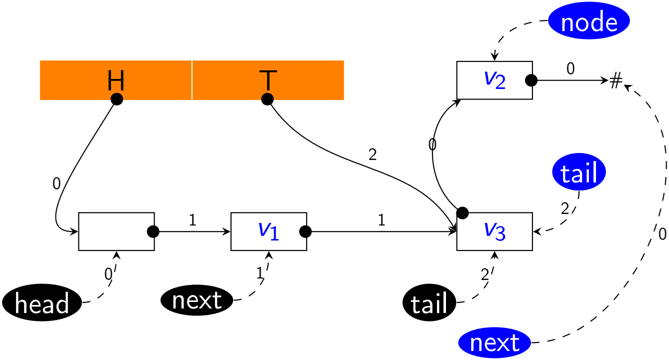
Concurrent Algorithms: a non-bloquing queue



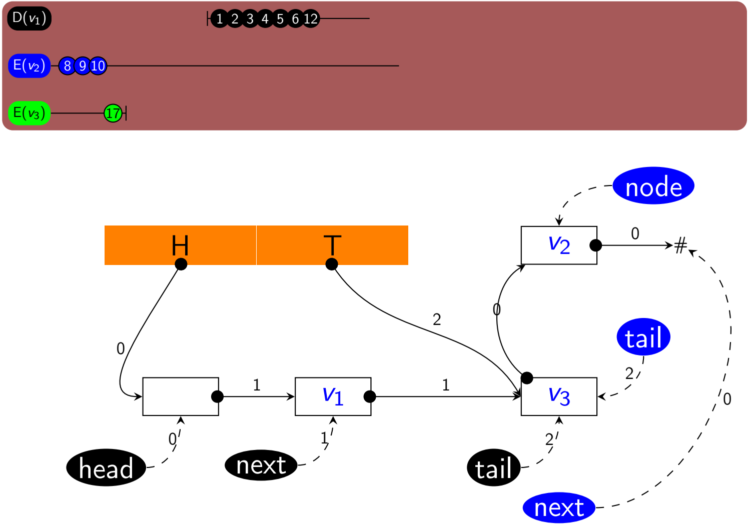


Concurrent Algorithms: a non-bloquing queue

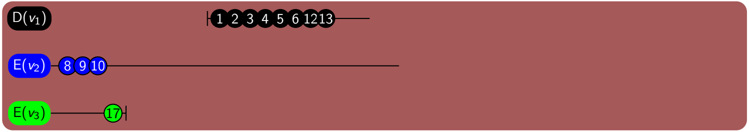


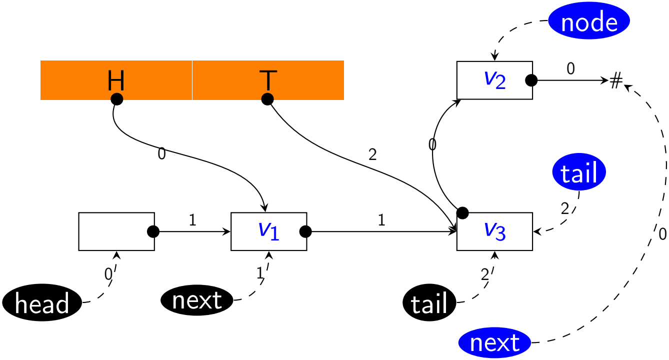


Concurrent Algorithms: a non-bloquing queue

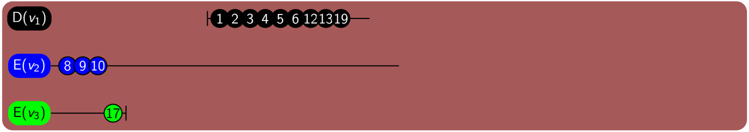


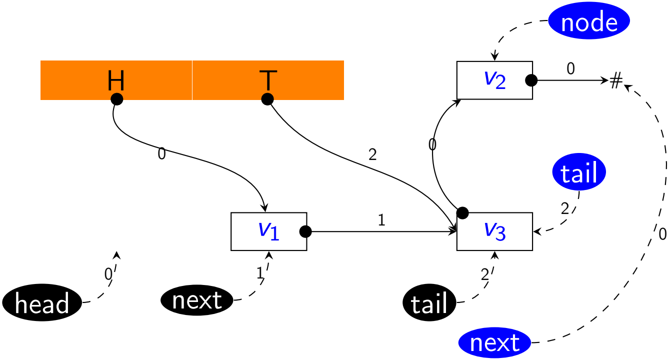
Concurrent Algorithms: a non-bloquing queue



