Abstraction-Guided Synthesis of Synchronization
Master Seminar SS'2012

Sergey Grebenshchikov

Is this program correct?

```
T1() {          T2() {          T3() {
  x+=z;           z++;            y1 = f(x);
  x+=z;           z++;            y2 = x;
                                  assert( y1 ≠ y2 );
}               }               }

                                f(x) {
                                  if(x == 1)
                                    return 3;
                                  else if(x == 2)
                                    return 6;
                                  else return 5;
                                }
```

Figure: Example: $T1 \parallel T2 \parallel T3$, f atomic, all variables initially 0

Is this program correct?

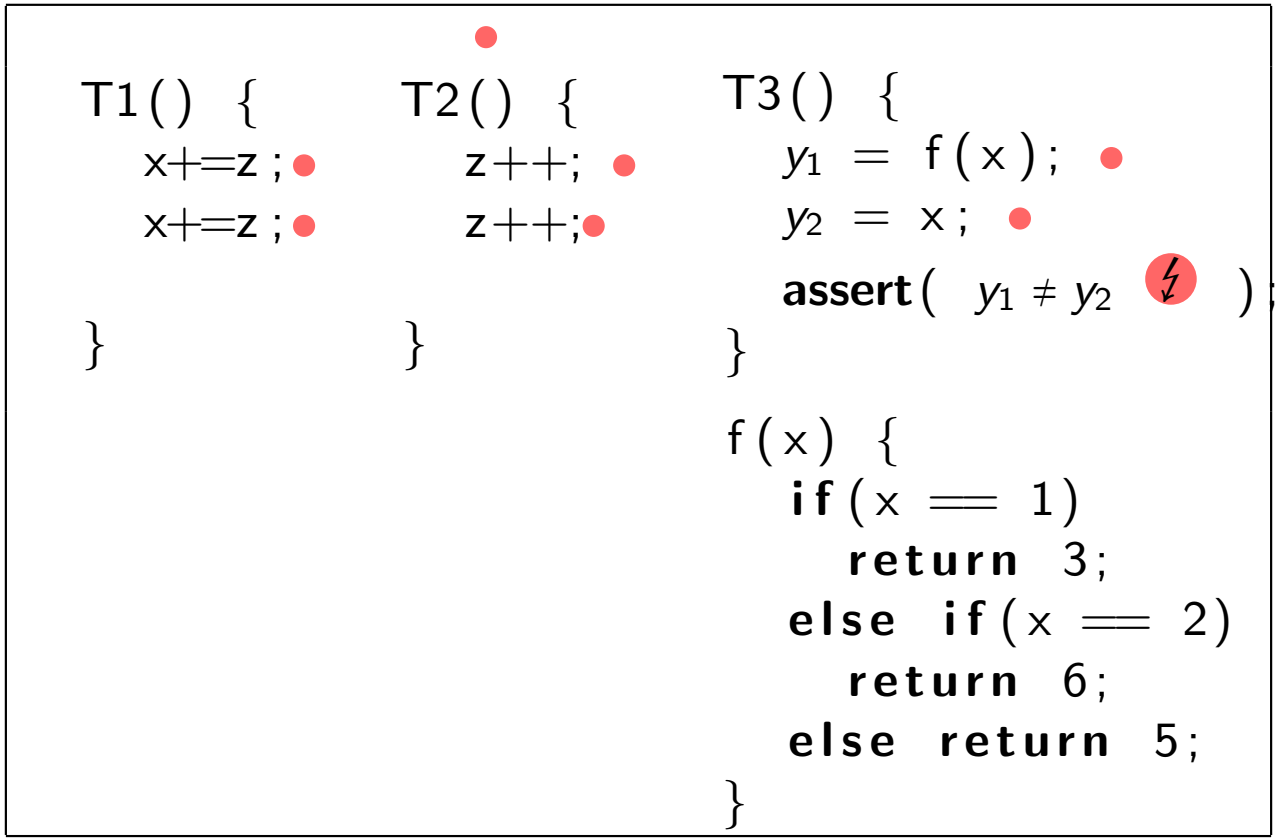


Figure: An interleaving that violates the assertion

Is this program correct?

