Verifying Concurrent Programs via Bounded Context-Switching and Induction

Prathmesh Prabhu [1]
Akash Lal [2]
Nicholas Kidd [3]
Thomas Reps [4]

September 27, 2011

[1]University of Wisconsin; Madison, WI USA
[2]Microsoft Research India; Bangalore, India
[3]Google Madison; WI USA
[4]University of Wisconsin; Madison, WI USA & GrammaTech Inc.; Ithaca, NY USA

Contents

Verifying Programs

Contents

Contents

Induction

Contents

At the end of this talk, I will know a way of...

Verifying Concurrent Programs via Bounded Context-Switching and Induction

At the end of this talk, I will know a way of...

Verifying Programs

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]
[p8]     if(a > b) a = 1/0;
[p9] }
```

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]
[p8]     if(a > b) a = 1/0;
[p9] }
```

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Assertions specify *Properties*

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Assertions specify *Properties*

Challenge!

Infinite / large type domains

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Assertions specify *Properties*

Challenge!

Infinite / large type domains

[a=1,b=0,pc=*p4*]

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Assertions specify *Properties*

Challenge!

Infinite / large type domains

[a=1,b=0,pc=*p4*]

[1,0,*p5*]

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Assertions specify *Properties*

Challenge!

Infinite / large type domains

[a=1,b=0,pc=*p4*]

[1,0,*p5*]

[2,0,*p6*]

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Assertions specify *Properties*

Challenge!

Infinite / large type domains

[a=1,b=0,pc=*p4*]

[1,0,*p5*]

[2,0,*p6*]

[2,0,*p4*]

Background: Software Verification

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Assertions specify *Properties*

Challenge!

Infinite / large type domains

[a=1,b=0,pc=*p4*]

[1,0,*p5*]

[2,0,*p6*]

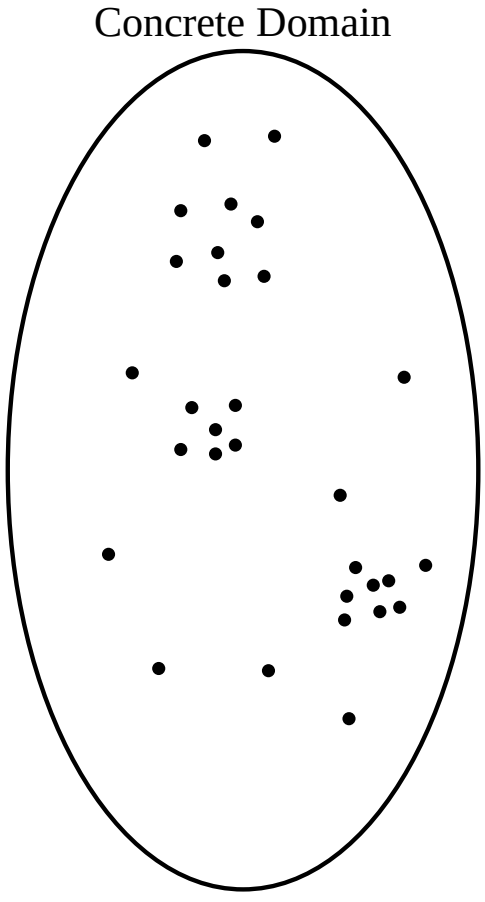
[2,0,*p4*]

[2,0,*p5*]

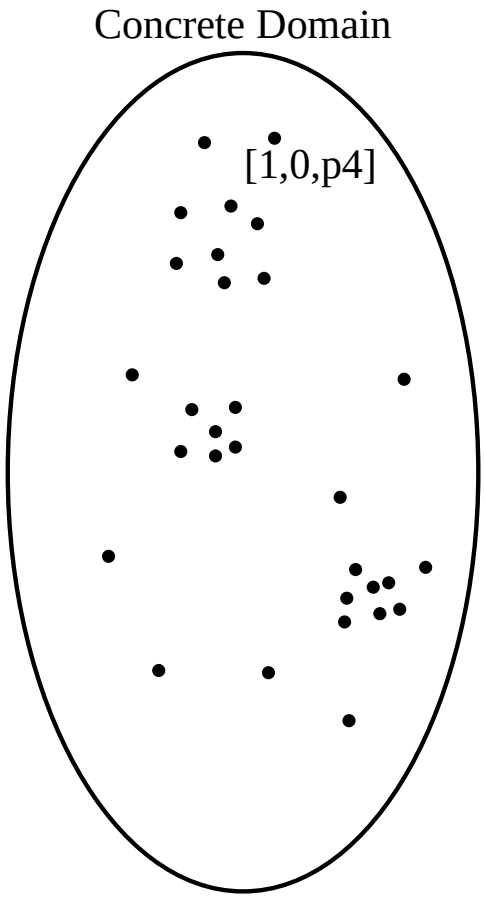
[3,0,*p6*]

...