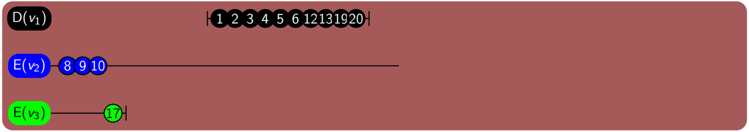


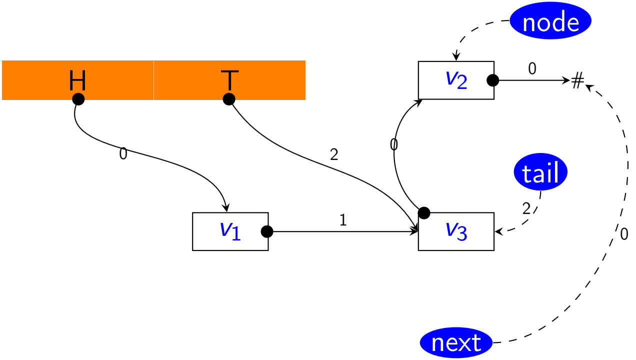
Concurrent Algorithms: a non-bloquing queue



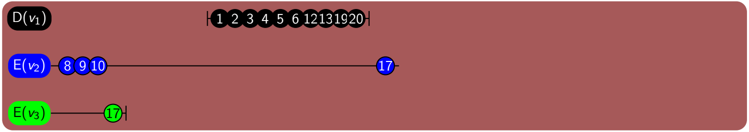


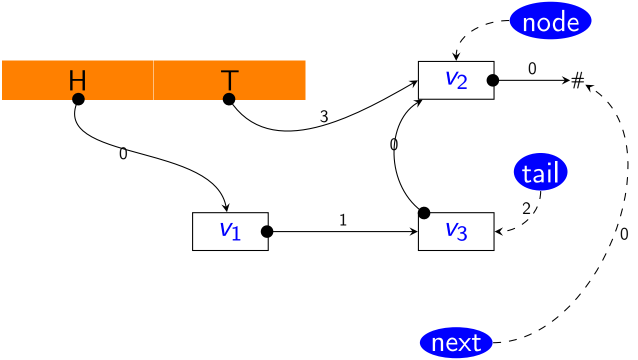
Concurrent Algorithms: a non-bloquing queue



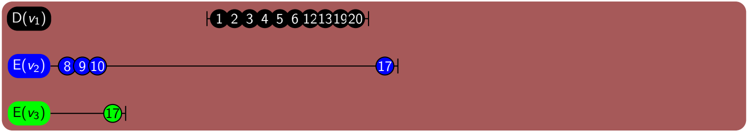


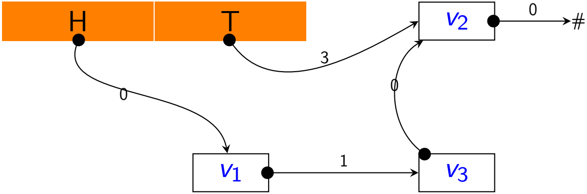
Concurrent Algorithms: a non-bloquing queue





Concurrent Algorithms: a non-bloquing queue





Concurrent Algorithms

Analysis needs to take into account several sources of difficulty:

- Arbitrary numbers of threads
- Infinite data domains
- Dynamic memory
- Memory model guaranteed by the machine
- ...

Outline

Introduction

Concurrent Algorithms

Symbolic Representation and Abstraction for Verification

Counter Example Guided Abstraction Refinement

Conclusion

Symbolic Representations for Infinite Sets of States

Symbolic Representations for Infinite Sets of States

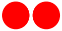
- Safety properties'
  Violations representable
  by finite traces

Symbolic Representations for Infinite Sets of States

- Safety properties' Violations representable by finite traces
- Use a finite state observer that synchronizes with the system

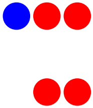
Symbolic Representations for Infinite Sets of States

- Safety properties' Violations representable by finite traces
- Use a finite state observer that synchronizes with the system



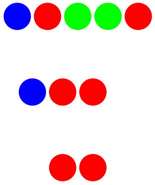
Symbolic Representations for Infinite Sets of States

- Safety properties' Violations representable by finite traces
- Use a finite state observer that synchronizes with the system
- Adding more threads still violates the safety property



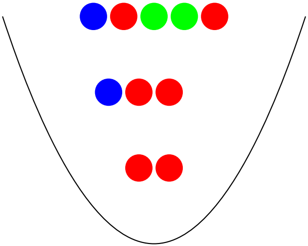
Symbolic Representations for Infinite Sets of States

- Safety properties' Violations representable by finite traces
- Use a finite state observer that synchronizes with the system
- Adding more threads still violates the safety property