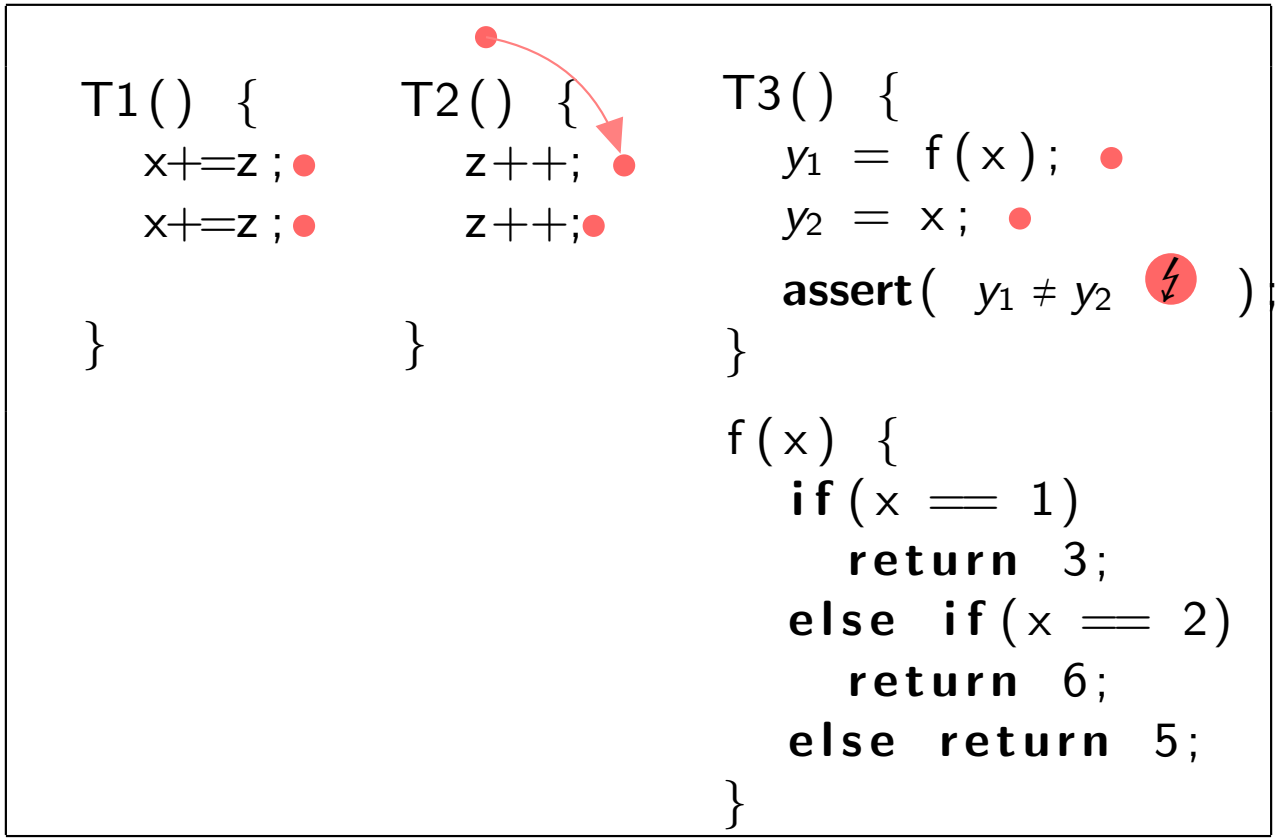


Figure: An interleaving that violates the assertion

Is this program correct?