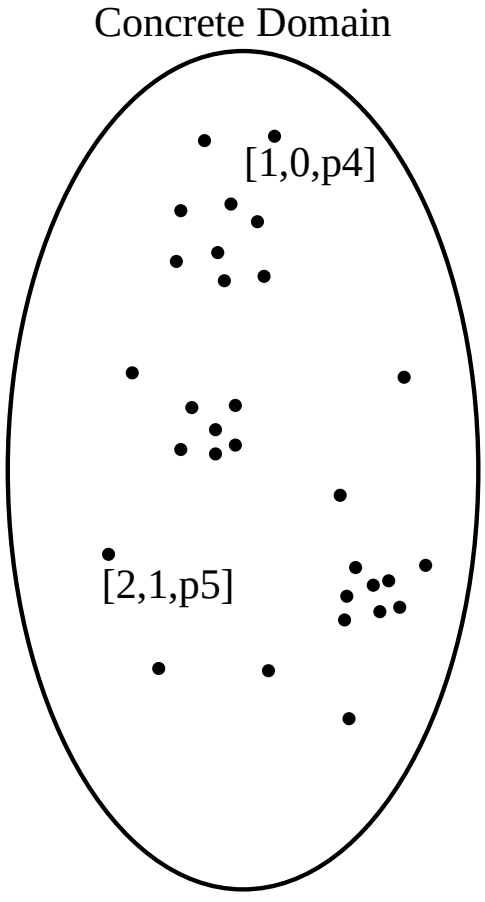
Background: Abstract Interpretation



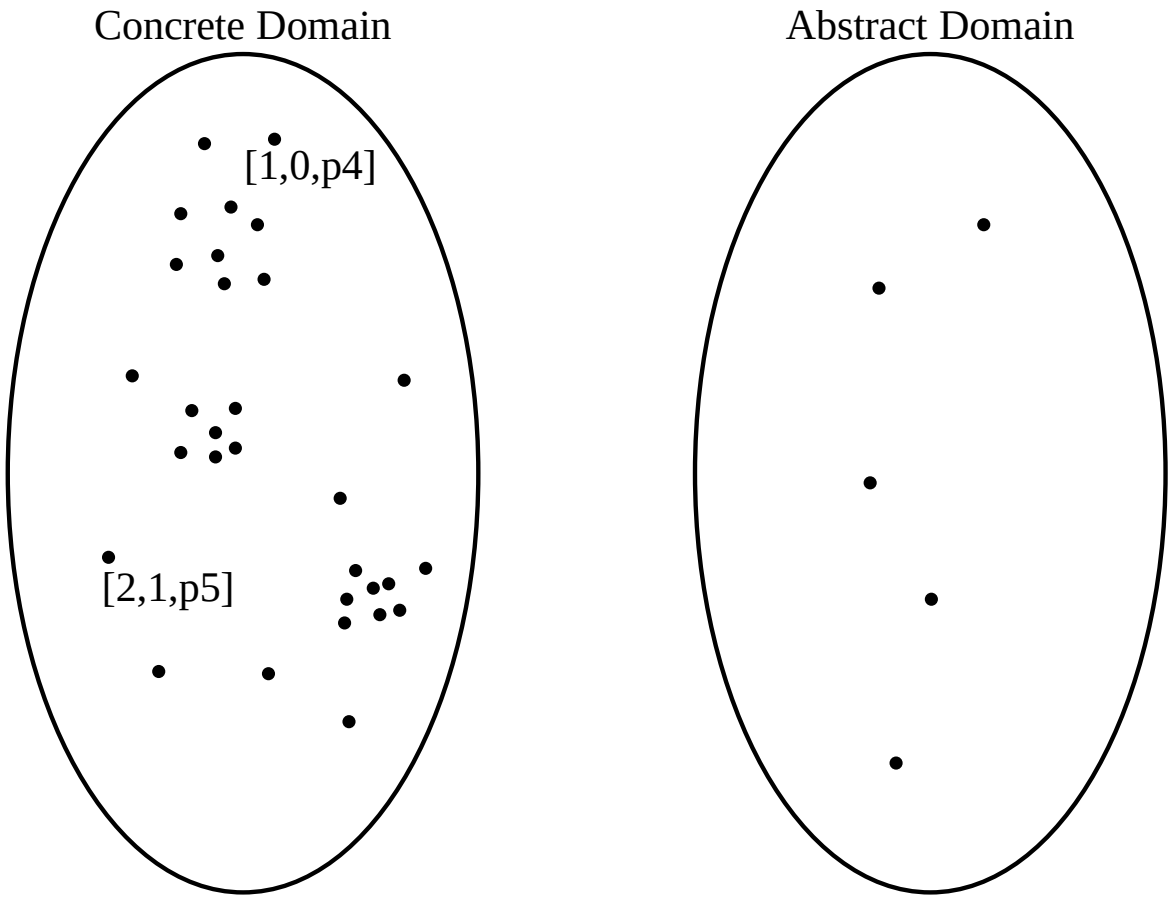
Background: Abstract Interpretation



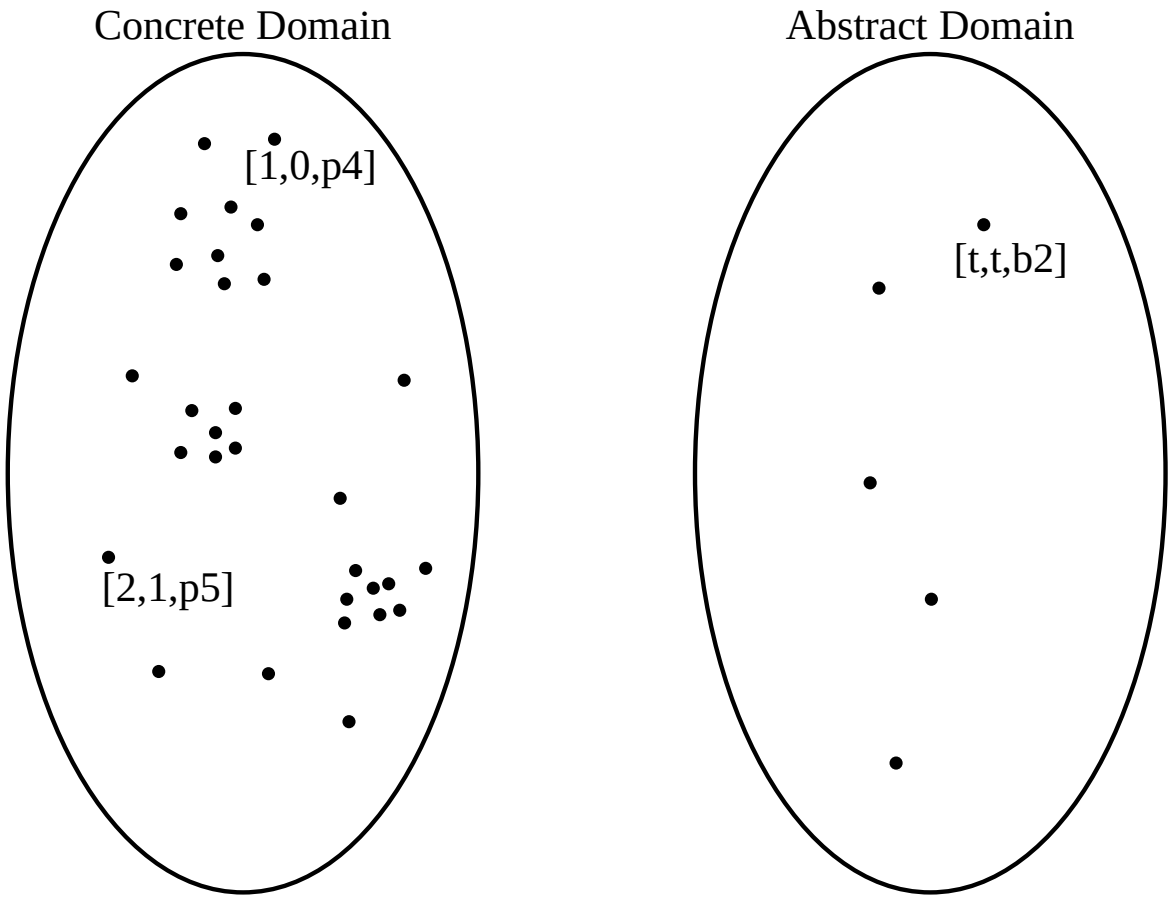
Background: Abstract Interpretation



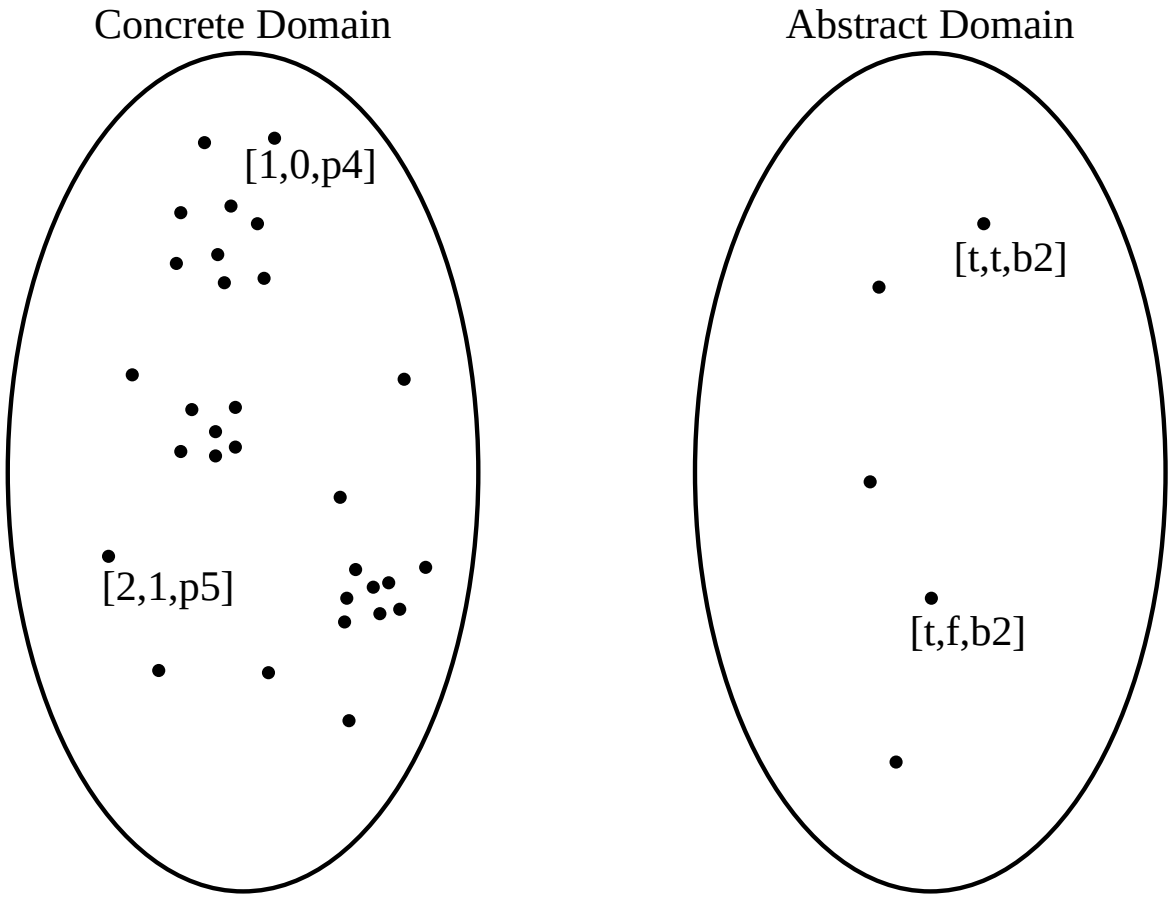
Background: Abstract Interpretation



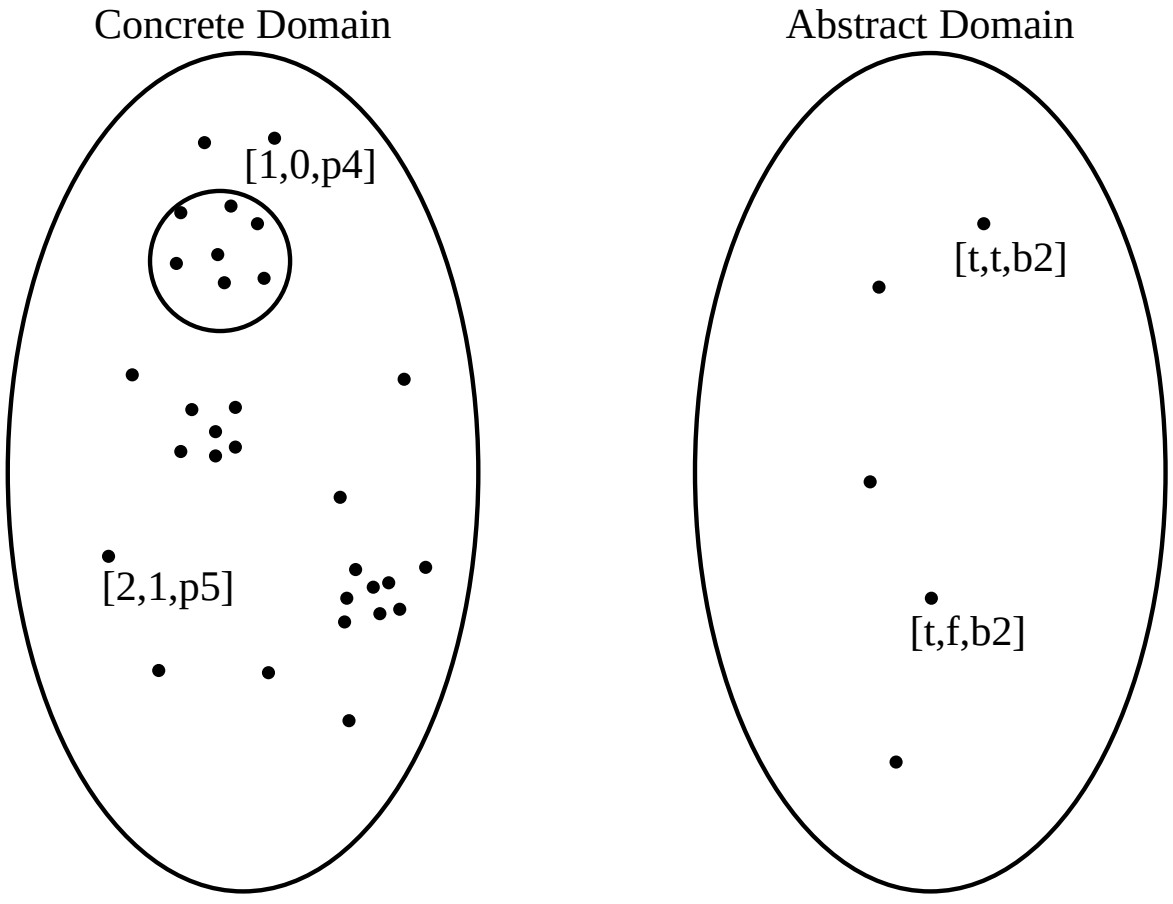
Background: Abstract Interpretation



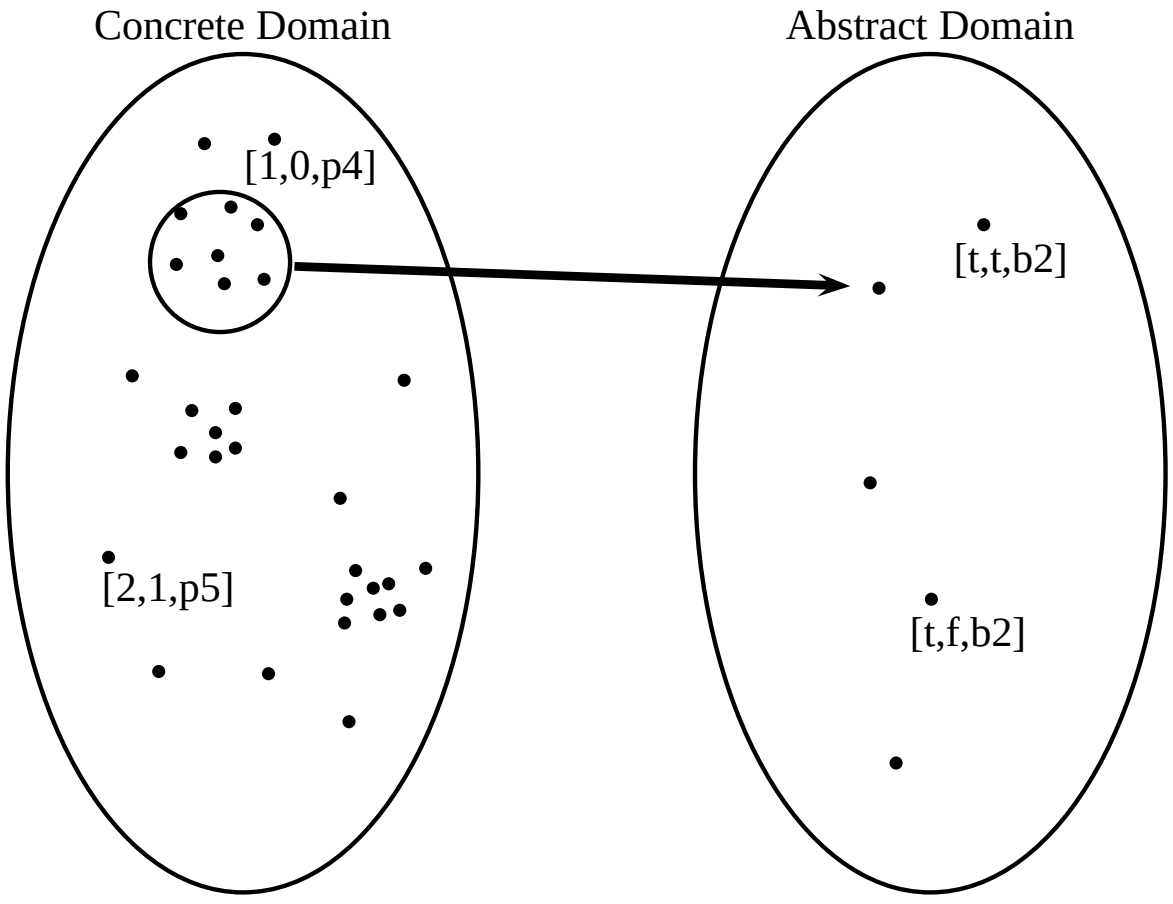
Background: Abstract Interpretation



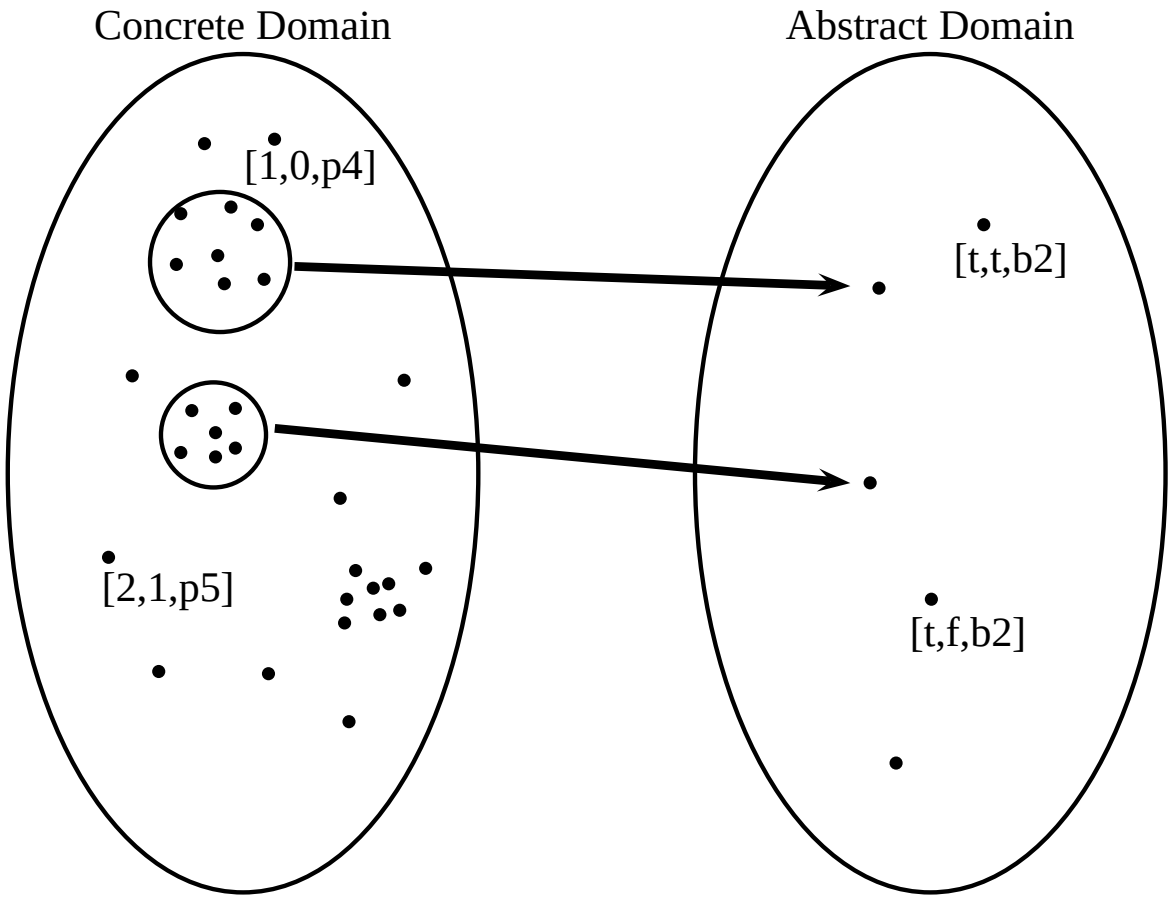
Background: Abstract Interpretation



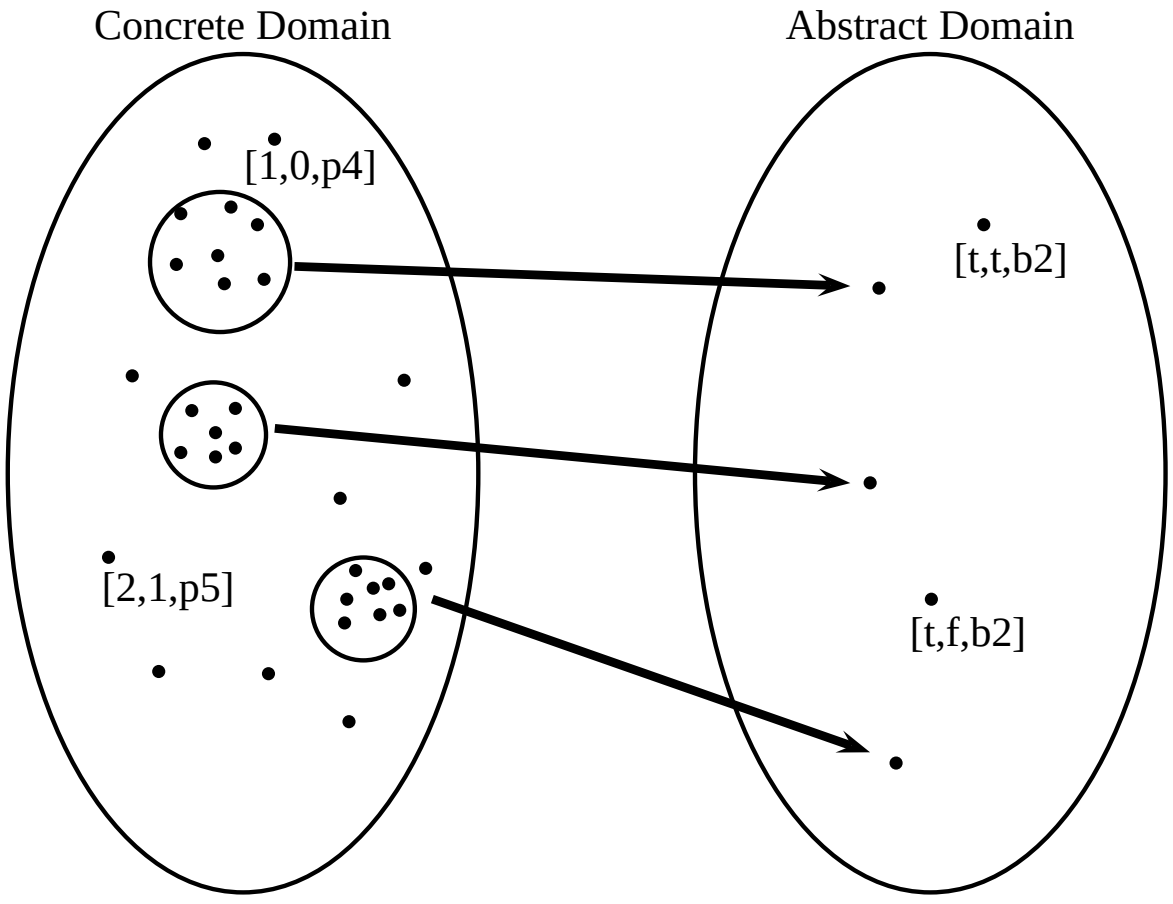
Background: Abstract Interpretation



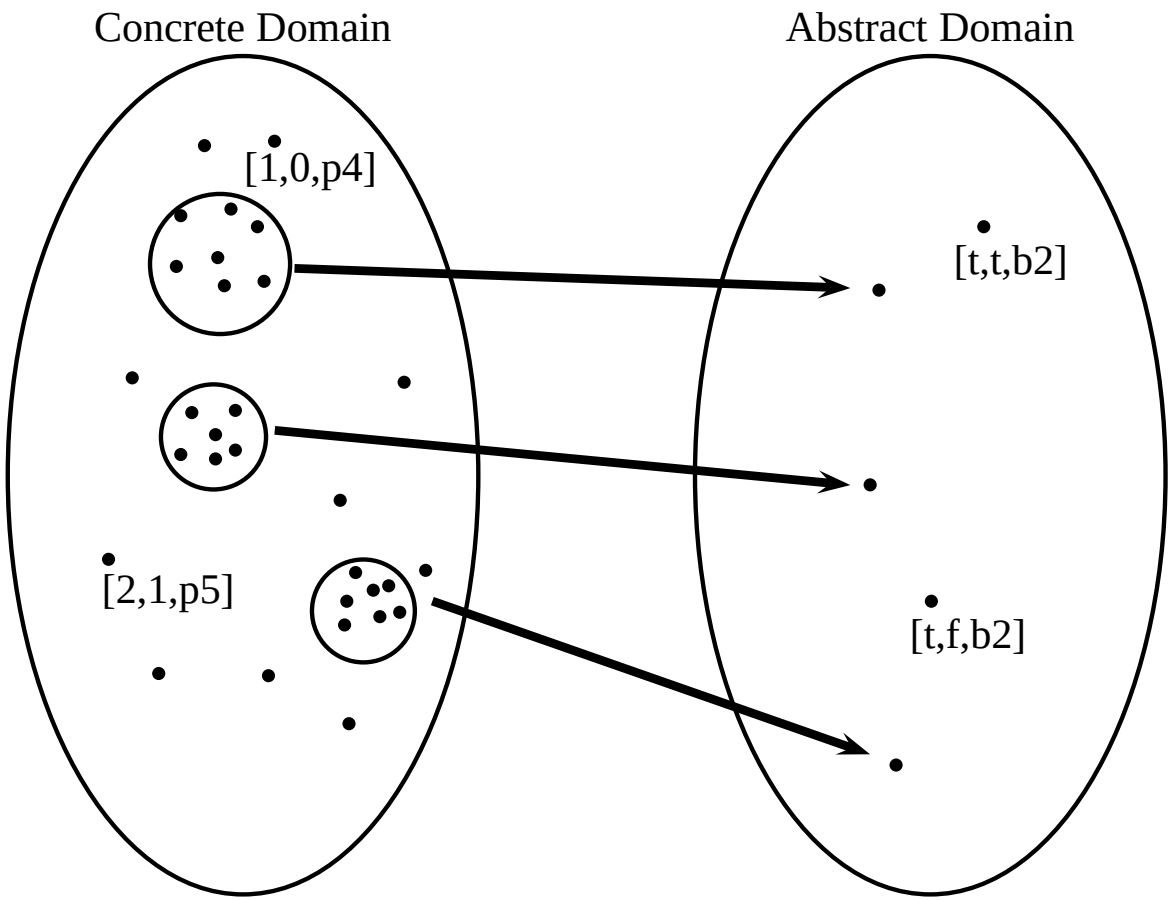
Background: Abstract Interpretation



Background: Abstract Interpretation



Background: Abstract Interpretation



Predicate Abstraction

Background: Abstract Interpretation - Boolean Programs

Background: Abstract Interpretation - Boolean Programs

Only Boolean Variables

a := *true*
| *false*
| b
| ! b
| b $\vee$ c
| b $\wedge$ c
| *

assume(expr)
if(*) {}
do(*) {}
label:
skip

Background: Abstract Interpretation - Boolean Programs

Only Boolean Variables

a := *true*
| *false*
| b
| ! b
| b $\vee$ c
| b $\wedge$ c
| *

assume(expr)
if(*) {}
do(*) {}
label:
skip

Background: Abstract Interpretation - Boolean Programs

Only Boolean Variables

a := *true*
| *false*
| b
| ! b
| b $\vee$ c
| b $\wedge$ c
| *

assume(expr)
if(*) {}
do(*) {}
label:
skip

States of the transition system

$\mathbb{S} = \{(v, e) | v \in \mathbb{V}ar \wedge e \in \{t, f\}\} \times pc$

e.g.: $[t, t, b1]$

Background: Abstract Interpretation - Boolean Programs

Only Boolean Variables

a := *true*
| *false*
| b
| ! b
| b ∨ c
| b ∧ c
| *

assume(expr)
if(*) {}
do(*) {}
label:
skip

States of the transition system