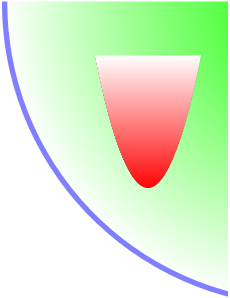
Symbolic Representations for Infinite Sets of States

- Safety properties' Violations representable by finite traces
- Use a finite state observer that synchronizes with the system
- Adding more threads still violates the safety property

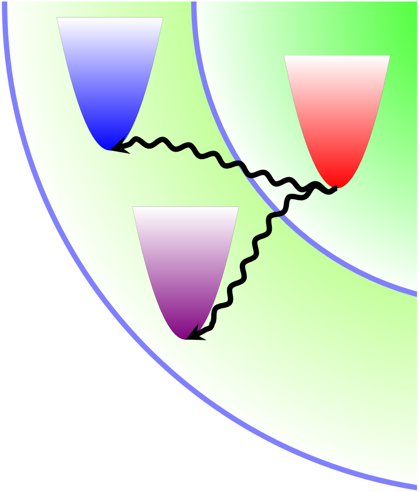


Use upward closed sets as symbolic representations for states violating safety properties

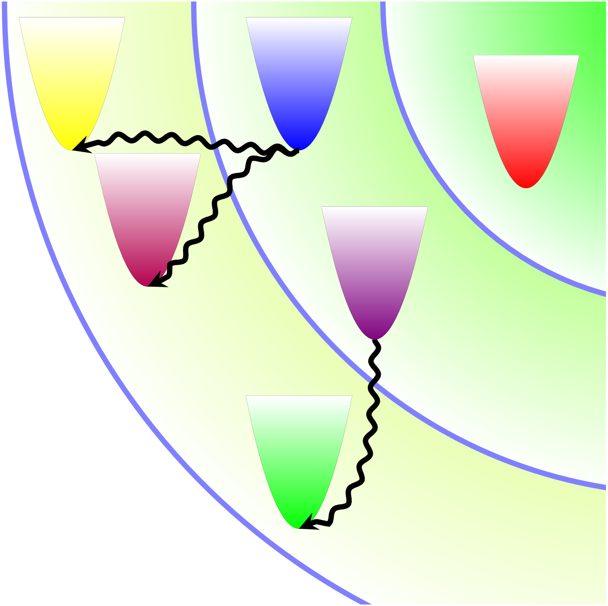
Symbolic Representation and Backward Analysis



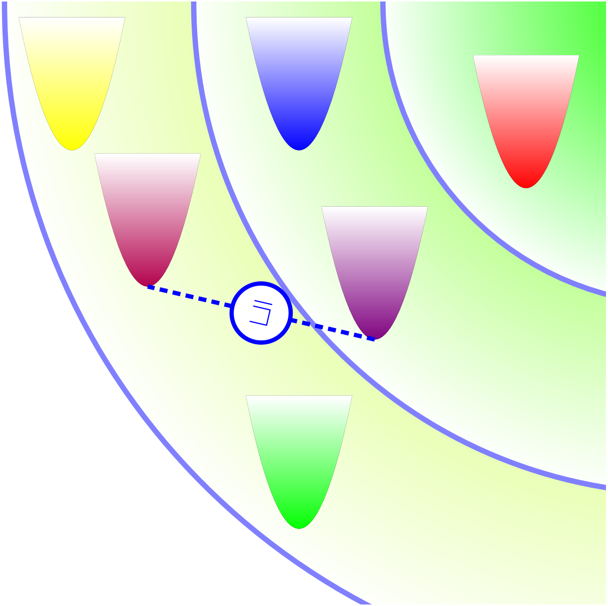
Symbolic Representation and Backward Analysis



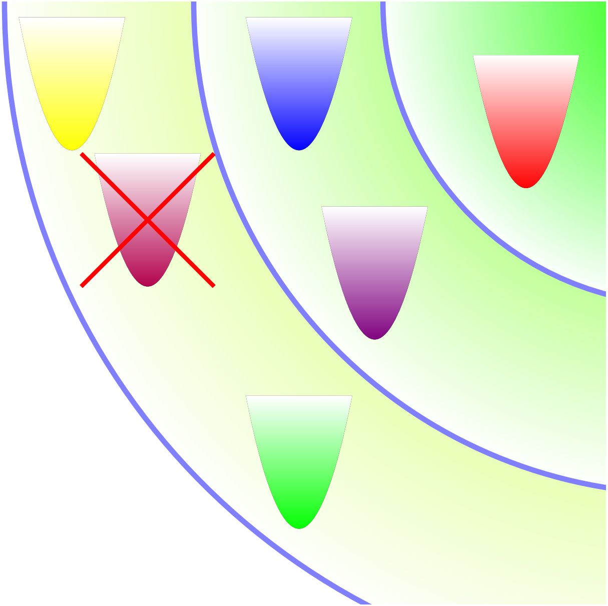
Symbolic Representation and Backward Analysis