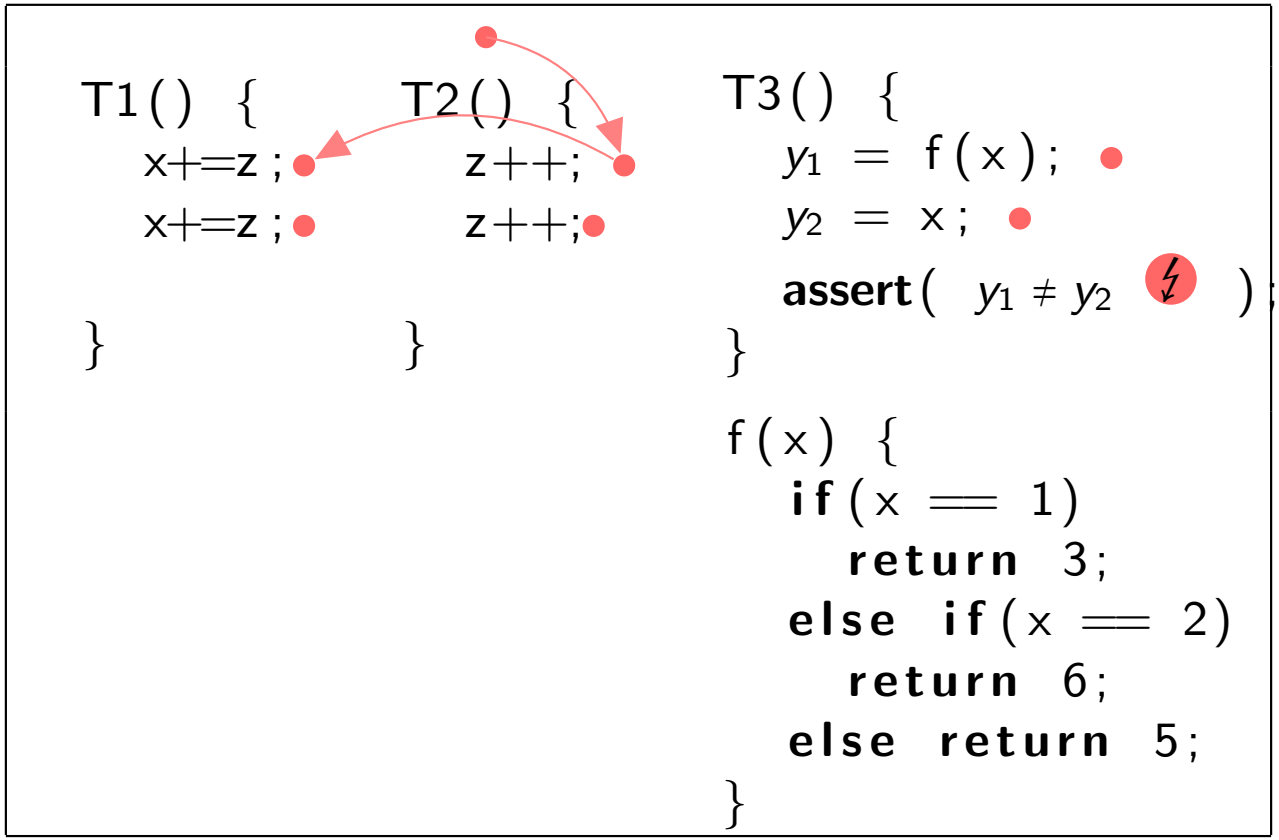


Figure: An interleaving that violates the assertion

Is this program correct?