$\mathbb{S} = \{(v, e) | v \in \mathbb{V}ar \wedge e \in \{t, f\}\} \times pc$

e.g.: $[t, t, b1]$

$\Delta[a := true](pc, \overline{v}) = (pc', \overline{v}[a \mapsto t])$

$\Delta[a := *](pc, \overline{v}) = (pc', \overline{v}[a \mapsto t]) \cup (pc', \overline{v}[a \mapsto f])$

$\Delta[if(*)\{pc_{in}\}pc_{out}](pc, \overline{v}) = (pc_{in}, \overline{v}) \cup (pc_{out}, \overline{v})$

$\Delta[assume(\text{expr})](pc, \overline{v}) = \text{if } [\text{expr}](\overline{v}) = true \text{ then } (pc', \overline{v}) \text{ else } \phi$

Background: Boolean Programs - example

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Background: Boolean Programs - example

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Predicate

$X = a > b$

Background: Boolean Programs - example

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a + +;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Predicate

$X = a > b$

```
[b1] main(){
[b2]     bool X;
[b3]     X := *;
[b4]     do(*){
[b5]         assume(X);
[b6]         if(*){
[b7]           assume(X);
[b8]           X :=true;
[b9]         }
[b10]    }
[b11]    assume(!X)
[b12]    if(*){
[b13]       assume(X);
[b14]       error: skip;
[b15]    }
[b16] }
```

Background: Boolean Programs - example

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a + +;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Predicate

$X = a > b$

```
[b1] main(){
[b2]     bool X;
[b3]     X := *;
[b4]     do(*){
[b5]         assume(X);
[b6]         if(*){
[b7]           assume(X);
[b8]           X :=true;
[b9]         }
[b10]    }
[b11]    assume(!X)
[b12]    if(*){
[b13]       assume(X);
[b14]       error: skip;
[b15]    }
[b16] }
```

Background: Boolean Programs - example

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Predicate

$X = a > b$

```
[b1] main(){
[b2]     bool X;