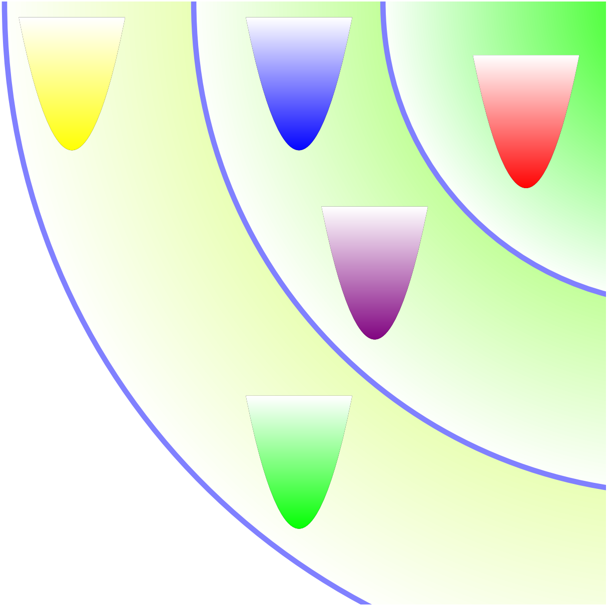
Symbolic Representation and Backward Analysis



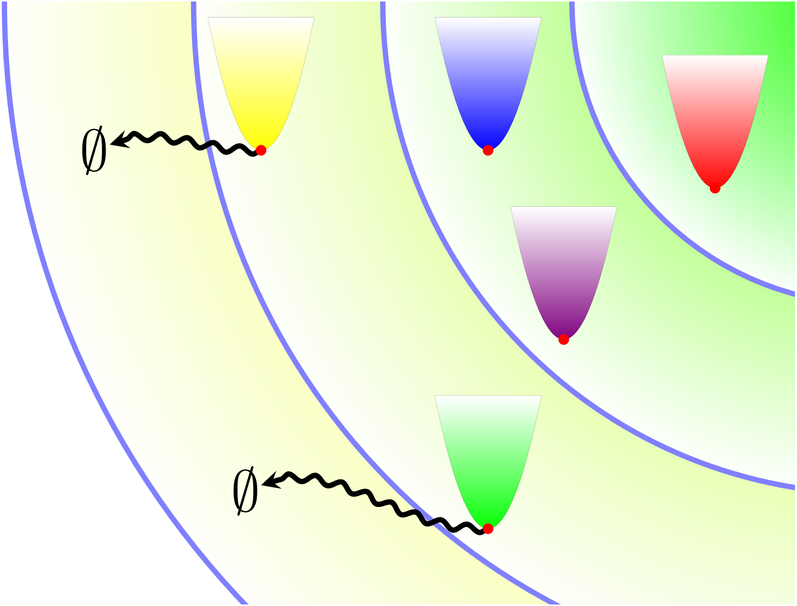
Symbolic Representation and Backward Analysis



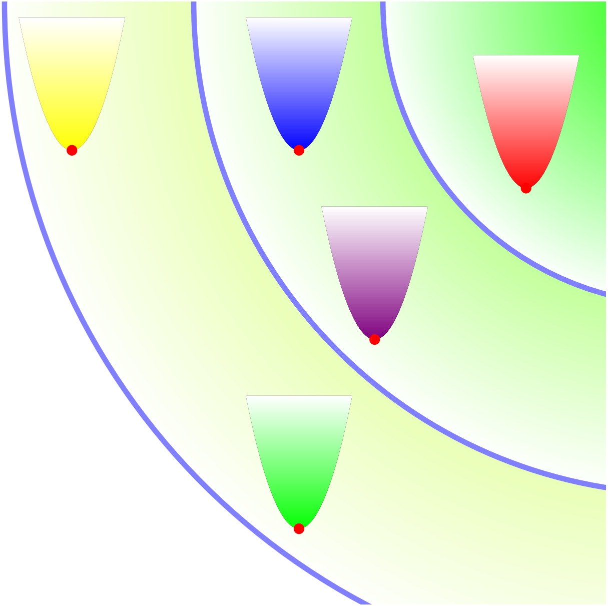
Symbolic Representation and Backward Analysis