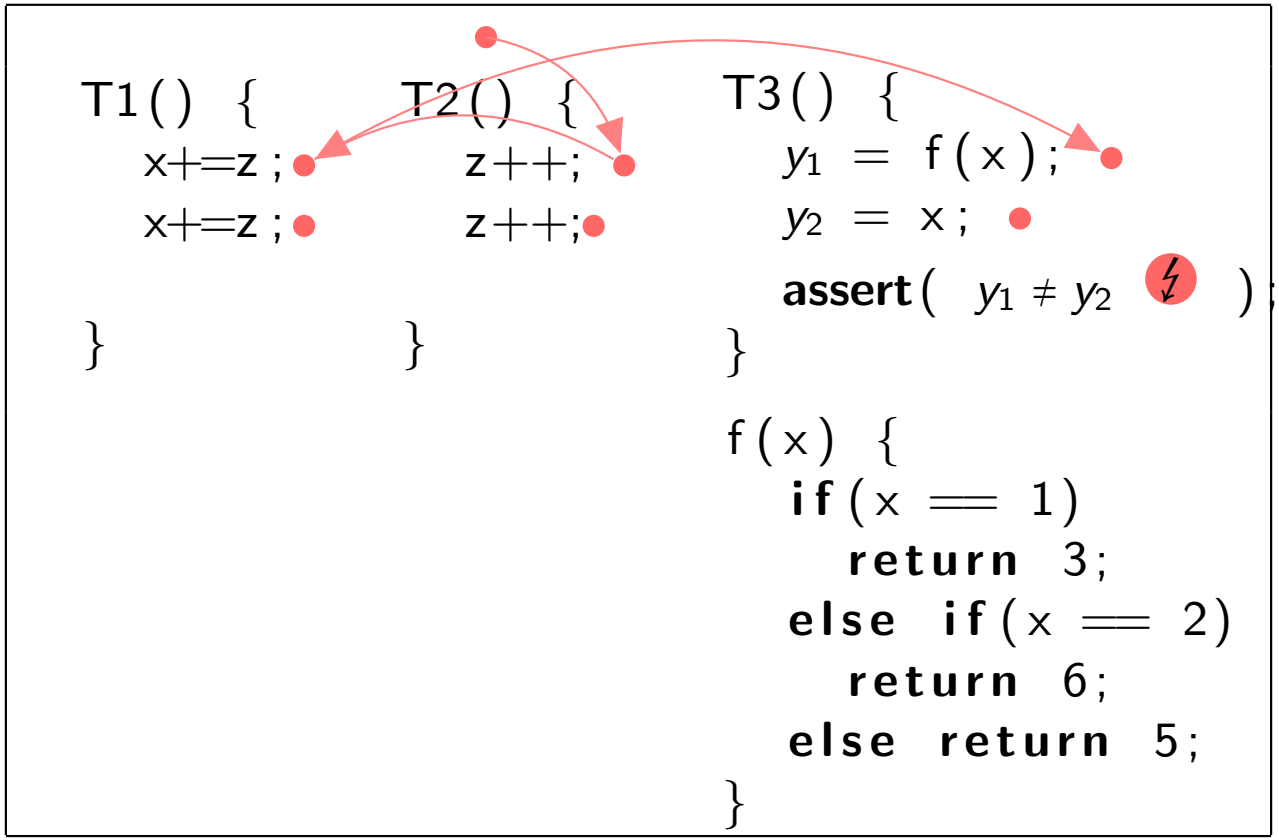


Figure: An interleaving that violates the assertion

Is this program correct?

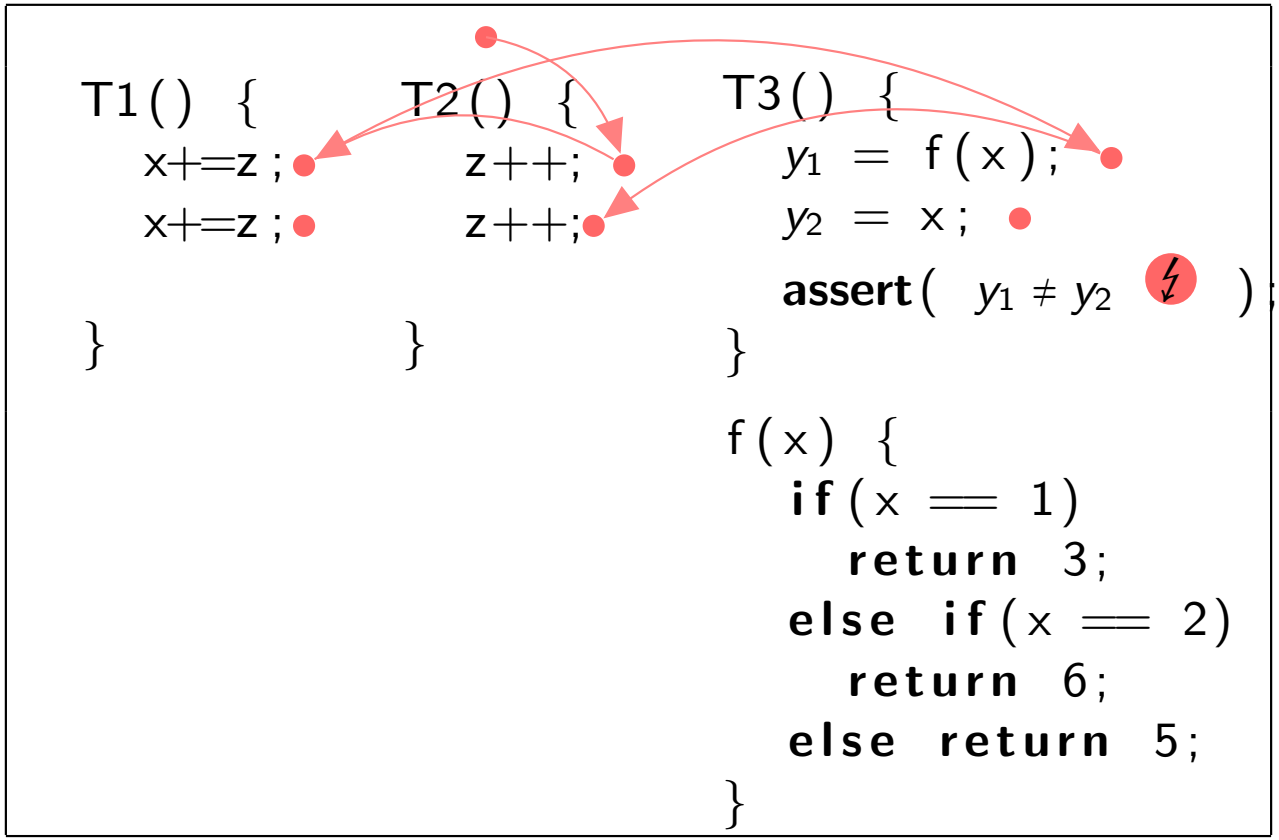


Figure: An interleaving that violates the assertion

Is this program correct?