[b3]     X := *;
[b4]     do(*){
[b5]         assume(X);
[b6]         if(*){
[b7]           assume(X);
[b8]           X :=true;
[b9]         }
[b10]    }
[b11]    assume(!X)
[b12]    if(*){
[b13]       assume(X);
[b14]       error: skip;
[b15]    }
[b16] }
```

Background: Boolean Programs - example

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a + +;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Predicate

$X = a > b$

```
[b1] main(){
[b2]     bool X;
[b3]     X := *;
[b4]     do(*){
[b5]         assume(X);
[b6]         if(*){
[b7]           assume(X);
[b8]           X :=true;
[b9]         }
[b10]    }
[b11]    assume(!X)
[b12]    if(*){
[b13]       assume(X);
[b14]       error: skip;
[b15]    }
[b16] }
```

Background: Boolean Programs - example

```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Predicate

$X = a > b$

```
[b1] main(){
[b2]     bool X;
[b3]     X := *;
[b4]     do(*){
[b5]         assume(X);
[b6]         if(*){
[b7]           assume(X);
[b8]           X :=true;
[b9]         }
[b10]    }
[b11]    assume(!X)
[b12]    if(*){
[b13]        assume(X);
[b14]        error: skip;
[b15]    }
[b16] }
```

Background: Boolean Programs - example

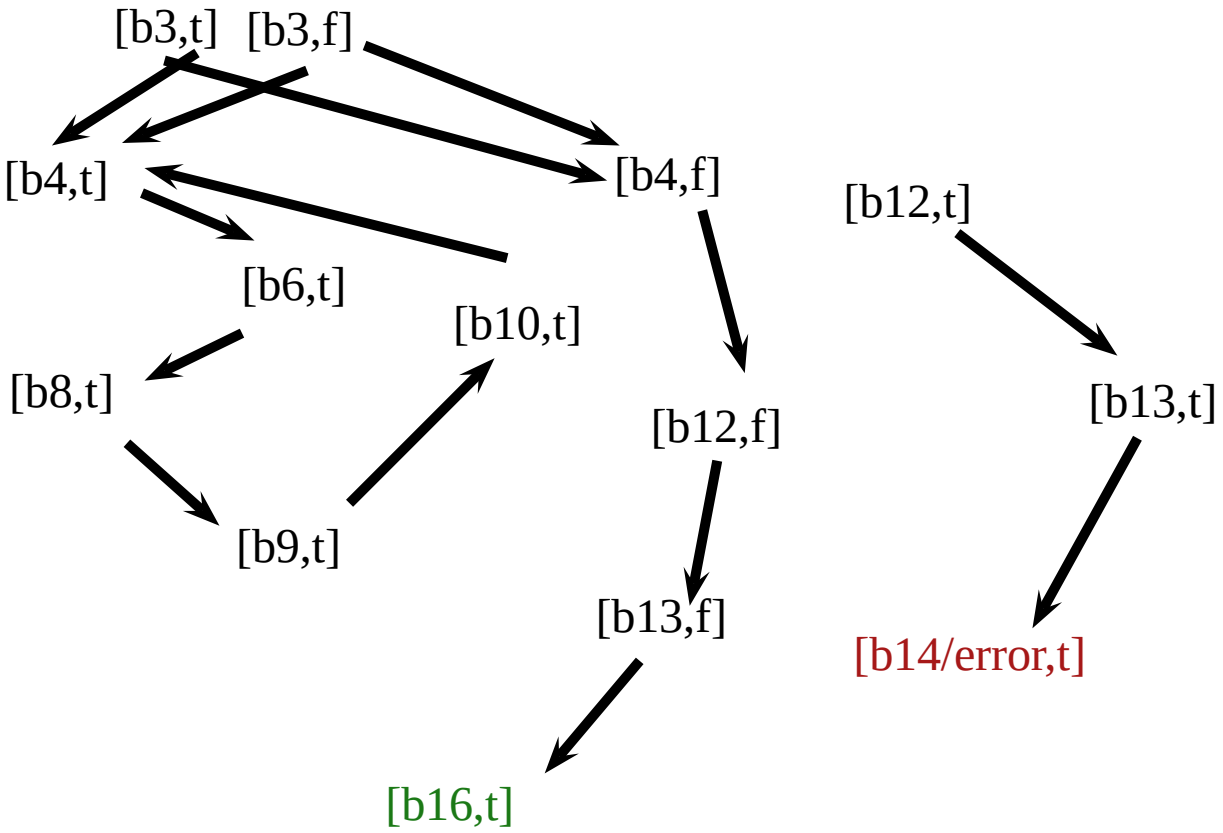
```
[p1] void main(){
[p2]     int a, b;
[p3]     cin >> a >> b;
[p4]     while(a > b){
[p5]         a++;
[p6]     }
[p7]     assert(a <= b);
[p8]     if(a > b) a = 1/0;
[p9] }
```

Predicate

$X = a > b$

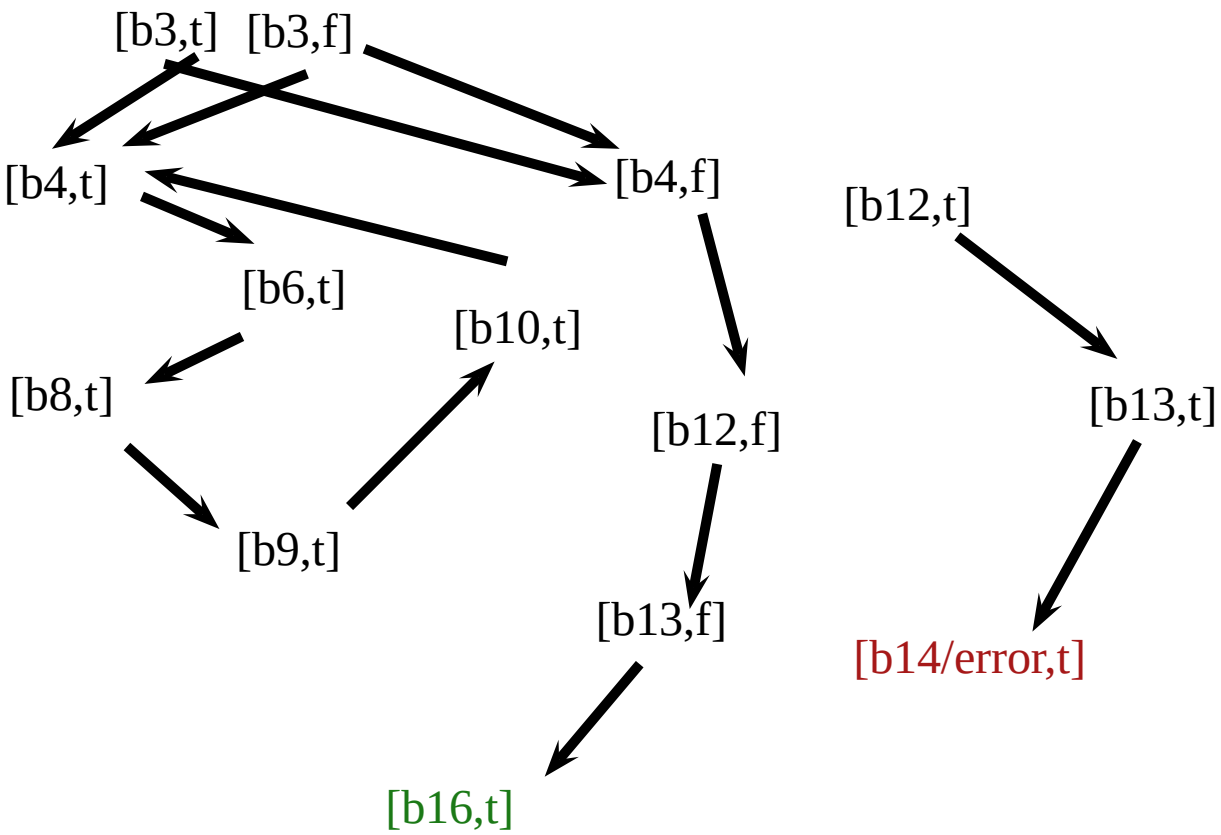
```
[b1] main(){
[b2]     bool X;
[b3]     X := *;
[b4]     do(*){
[b5]         assume(X);
[b6]         if(*){
[b7]           assume(X);
[b8]           X :=true;
[b9]         }
[b10]    }
[b11]    assume(!X)
[b12]    if(*){
[b13]       assume(X);
[b14]       error: skip;
[b15]    }
[b16] }
```

Background: Boolean Programs - model checking