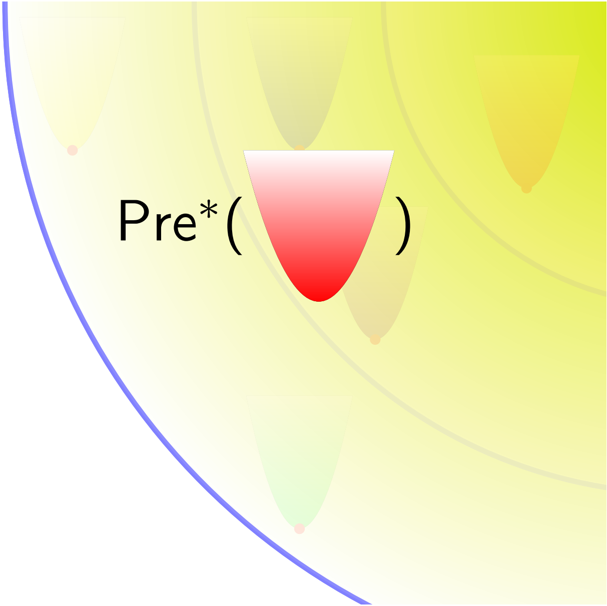
Symbolic Representation and Backward Analysis



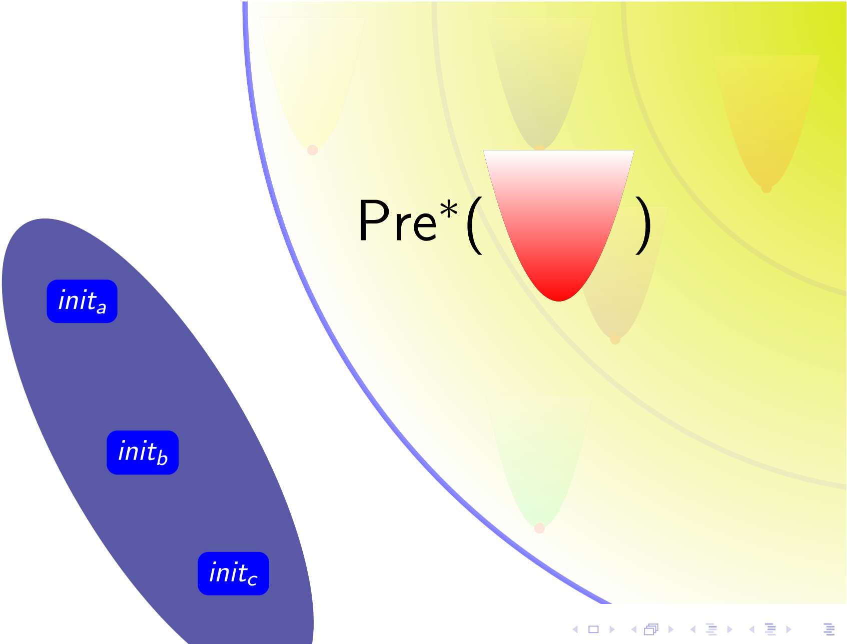
Symbolic Representation and Backward Analysis