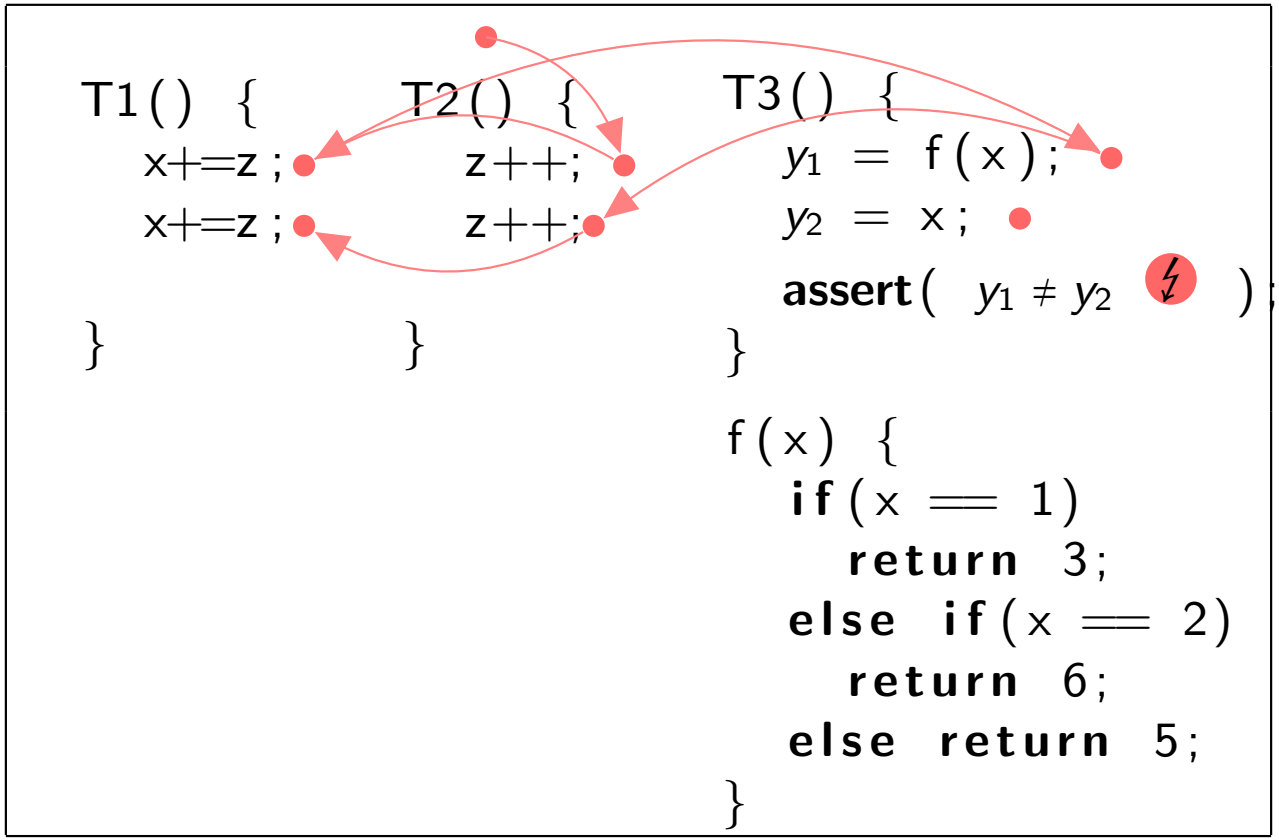


Figure: An interleaving that violates the assertion

Is this program correct?