```
[b1] main(){
[b2]     bool X;
[b3]     X := *;
[b4]     do(*){
[b5]         assume(X);
[b6]         if(*){
[b7]           assume(X);
[b8]           X :=true;
[b9]         }
[b10]    }
[b11]    assume(!X)
[b12]    if(*){
[b13]       assume(X);
[b14]       error: skip;
[b15]    }
[b16] }
```

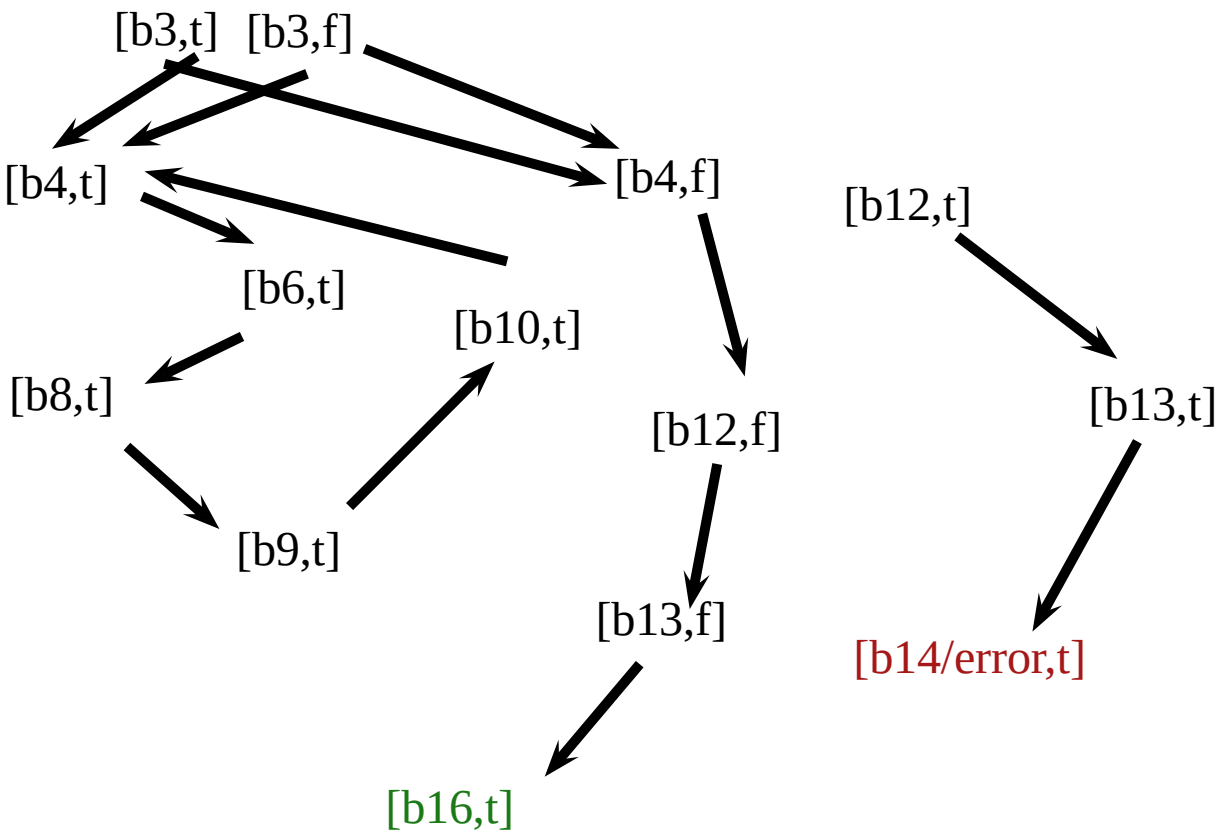
Background: Boolean Programs - model checking



Takeaways

- Use predicate abstraction and develop verification methods for boolean programs

Background: Boolean Programs - model checking



Takeaways

- Use predicate abstraction and develop verification methods for boolean programs
- Restricted to safety properties

At the end of this talk, I will know a way of...

Verifying Programs

At the end of this talk, I will know a way of...

Verifying Concurrent Programs

At the end of this talk, I will know a way of...

Verifying Recursive Concurrent Programs

Extending the model to cover more realistic programs

Recursion

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }
- State space no longer finite

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }
- State space no longer finite
- Sharir and Pnuelli Φ-functions, Push Down Systems, ICFGs

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }
- State space no longer finite
- Sharir and Pnuelli Φ-functions, Push Down Systems, ICFGs

Concurrency

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }
- State space no longer finite
- Sharir and Pnuelli Φ-functions, Push Down Systems, ICFGs

Concurrency

- Multiple threads with designated entry functions

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }
- State space no longer finite
- Sharir and Pnuelli Φ-functions, Push Down Systems, ICFGs

Concurrency

- Multiple threads with designated entry functions
- Shared memory model - global variables treated as shared variables

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }
- State space no longer finite
- Sharir and Pnuelli Φ-functions, Push Down Systems, ICFGs

Concurrency

- Multiple threads with designated entry functions
- Shared memory model - global variables treated as shared variables
- Add an atomic construct to the language *atomic* { . . . }

Extending the model to cover more realistic programs

Recursion

- Add function calls to the modelling language, *bool foo(bool . . .)* { . . . }
- State space no longer finite
- Sharir and Pnuelli Φ-functions, Push Down Systems, ICFGs

Concurrency

- Multiple threads with designated entry functions
- Shared memory model - global variables treated as shared variables
- Add an atomic construct to the language *atomic* { . . . }

Tough luck!

Proving safety properties for *concurrent recursive boolean programs* undecidable[7]

At the end of this talk, I will know a way of...

Verifying Concurrent Programs

At the end of this talk, I will know a way of...

Verifying Concurrent Programs via Induction

Inductive proofs on transition systems

Sound and Incomplete proof technique

Inductive proofs on transition systems

Sound and Incomplete proof technique

Transition System: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $\forall i \in I \quad P(i) i$
2. $\forall s, s' \in S, P(s) \wedge (s, s') \in \Delta \Rightarrow P(s')$

Inductive proofs on transition systems

Sound and Incomplete proof technique

Transition System: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $\forall i \in I \quad P(i) i$ (base case)
2. $\forall s, s' \in S, P(s) \wedge (s, s') \in \Delta \Rightarrow P(s')$

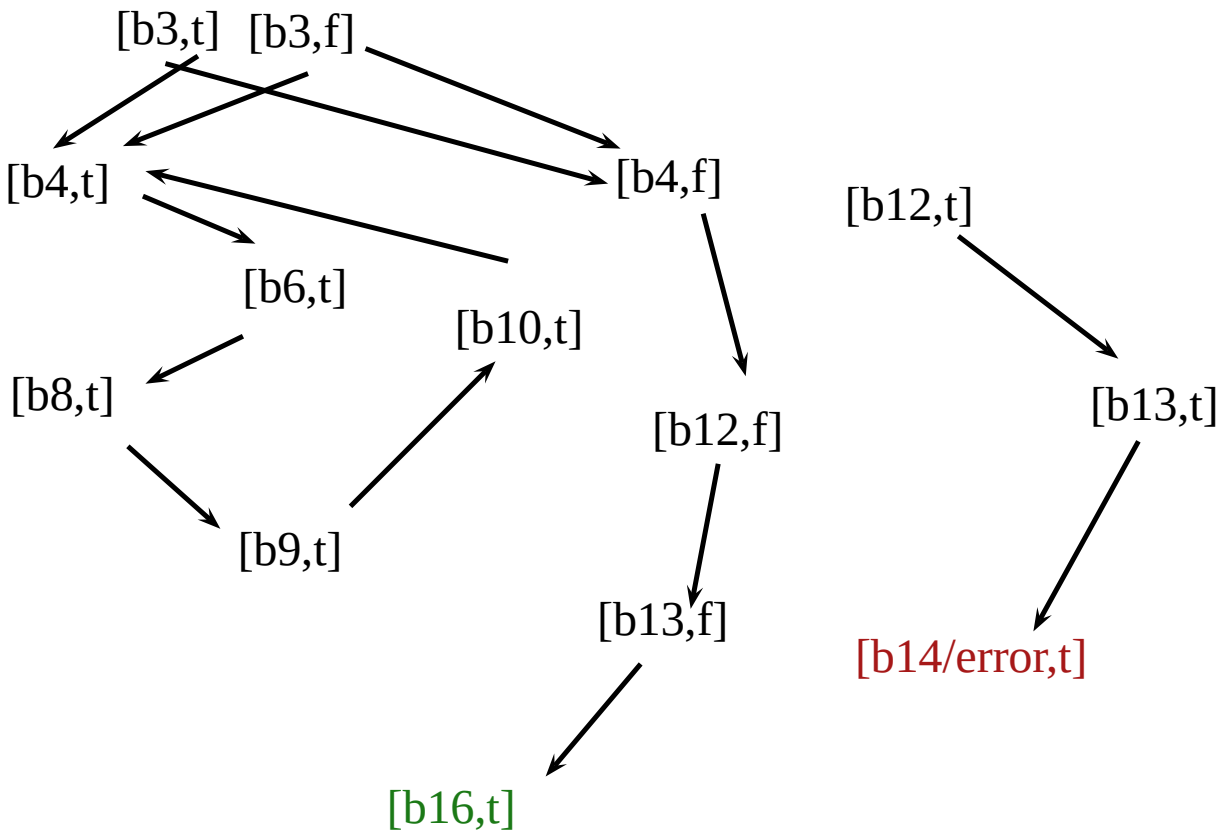
Inductive proofs on transition systems

Sound and Incomplete proof technique

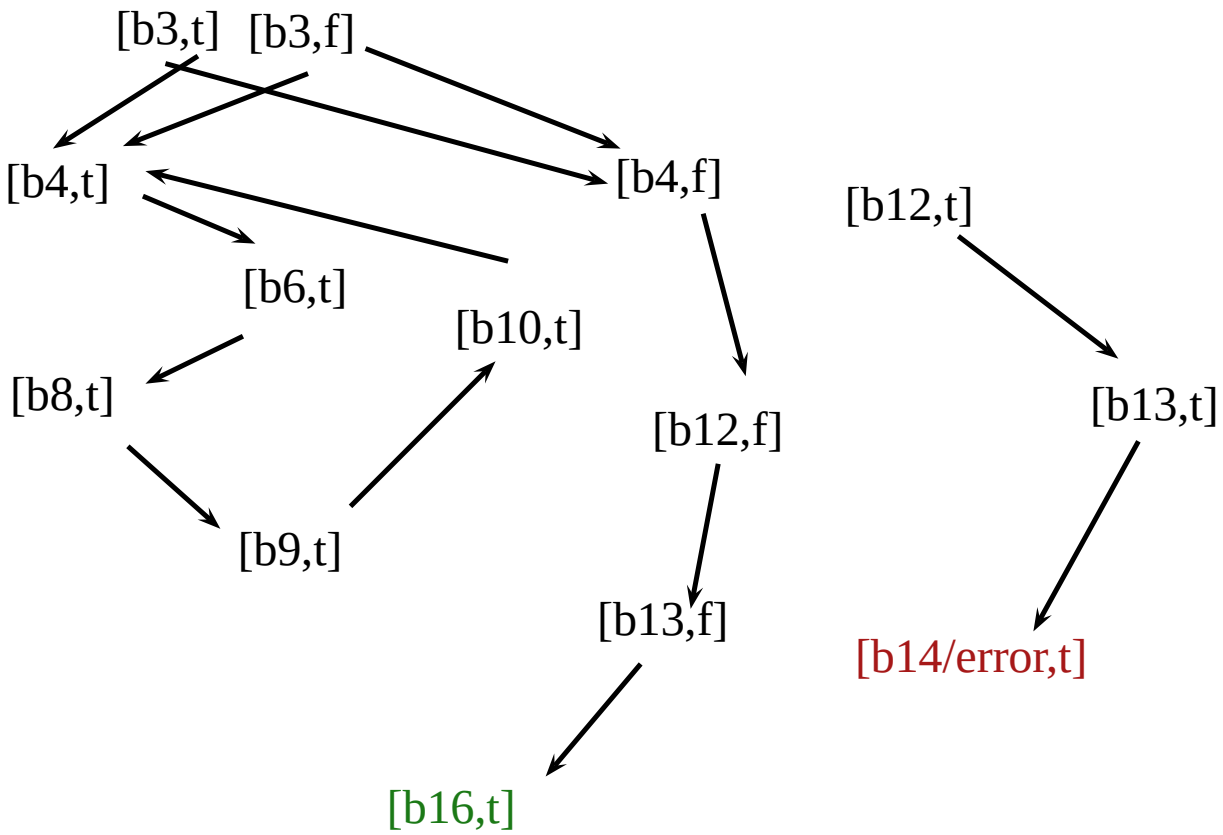
Transition System: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $\forall i \in I \quad P(i) i$ (base case)
2. $\forall s, s' \in S, P(s) \wedge (s, s') \in \Delta \Rightarrow P(s')$ (induction step)

Inductive proofs on transitions systems - example

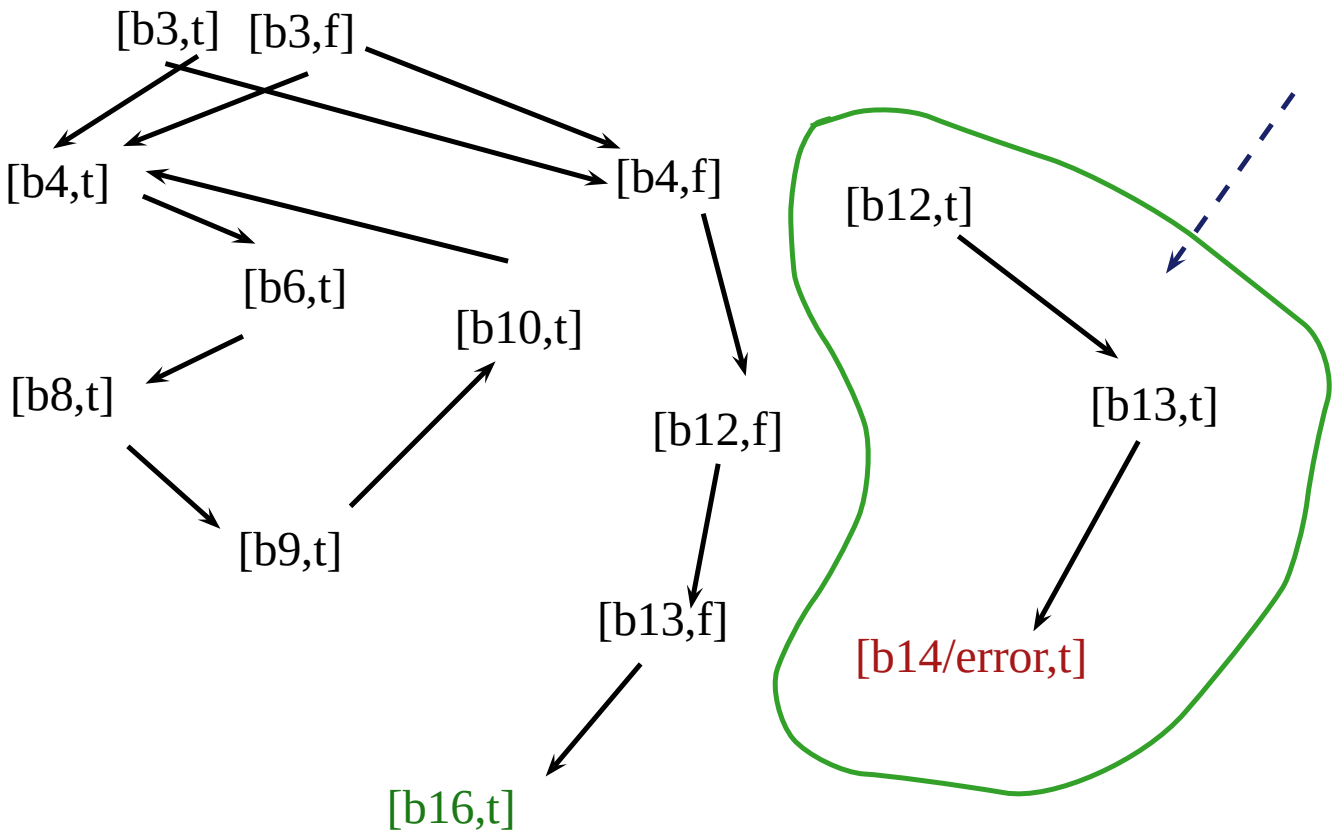


Inductive proofs on transitions systems - example



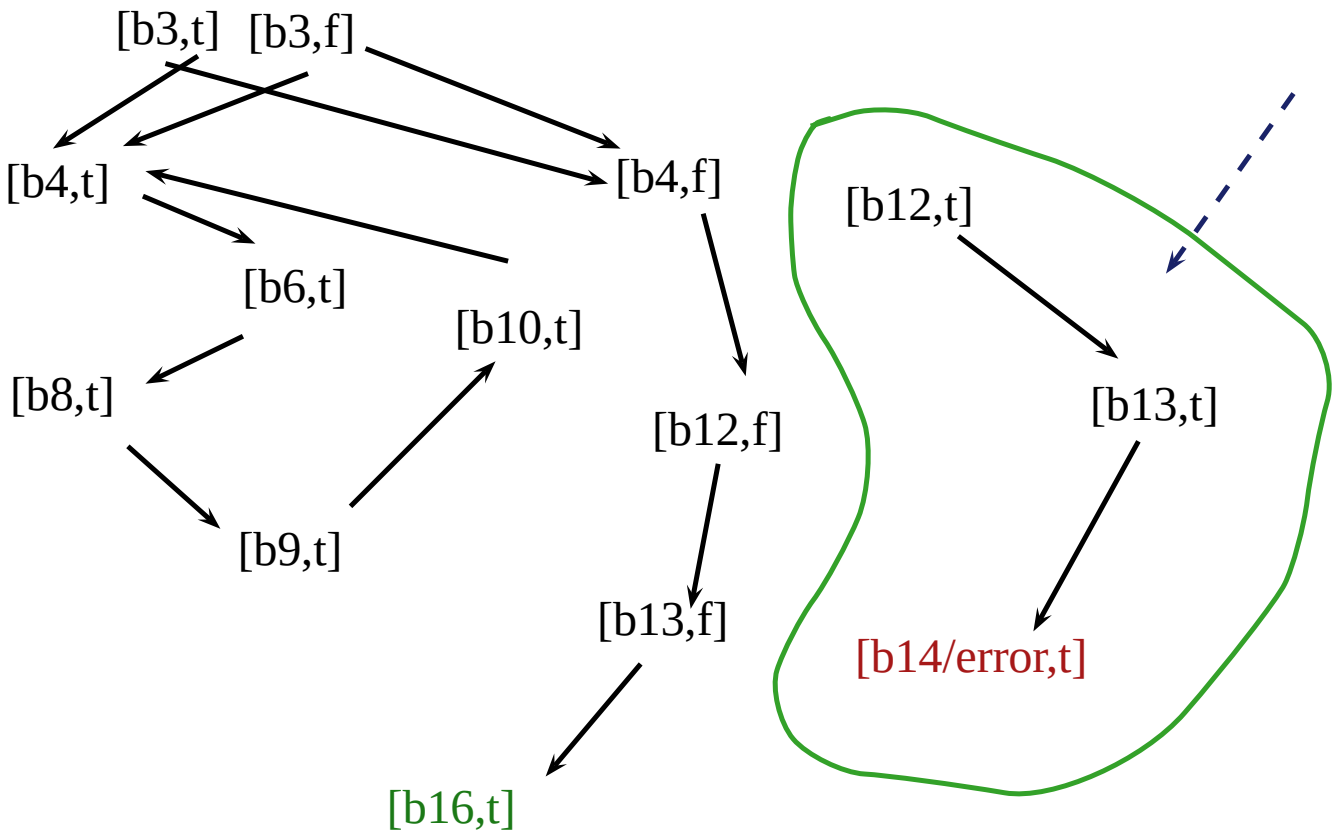
$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$

Inductive proofs on transitions systems - example



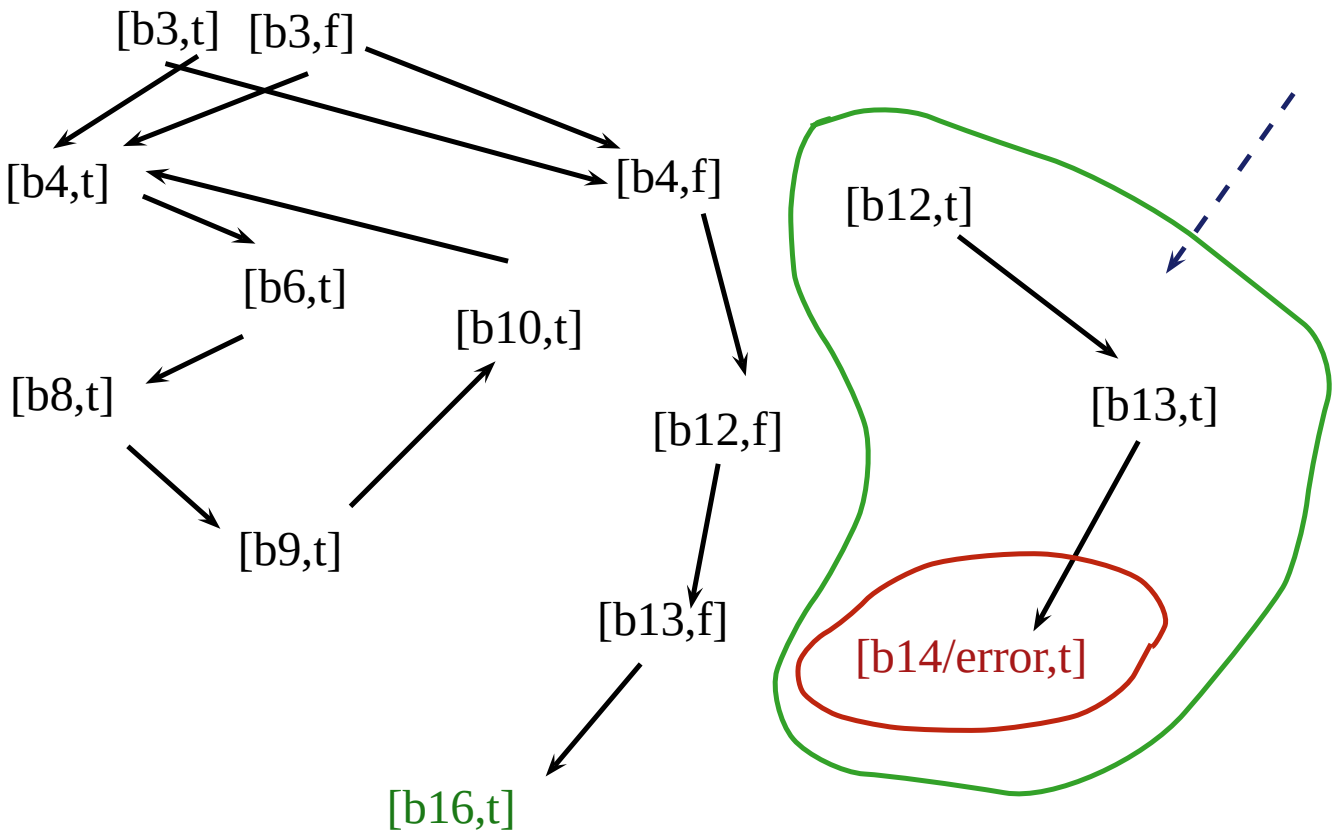
$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$

Inductive proofs on transitions systems - example



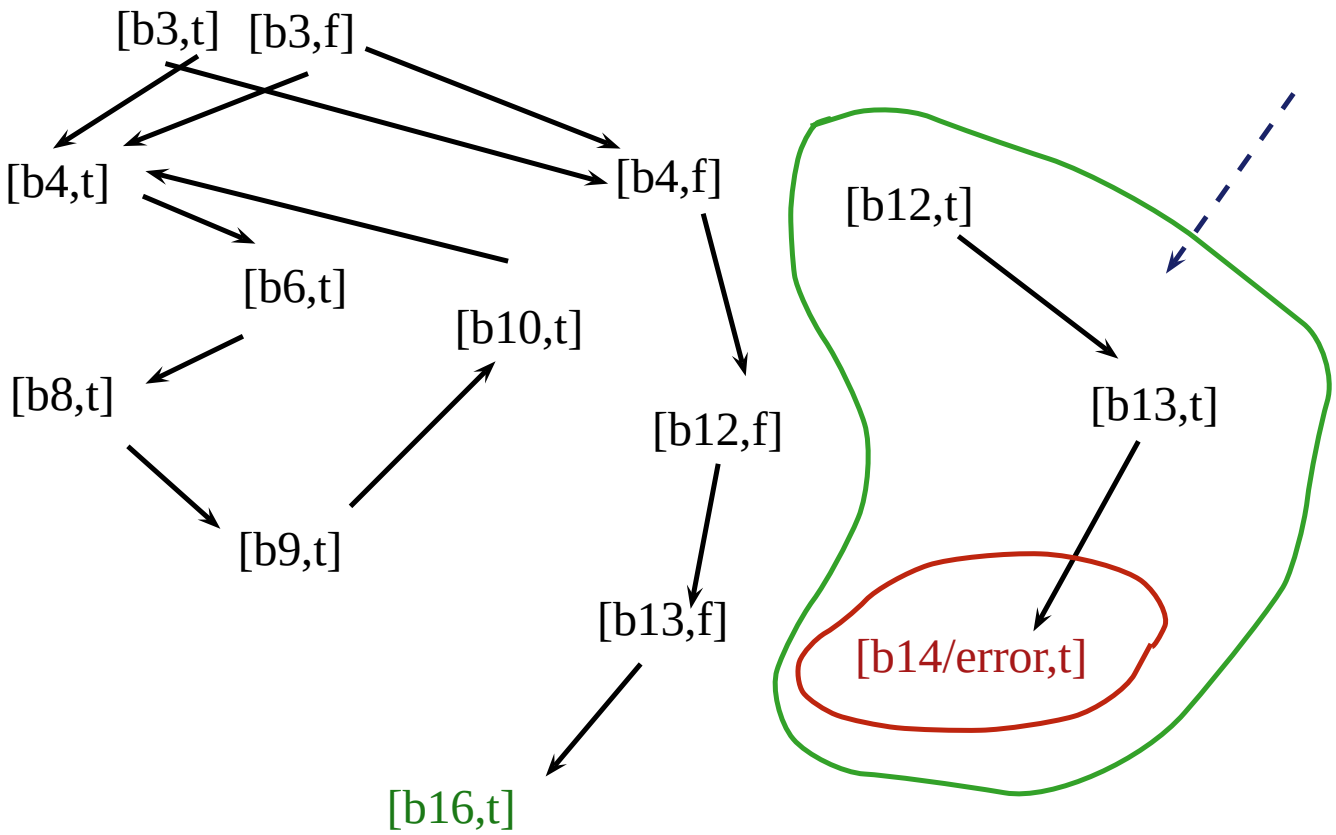
$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$
$P_{error} = S \backslash \{[error, t], [error, f]\}$

Inductive proofs on transitions systems - example



$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$
$P_{error} = S \backslash \{[error, t], [error, f]\}$

Inductive proofs on transitions systems - example



$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$

$P_{error} = S \backslash \{[error, t], [error, f]\}$

$P \Rightarrow P_{error}$ (Strengthening the property)

K-induction proofs on transition systems

Finding inductive strengthening of a property is hard!

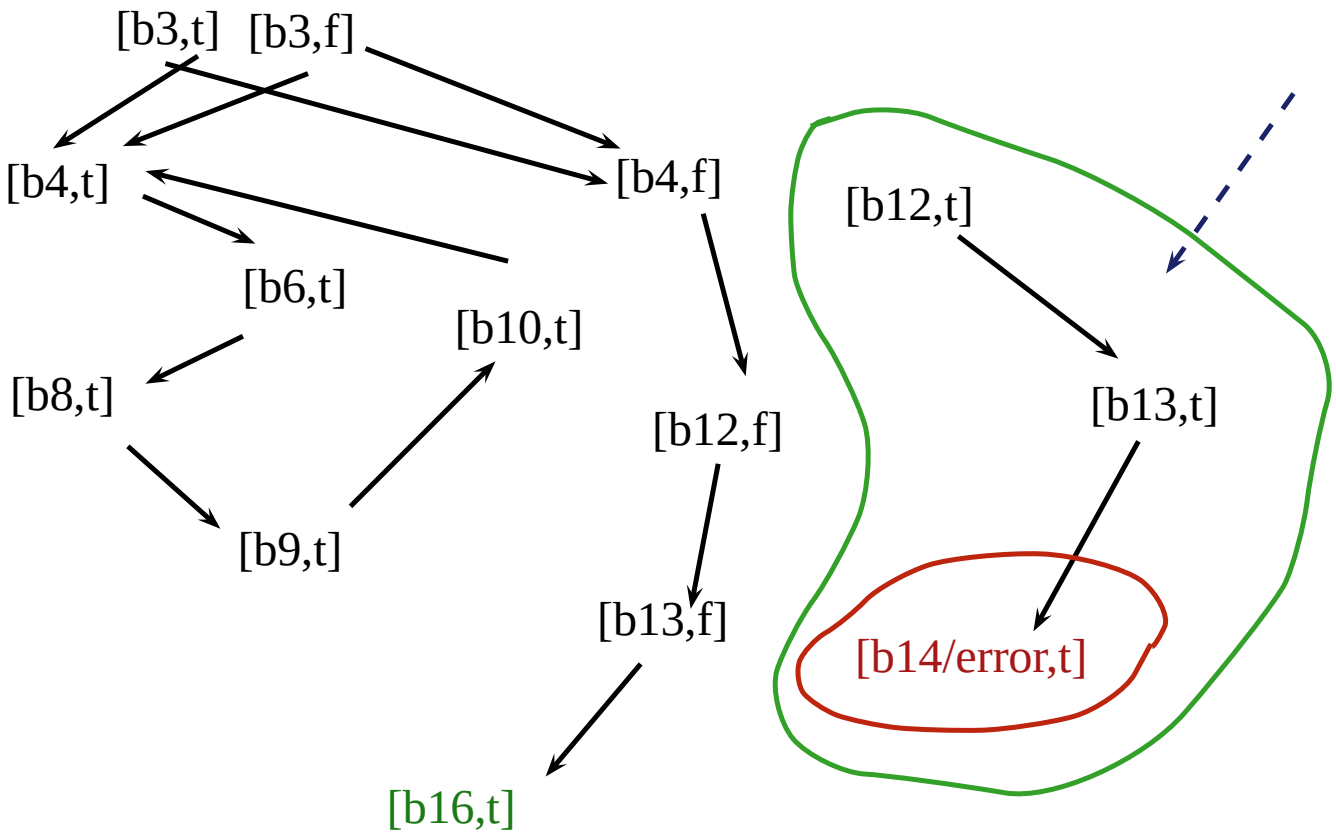
K-induction proofs on transition systems

Finding inductive strengthening of a property is hard!

Transition System[2]: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $\forall i_1, s_2 \ldots s_K \quad i_1 \in I \wedge i, s_2 \ldots s_K \in \rho \Rightarrow P(i_1) \wedge P(s_2) \ldots P(s_K)$
2. $\forall s_1, s_2, s_3 \ldots s_K, s_{K+1} \in \rho \quad P(s_1) \wedge P(s_2) \wedge P(s_3) \ldots P(s_K) \Rightarrow P(s_{K+1})$

K-induction proofs on transition systems - example

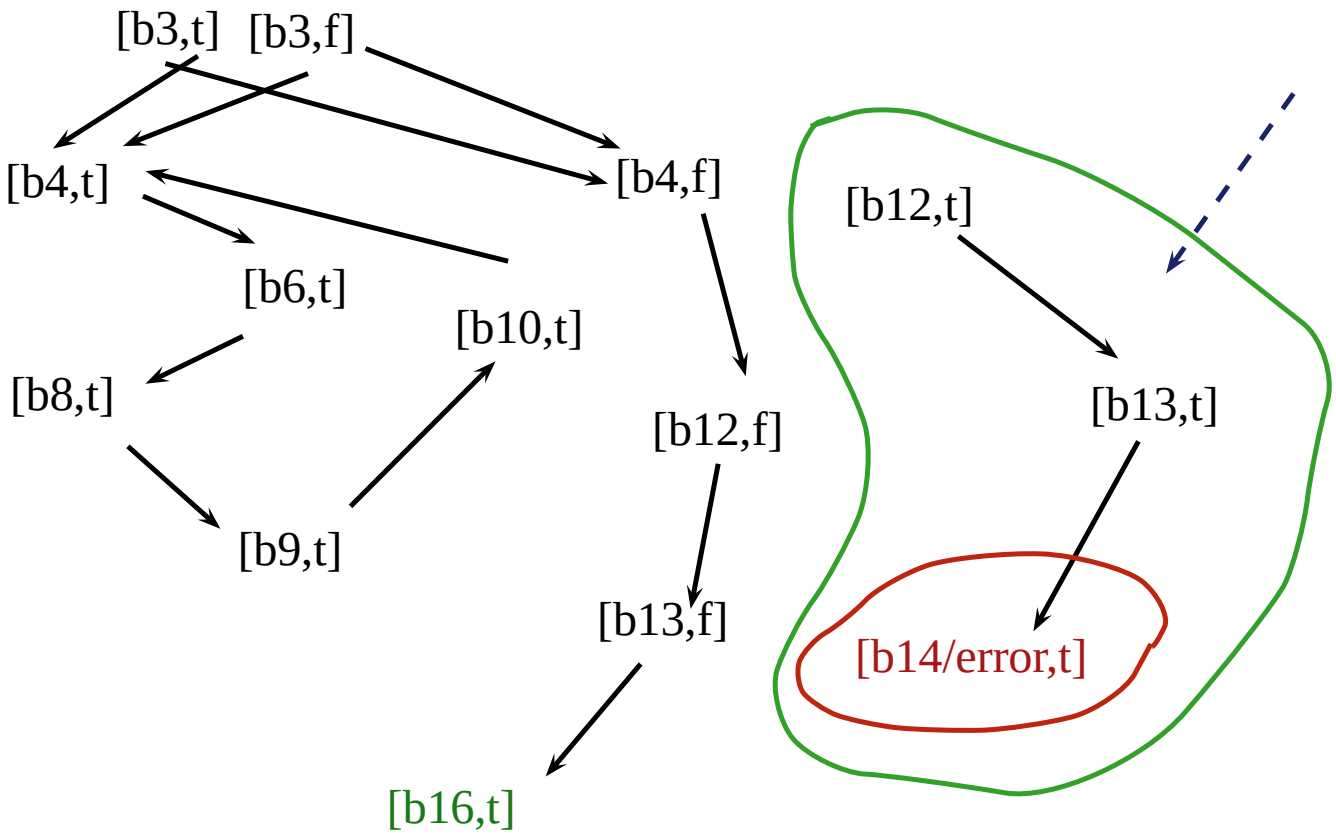


$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$

$P_{error} = S \backslash \{[error, t], [error, f]\}$

$P \Rightarrow P_{error}$ (Strengthening the property)

K-induction proofs on transition systems - example

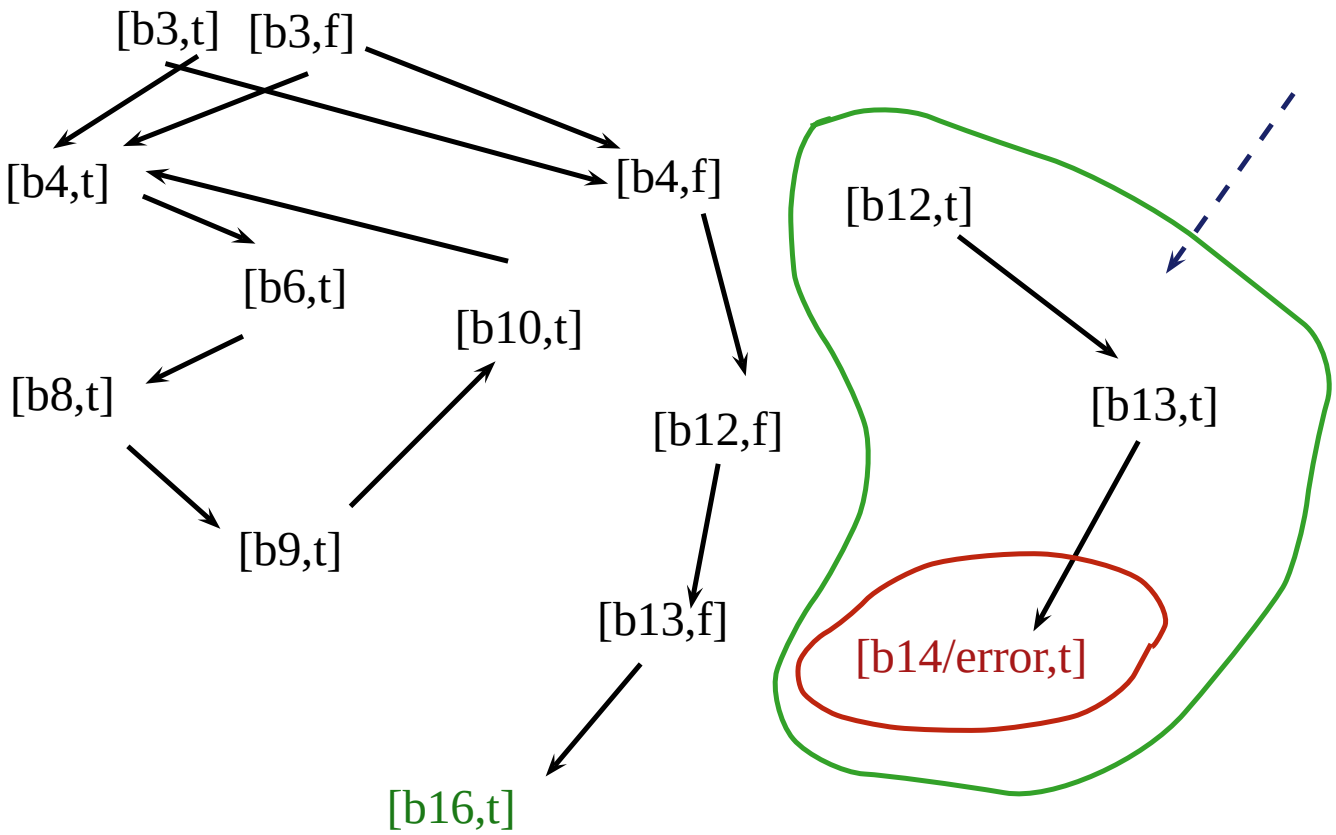


$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$ K = 1

$P_{error} = S \backslash \{[error, t], [error, f]\}$

$P \Rightarrow P_{error}$ (Strengthening the property)

K-induction proofs on transition systems - example



$P = S \backslash \{[b12, t], [b13, t], [error, t]\}$ K = 1

$P_{error} = S \backslash \{[error, t], [error, f]\}$ K = 3

$P \Rightarrow P_{error}$ (Strengthening the property)

Extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

States of the transition system

Values of the Global Variable

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

States of the transition system

Values of the Global Variable