Symbolic Representation and Backward Analysis

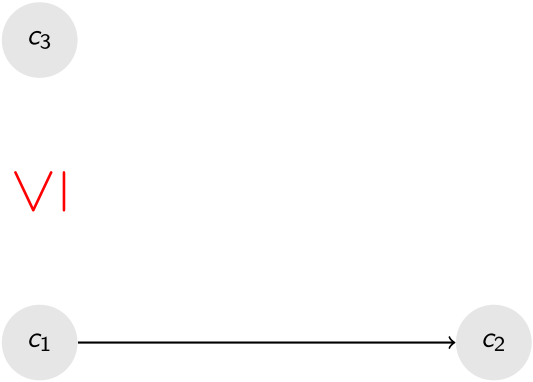


Symbolic Representation and Backward Analysis



Abstraction to Permit Symbolic Analysis

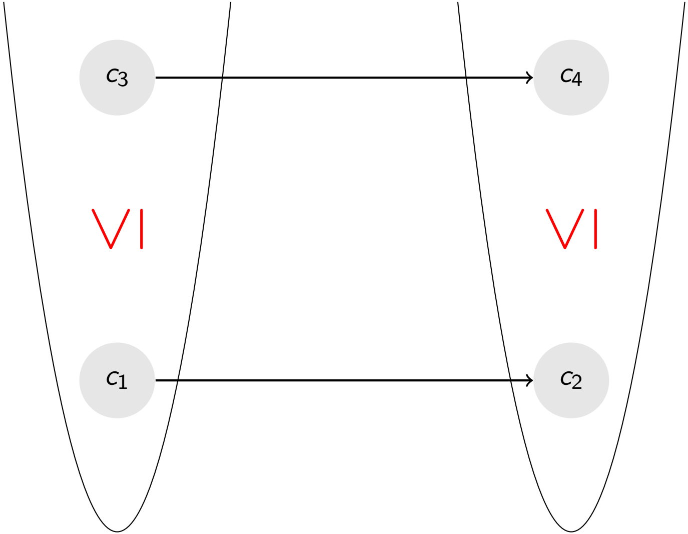
Abstraction to Permit Symbolic Analysis



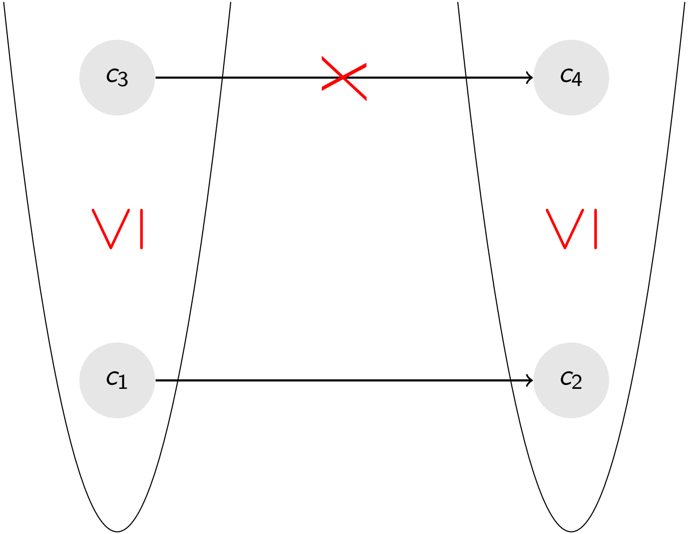
Abstraction to Permit Symbolic Analysis

$$\begin{tikzcd}
c_3 \arrow[r] & c_4 \\
c_1 \arrow[r] \arrow[u, phantom, sloped, "\leq"] & c_2 \arrow[u, phantom, sloped, "\leq"]
\end{tikzcd}$$