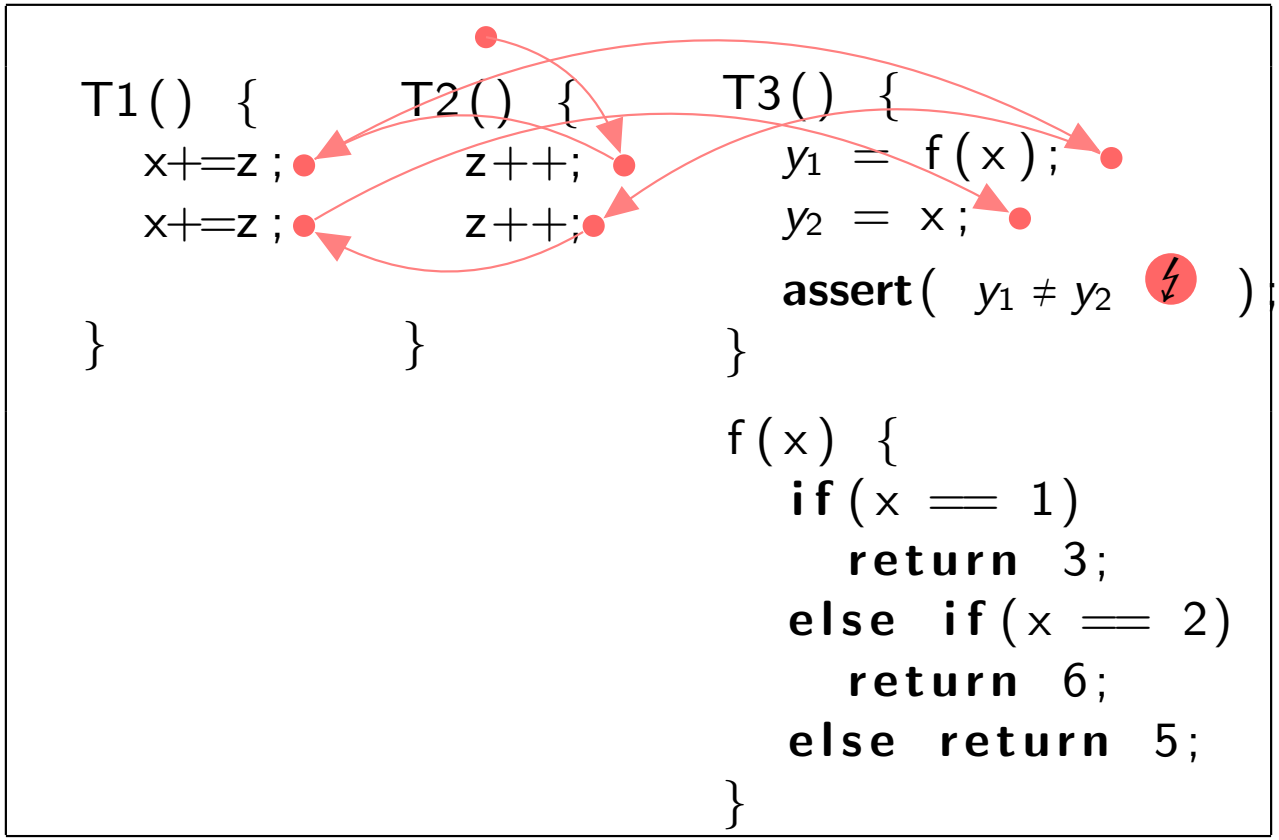


Figure: An interleaving that violates the assertion

Is this program correct?