Activation stack of each of the threads - (local variable values + program counter)

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

States of the transition system

Values of the Global Variable
Activation stack of each of the threads - (local variable values + program counter)

$$\mathbb{G}lobals \rightarrow \{t, f\} \times \Pi_i^{i \in \mathbb{T}hreads}(pc_i \times \mathbb{L}ocals \rightarrow \{t, f\})^*$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

States of the transition system

Values of the Global Variable
Activation stack of each of the threads - (local variable values + program counter)

$$\mathbb{G}lobals \rightarrow \{t, f\} \times \Pi_i^{i \in \mathbb{T}hreads}(pc_i \times \mathbb{L}ocals \rightarrow \{t, f\})^*$$

$$[x, y, pc_p, pc_q]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

$[f, f, p1, q1] \rightsquigarrow [f, f, p1, q2] \rightsquigarrow [f, f, p1, l1] \rightsquigarrow \ldots [t, f, p1, q3] \rightsquigarrow [t, t, p1, q3] \rightsquigarrow [t, t, p2, q3] \rightsquigarrow [t, t, l1, q3] \rightsquigarrow [t, t, l4, q3] \ldots$

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Base Case

$[f, f, p1, q1] \rightsquigarrow [f, f, p2, q1] \rightsquigarrow [f, f, l2, q1] \rightsquigarrow \ldots [t, f, p3, q1] \rightsquigarrow [t, t, p4, q1] \rightsquigarrow [t, t, p4, q2] \rightsquigarrow [t, t, p4, l1] \rightsquigarrow [t, t, p4, l4] \ldots$

$[f, f, p1, q1] \rightsquigarrow [f, f, p1, q2] \rightsquigarrow [f, f, p1, l1] \rightsquigarrow \ldots [t, f, p1, q3] \rightsquigarrow [t, t, p1, q3] \rightsquigarrow [t, t, p2, q3] \rightsquigarrow [t, t, l1, q3] \rightsquigarrow [t, t, l4, q3] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

Induction case

K=2:

$[f, t, p6, q3] \rightarrow [f, t, p6, q4] \rightarrow [f, t, p6, q3] \rightarrow [f, t, error, q3]$

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

Induction case

K=2:

$[f, t, p6, q3] \rightarrow [f, t, p6, q4] \rightarrow [f, t, p6, q3] \rightarrow [f, t, error, q3]$

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Induction case

K=2:

$[f, t, p6, q3] \rightarrow [f, t, p6, q4] \rightarrow [f, t, p6, q3] \rightarrow [f, t, error, q3]$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Induction case

K=2:

$[f, t, p6, q3] \rightarrow [f, t, p6, q4] \rightarrow [f, t, p6, q3] \rightarrow [f, t, error, q3]$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Induction case

K=2:
$[f, t, p6, q3] \rightarrow [f, t, p6, q4] \rightarrow [f, t, p6, q3] \rightarrow [f, t, error, q3]$
K=3:
$[f, t, p6, q4] \rightarrow [f, t, p6, q3] \ldots$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

Induction case

K=2:

$[f, t, p6, q3] \rightarrow [f, t, p6, q4] \rightarrow [f, t, p6, q3] \rightarrow [f, t, error, q3]$

K=3:

$[f, t, p6, q4] \rightarrow [f, t, p6, q3] \ldots$

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction - extended example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Induction case

K=2:
$[f, t, p6, q3] \rightarrow [f, t, p6, q4] \rightarrow [f, t, p6, q3] \rightarrow [f, t, error, q3]$
K=3:
$[f, t, p6, q4] \rightarrow [f, t, p6, q3] \ldots$

Haven't *really* faced concurrency yet!

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

At the end of this talk, I will know a way of...

Verifying Concurrent Programs via Induction

At the end of this talk, I will know a way of...

Verifying Concurrent Programs via Bounded Context-Switching and Induction

Context Bounded Analysis[5, 6]

- under-approximation of the behaviour of the program
- Bug-finding technique

Context Bounded Analysis[5, 6]

- under-approximation of the behaviour of the program
- Bug-finding technique

Context Switch

A conceptual step in which the scheduler chooses the next thread to continue its execution among all the active threads.

Context Bounded Analysis[5, 6]

- under-approximation of the behaviour of the program
- Bug-finding technique

Context Switch

A conceptual step in which the scheduler chooses the next thread to continue its execution among all the active threads.

Idea

Check the required property for all states reachable from the start state within C context switches.

$\cdot \overset{p}{\rightsquigarrow} \cdot \overset{CS}{\rightarrow}$
$\cdot \overset{q}{\rightsquigarrow} \cdot \overset{CS}{\rightarrow}$
$\cdot \overset{p}{\rightsquigarrow} \ldots$

Context Bounded Analysis - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Context Bounded Analysis - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

C=2

$[f, f, p1, q1] \rightarrow [f, f, p2, q1] \rightarrow [f, f, l1, q1] \overset{CS}{\rightsquigarrow}$
$[f, f, l1, q1] \rightarrow [f, f, l1, q2] \overset{CS}{\rightsquigarrow} [f, f, l1, q2] \rightarrow$
$[f, f, l2, q2]$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Context Bounded Analysis - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

C=2

$[f, f, p1, q1] \rightarrow [f, f, p2, q1] \rightarrow [f, f, l1, q1] \overset{CS}{\rightsquigarrow}$
$[f, f, l1, q1] \rightarrow [f, f, l1, q2] \overset{CS}{\rightsquigarrow} [f, f, l1, q2] \rightarrow$
$[f, f, l2, q2]$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Context Bounded Analysis - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

C=2

$[f, f, p1, q1] \rightarrow [f, f, p2, q1] \rightarrow [f, f, l1, q1] \overset{CS}{\rightsquigarrow}$
$[f, f, l1, q1] \rightarrow [f, f, l1, q2] \overset{CS}{\rightsquigarrow} [f, f, l1, q2] \rightarrow$
$[f, f, l2, q2]$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Context Bounded Analysis - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Not with C=2

$$[f, f, p1, q1] \rightarrow [f, f, p2, q1] \rightarrow [f, f, l1, q1] \overset{CS}{\rightsquigarrow}$$
$$[f, f, l1, q1] \rightarrow [f, f, l1, q2] \overset{CS}{\rightsquigarrow} [f, f, l1, q2] \rightarrow$$
$$[f, f, l2, q2] \overset{CS}{\rightsquigarrow} [f, f, l2, q2] \rightarrow [f, f, l2, l1]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Context Bounded Analysis - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

Not with C=2

$$[f, f, p1, q1] \rightarrow [f, f, p2, q1] \rightarrow [f, f, l1, q1] \overset{CS}{\rightsquigarrow}$$
$$[f, f, l1, q1] \rightarrow [f, f, l1, q2] \overset{CS}{\rightsquigarrow} [f, f, l1, q2] \rightarrow$$
$$[f, f, l2, q2] \overset{CS}{\rightsquigarrow} [f, f, l2, q2] \rightarrow [f, f, l2, l1]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