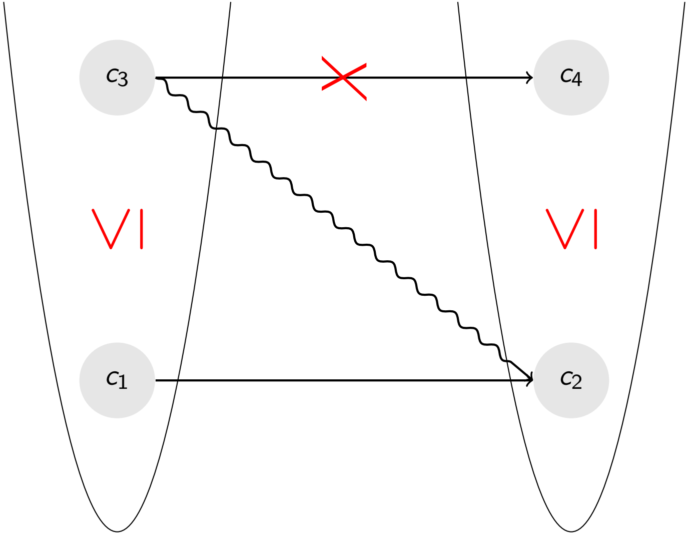
Abstraction to Permit Symbolic Analysis



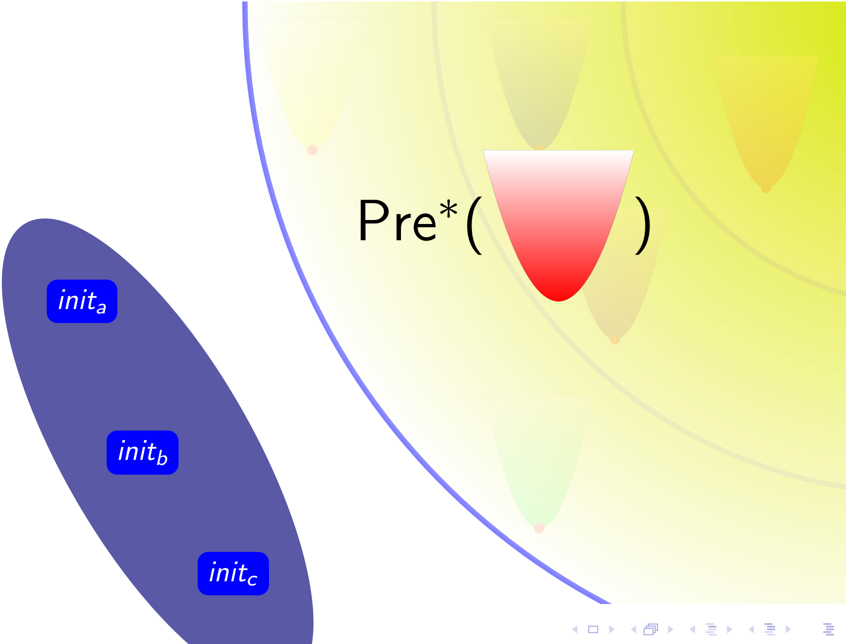
Abstraction to Permit Symbolic Analysis



Abstraction to Permit Symbolic Analysis



Abstraction and Backward Analysis



Outline

Introduction

Concurrent Algorithms

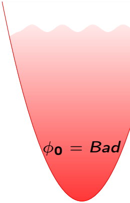
Symbolic Representation and Abstraction for Verification

Counter Example Guided Abstraction Refinement

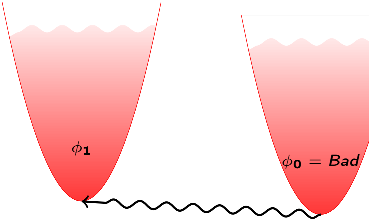
Conclusion

Counter Example Guided Abstraction Refinement

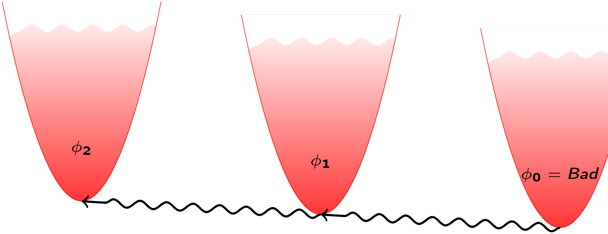
Counter Example Guided Abstraction Refinement



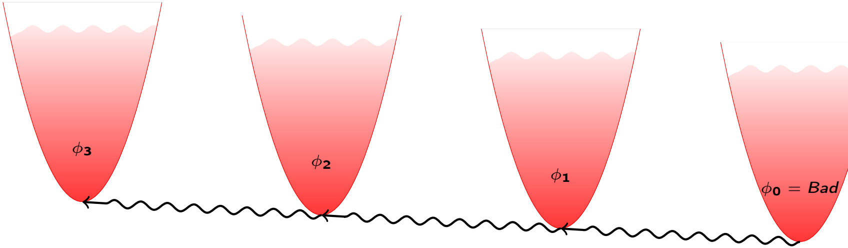
Counter Example Guided Abstraction Refinement



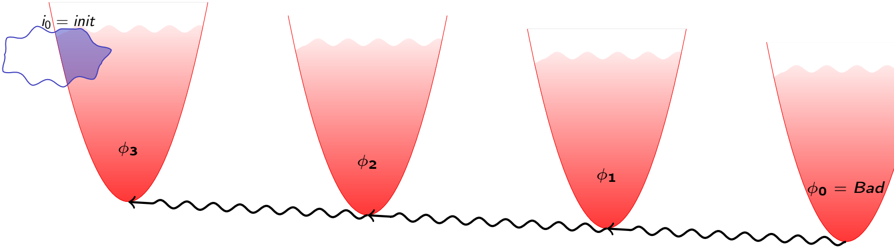
Counter Example Guided Abstraction Refinement



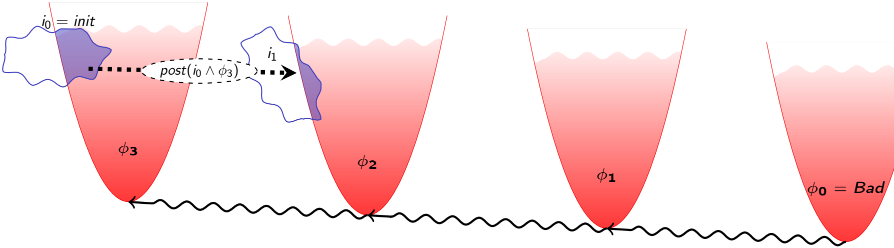
Counter Example Guided Abstraction Refinement



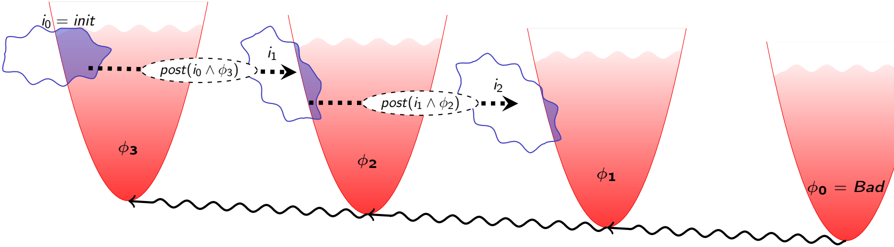
Counter Example Guided Abstraction Refinement