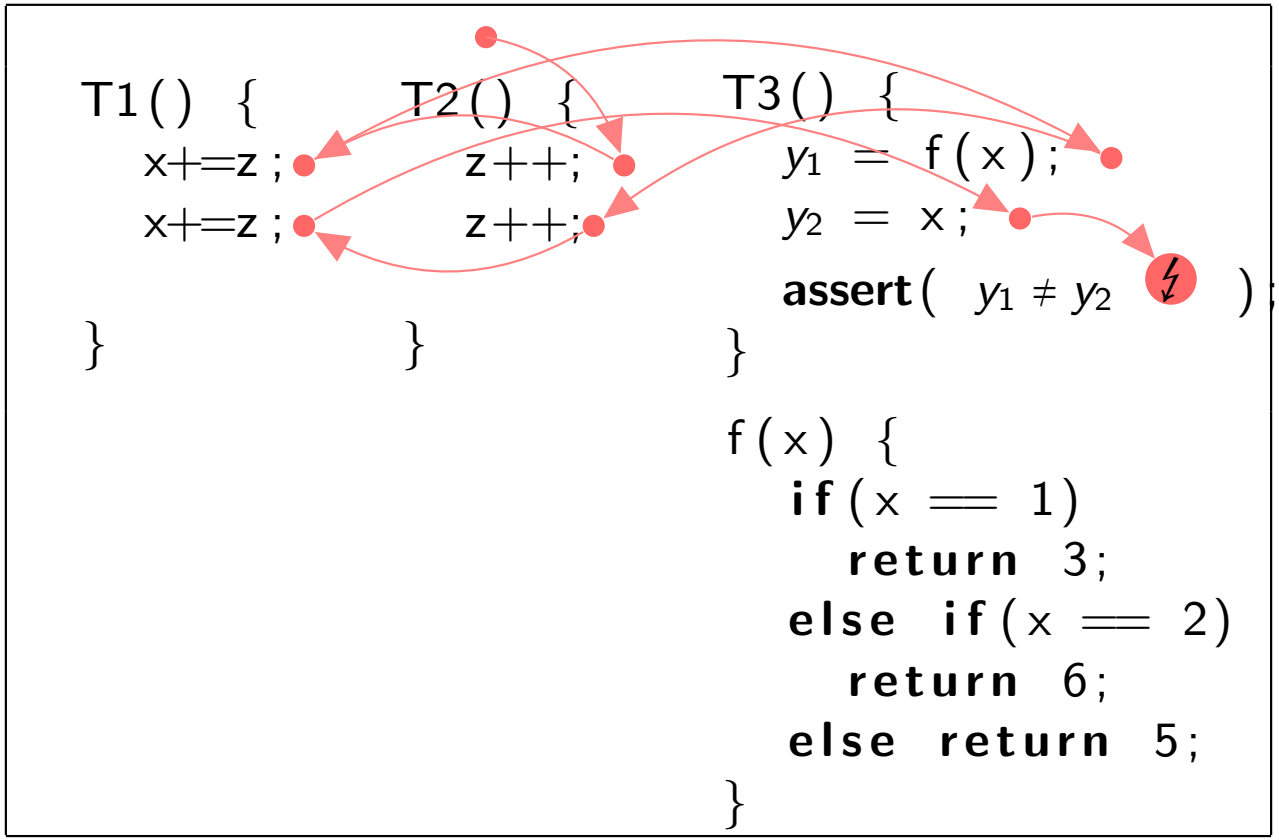


Figure: An interleaving that violates the assertion

Let's try to fix it!

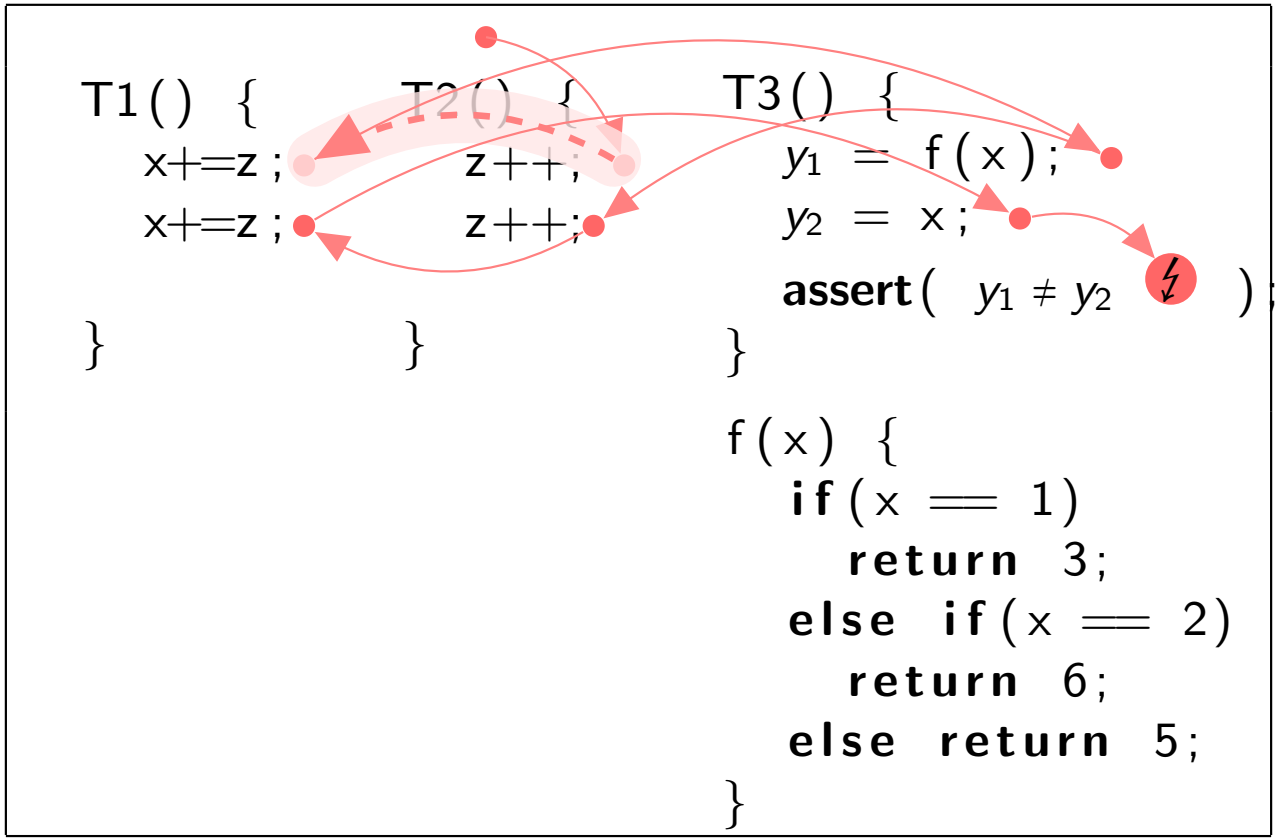


Figure: Avoiding the interleaving: Disabling a context switch

Fixed?

```
T1() {
  x+=z;
  x+=z;

}
```

```
T2() {
  atomically {
    z++;
    z++;
  }
}
```

```
T3() {
  y1 = f(x);
  y2 = x;
  assert( y1 ≠ y2 );
}

f(x) {
  if(x == 1)
    return 3;
  else if(x == 2)
    return 6;
  else return 5;
}
```

Figure: Example: $T1 \parallel T2 \parallel T3$, f atomic

Can we check this automatically?