Context Bounded Analysis - Continued

- Efficient algorithms exist to check safety properties within bounded contexts
- Sequentialization

Context Bounded Analysis - Continued

- Efficient algorithms exist to check safety properties within bounded contexts
- Sequentialization

Black Box

$$\tau(P, C, error) = P_C^{Seq}$$

$$P_C^{Seq}(i) :^? i \overset{P}{\underset{C}{\rightsquigarrow}} error$$

Call this check $CBA[P, C, i]$

CBA + K-induction = Verification?

Technique

K-induction while inducting over the number of context switches instead of program steps

CBA + K-induction = Verification?

Technique

K-induction while inducting over the number of context switches instead of program steps

Proof of correctness??

CBA + K-induction = Verification?

Technique

K-induction while inducting over the number of context switches instead of program steps

Proof of correctness??

Base Case

$$\forall i_1, s_2 \ldots s_K \quad i_1 \in I \wedge i, s_2 \ldots s_K \in \rho \Rightarrow P(i_1) \wedge P(s_2) \ldots P(s_K)$$

Context Bounded Analysis with C = K, $CBA[P, K, i]$

CBA + K-induction = Verification?

Technique

K-induction while inducting over the number of context switches instead of program steps

Proof of correctness??

Base Case

$$\forall i_1, s_2 \ldots s_K \quad i_1 \in I \wedge i, s_2 \ldots s_K \in \rho \Rightarrow P(i_1) \wedge P(s_2) \ldots P(s_K)$$

Context Bounded Analysis with C = K, $CBA[P, K, i]$

Induction Step

$$\forall s_1, s_2, s_3 \ldots s_K, s_{K+1} \in \rho \quad P(s_1) \wedge P(s_2) \wedge P(s_3) \ldots P(s_K) \Rightarrow P(s_{K+1})$$

- Generate an arbitrary program state, s.
- Modify $CBA'[P, K, s]$: Check if *error* is reached within K context switches. If not, verify that *error* isn't reached within the next context.

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=1

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=1

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=1

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=3

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow}$$
$$[f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow}$$
$$[f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=3

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow}$$
$$[f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow}$$
$$[f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=3

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow}$$
$$[f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow}$$
$$[f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=3

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow}$$
$$[f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow}$$
$$[f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + K-induction = Verification? - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=3

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow}$$
$$[f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow}$$
$$[f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + induction step : Challenge I

Reduction to program steps

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow}$$

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow}$$

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow}$$

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

CBA + induction step : Challenge I

Reduction to program steps

$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$

$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow}$
$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$

$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow}$
$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow}$
$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$

$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow}$
$[f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$
$[f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$

Solution: improved-induction[1]

- Only consider paths with the *least number of* context switches.
- Maximum sequential reach for each context switch.

Solution: improved-induction[1]

- Only consider paths with the *least number of* context switches.
- Maximum sequential reach for each context switch.

K-induction

Transition System: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$, $P(s)$ true iff the required property holds for state s.
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $\forall i_1, s_2 \ldots s_K \quad i_1 \in I \wedge i, s_2 \ldots s_K \in \rho \Rightarrow P(i_1) \wedge P(s_2) \ldots P(s_K)$
2. $\forall s_1, s_2, s_3 \ldots s_K, s_{K+1} \in \rho \quad P(s_1) \wedge P(s_2) \wedge P(s_3) \ldots P(s_K) \Rightarrow P(s_{K+1})$

Solution: improved-induction[1]

- Only consider paths with the *least number of* context switches.
- Maximum sequential reach for each context switch.

K-induction

Transition System: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$, $P(s)$ true iff the required property holds for state s.
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $CBA[P, K, i]$
2. $\forall s_1, s_2, s_3 \ldots s_K, s_{K+1} \in \rho \quad P(s_1) \wedge P(s_2) \wedge P(s_3) \ldots P(s_K) \Rightarrow P(s_{K+1})$

Solution: improved-induction[1]

- Only consider paths with the *least number of* context switches.
- Maximum sequential reach for each context switch.

improved K-induction

Transition System: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$, $P(s)$ true iff the required property holds for state s.
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $CBA[P, K, i]$
2. $(\forall s_1, \ldots s_K \in \rho P(s_1) \wedge P(s_2) \wedge P(s_3) \ldots P(s_K)) \Rightarrow$
   $(\forall s_1, s_2, s_3 \ldots s_K, s_{K+1} \in \rho \quad P(s_{K+1}))$

Solution: improved-induction[1]

- Only consider paths with the *least number of* context switches.
- Maximum sequential reach for each context switch.

K-induction with amplification

Transition System: $T = (S, \Delta \subseteq S \times S, I)$
Property: $P : S \rightarrow \{t, f\}$, $P(s)$ true iff the required property holds for state s.
$\forall s \in S_{Reach} \quad P(s) \iff$

1. $CBA[P, K, i]$
2. $\forall s \in S \quad ! \; CBA[P, K, s] \vee CBA[P, K+1, s]$

K-induction with amplification - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=2

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow}$$
$$[f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

K-induction with amplification - example