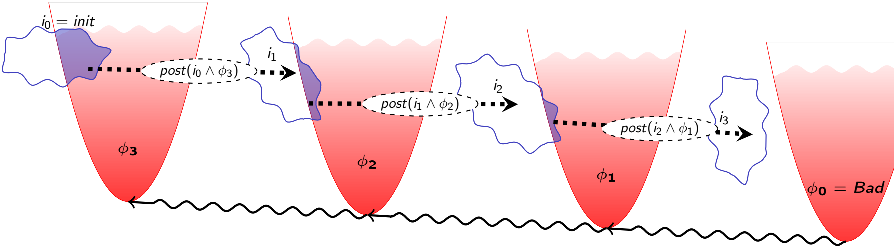
Counter Example Guided Abstraction Refinement



Counter Example Guided Abstraction Refinement

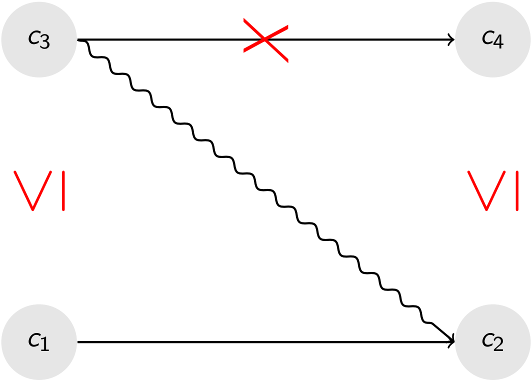


Counter Example Guided Abstraction Refinement

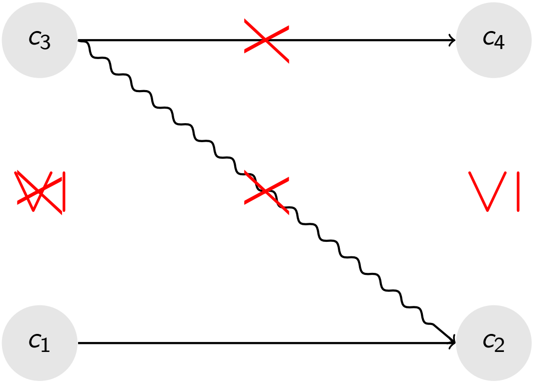

$i_0 = init$
$post(i_0 \wedge \phi_3)$
i_1
ϕ_3
$post(i_1 \wedge \phi_2)$
i_2
ϕ_2
$post(i_2 \wedge \phi_1)$
i_3
ϕ_1
$\phi_0 = Bad$

Counter Example Guided Abstraction Refinement

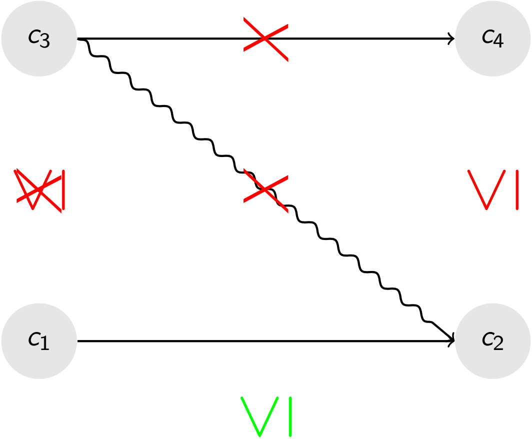
Counter Example Guided Abstraction Refinement



Counter Example Guided Abstraction Refinement



Counter Example Guided Abstraction Refinement



Outline

Introduction

Concurrent Algorithms

Symbolic Representation and Abstraction for Verification

Counter Example Guided Abstraction Refinement

Conclusion

Conclusion and Future Work

- Parallelization makes heavy use of concurrent algorithms

Conclusion and Future Work

- Parallelization makes heavy use of concurrent algorithms
- We build automatic symbolic techniques to increase confidence in concurrent algorithms

Conclusion and Future Work

- Parallelization makes heavy use of concurrent algorithms
- We build automatic symbolic techniques to increase confidence in concurrent algorithms
- Our techniques explore all interleavings for arbitrary numbers of threads

Conclusion and Future Work

- Parallelization makes heavy use of concurrent algorithms
- We build automatic symbolic techniques to increase confidence in concurrent algorithms
- Our techniques explore all interleavings for arbitrary numbers of threads
- We obtained successful results and are working on the verification of code found in widespread software, like the java.util.concurrent package