```
[l1] bool x,y;
[l2] lock()
[l3]   atomic
[l4]     do(*)
[l5]       assume(x);
[l6]     assume(!x);
[l7]     x = true;
```

K=2

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

$$[f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{1}{\rightsquigarrow} [f, f, p4, q3] \overset{CS}{\rightarrow} [f, f, p4, q3] \overset{2}{\rightsquigarrow} [f, t, p4, q5] \overset{CS}{\rightarrow} [f, t, p4, q5] \overset{1}{\rightsquigarrow} [f, t, error, q5]$$

```
[p1] p()
[p2]   lock()
[p3]   y = false
[p4]   do(*)
[p5]     assume(!y);
[p6]   assume(y);
[p7]   error: skip;
```

```
[q1] q()
[q2]   lock();
[q3]   do(*)
[q4]     skip;
[q5]   y = true;
```

CBA + induction : Challenge II

$$\forall s \in S \quad ! \; CBA[P, K, s] \vee CBA[P, K+1, s]$$

- Infinite state space
- Need to prune the states considered

CBA + induction : Challenge II

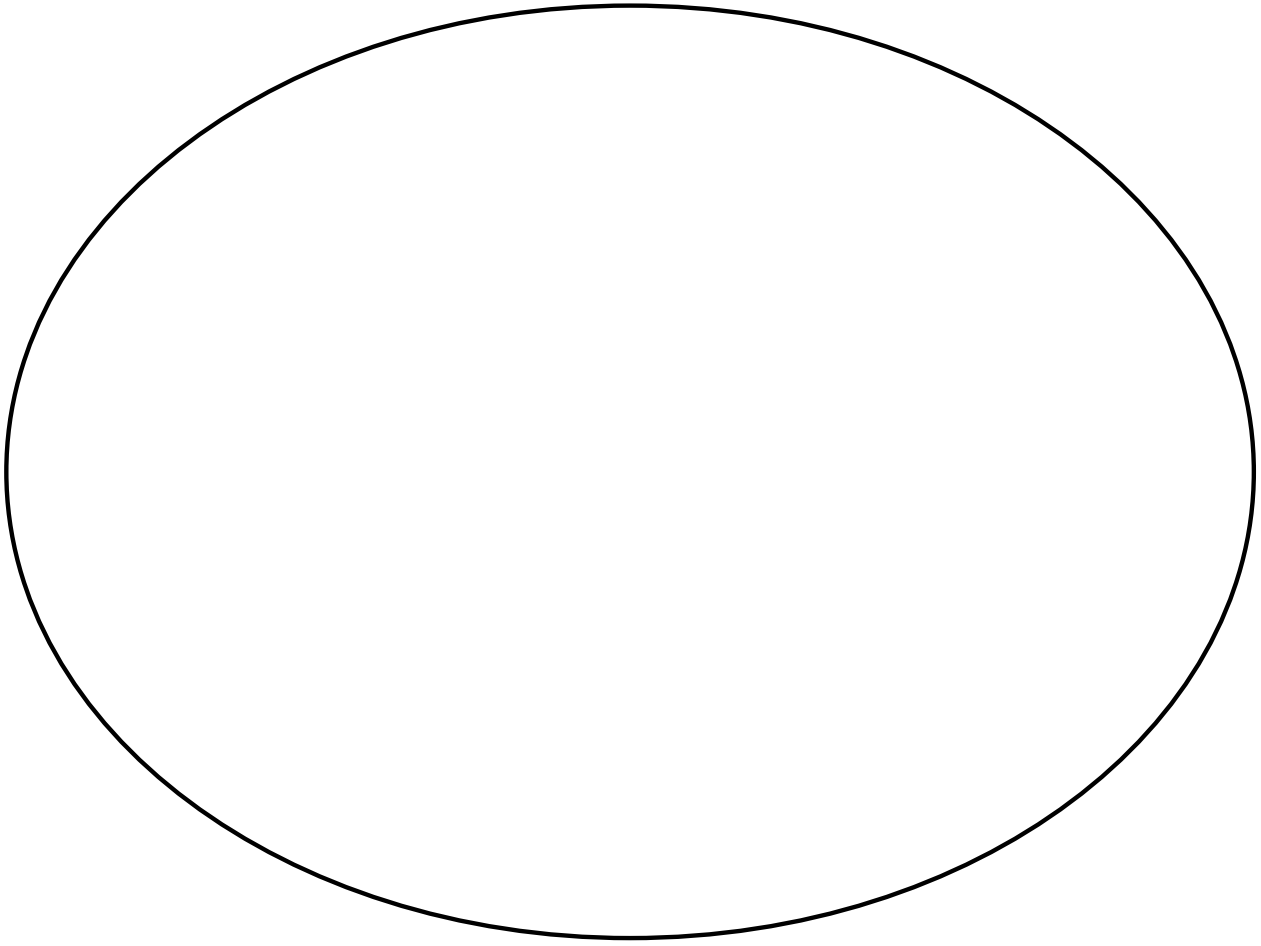
$$\forall s \in S \quad ! \; CBA[P, K, s] \vee CBA[P, K+1, s]$$

- Infinite state space
- Need to prune the states considered

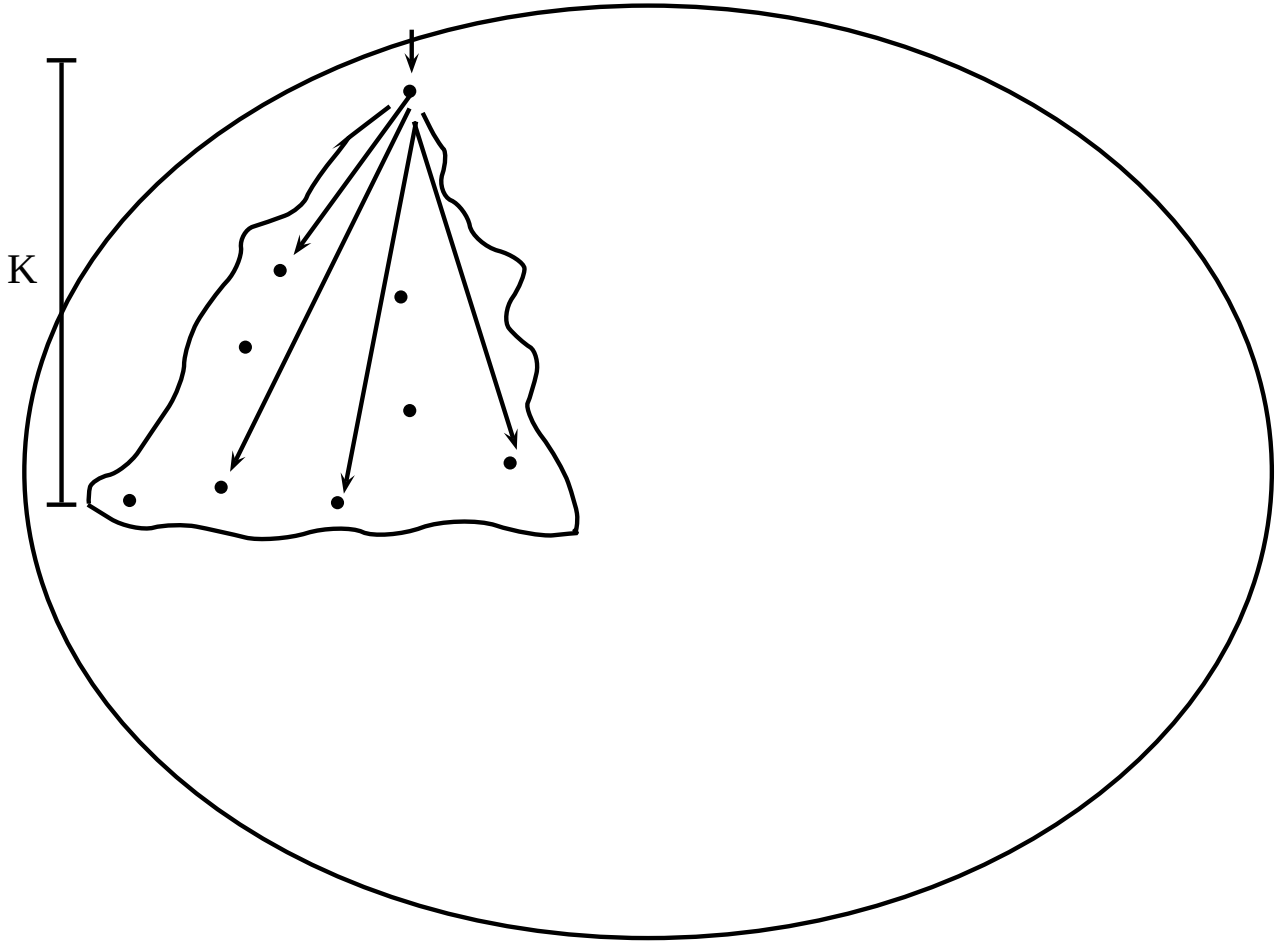
Express in terms of backward reachability and set difference

$$CBA^{rev}[P, K+1, error] - CBA^{rev}[P, K, error] = \emptyset$$

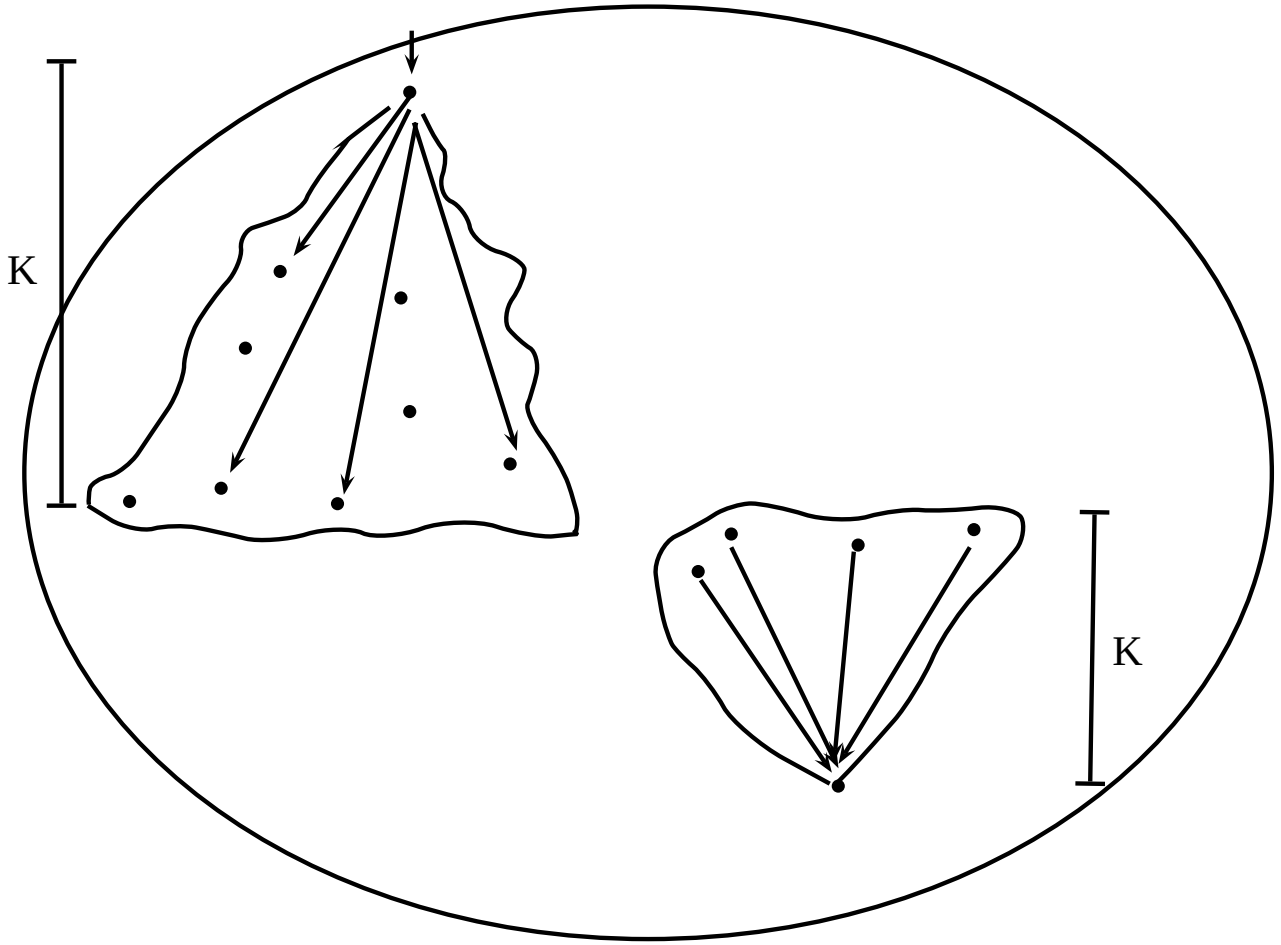
K-induction explained



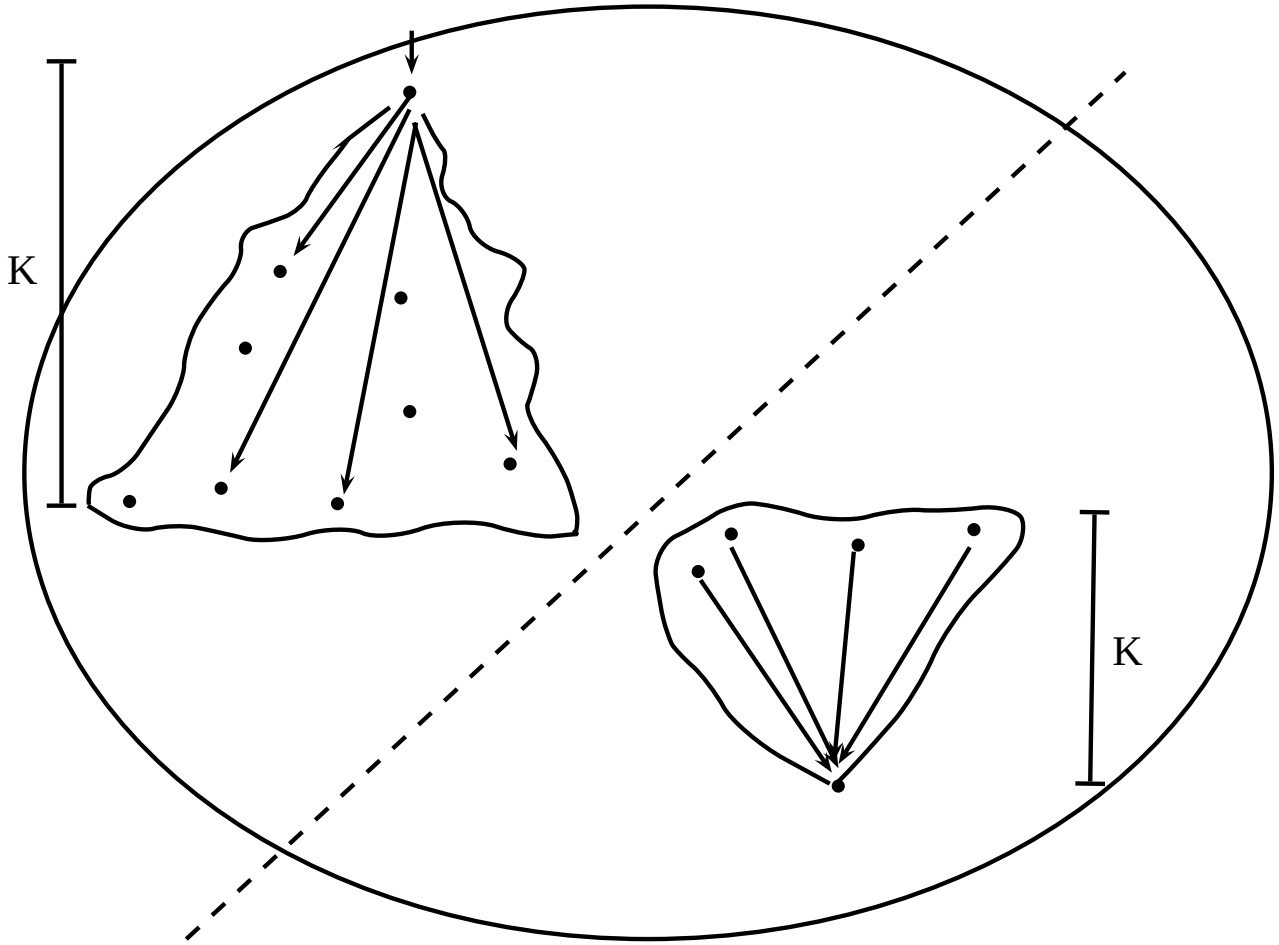
K-induction explained



K-induction explained



K-induction explained



The algorithm

Semi-decision procedure for concurrent recursive boolean programs:

K-induction-amplification

1: **for** $K = 1; true; K = K + 1$ **do**
2: **if** CBA^K(P) returns reachable **then**
3: // Error is reachable: bug found
4: EXIT
5: // Compute the set of states that reach error in K contexts, M^K
6: $M^K = prune(CBA^{Rev}[P, K, error])$
7: // Compute the set of states that reach error in $K + 1$ contextx, M$^K + 1$
8: $M^{K+1} = prune(CBA^{Rev}[P, K + 1, error])$
9: **if** $M^{K+1} - M^K = \emptyset$ **then**
10: // Both obligations passed: program proved correct
11: EXIT

Gotcha!

Reality Check - Incompleteness

For a give program P, a K that works may not exist

Gotcha!

Reality Check - Incompleteness

For a give program P, a K that works may not exist

Reality Check - Hardness

$$\forall s \in S \quad CBA[P, K, s] \vee! \; CBA[P, K+1, s]$$

- Quantifier alternation

Gotcha!

Reality Check - Incompleteness

For a give program P, a K that works may not exist

Reality Check - Hardness

$$\forall s \in S \quad CBA[P, K, s] \vee! \; CBA[P, K+1, s]$$

- Quantifier alternation

$$CBA^{rev}[P, K+1, error] - CBA^{rev}[P, K, error] = \emptyset$$

- Set subtraction
- Finite State Machine Complementation

Experiments

Experiments on an implementation using MOPED[3]. Models from a Bluetooth driver model and some mutual exclusion protocols[4].

Program	#threads	#globals	#locals	K	Time(s)
blue2	2	5	6	3	32.44
dekkers	2	4	1	3	3.74
lock_unlock_K	2	2	1	4	0.6
lock_unlock_3	3	4	3	3	44.02
petersons	2	4	1	3	0.3
petersons_loop	2	4	1	3	6.05

Experiments

Experiments on an implementation using MOPED[3]. Models from a Bluetooth driver model and some mutual exclusion protocols[4].

Program	#threads	#globals	#locals	K	Time(s)
blue2	2	5	6	3	32.44
dekkers	2	4	1	3	3.74
lock_unlock_K	2	2	1	4	0.6
lock_unlock_3	3	4	3	3	44.02
petersons	2	4	1	3	0.3
petersons_loop	2	4	1	3	6.05

Need symbolic methods for the set (automata) subtraction in the last step to make the technique efficient.

Takeaways

- Concurrent program verification is an important and hard problem.

Takeaways

- Concurrent program verification is an important and hard problem.
- There exist bug founding approaches that only explore bounded executions of the program.

Takeaways

- Concurrent program verification is an important and hard problem.
- There exist bug founding approaches that only explore bounded executions of the program.
- Induction built over these Bounded Context methods lets us directly address the complexity arising from interleaving of threads.

Takeaways

- Concurrent program verification is an important and hard problem.
- There exist bug founding approaches that only explore bounded executions of the program.
- Induction built over these Bounded Context methods lets us directly address the complexity arising from interleaving of threads.
- A stronger version of K-induction is needed to effectively induct over these path blocks. Proving the induction step in this modified induction is harder.

Further Reading I


K. Claessen.
Induction and state machines.
In *Winter Meeting of the Computing Science Dept.* Chalmers Univ. of Tech., Sweden, 1999.


A. Donaldson, L. Haller, D. Kroening, and P. Ruemmer.
Software verification using k-induction.
In *SAS*, 2011.


J. Esparza and S. Schwoon.
A BDD-based model checker for recursive programs.
In *CAV*. 2001.


N. Kidd.
The Bluetooth driver models, 2009.
http://pages.cs.wisc.edu/~kidd/bluetooth.

Further Reading II

A. Lal and T. Reps.
Reducing concurrent analysis under a context bound to sequential analysis.
In *CAV*. 2008.

S. Qadeer and J. Rehof.
Context-bounded model checking of concurrent software.
In *TACAS*, 2005.

G. Ramalingam.
Context-sensitive synchronization-sensitive analysis is undecidable.
TOPLAS, 22(2), 2000.

Thank you!

Thank you for listening...

Questions?