Safety is a **reachability problem**

As always: **State space explosion**

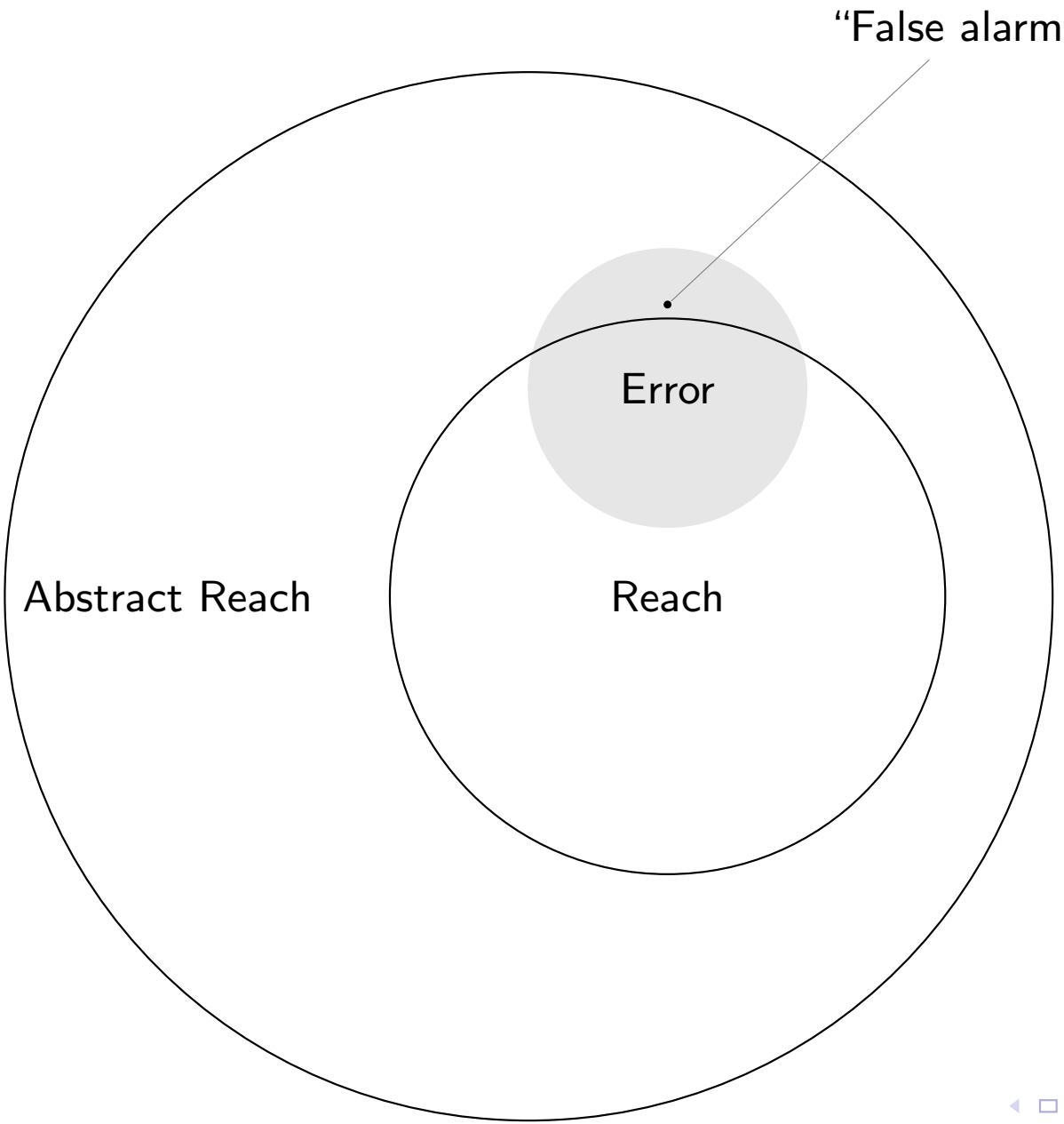
Remedy: **Abstract interpretation**

Guessing game


It’s a bear!

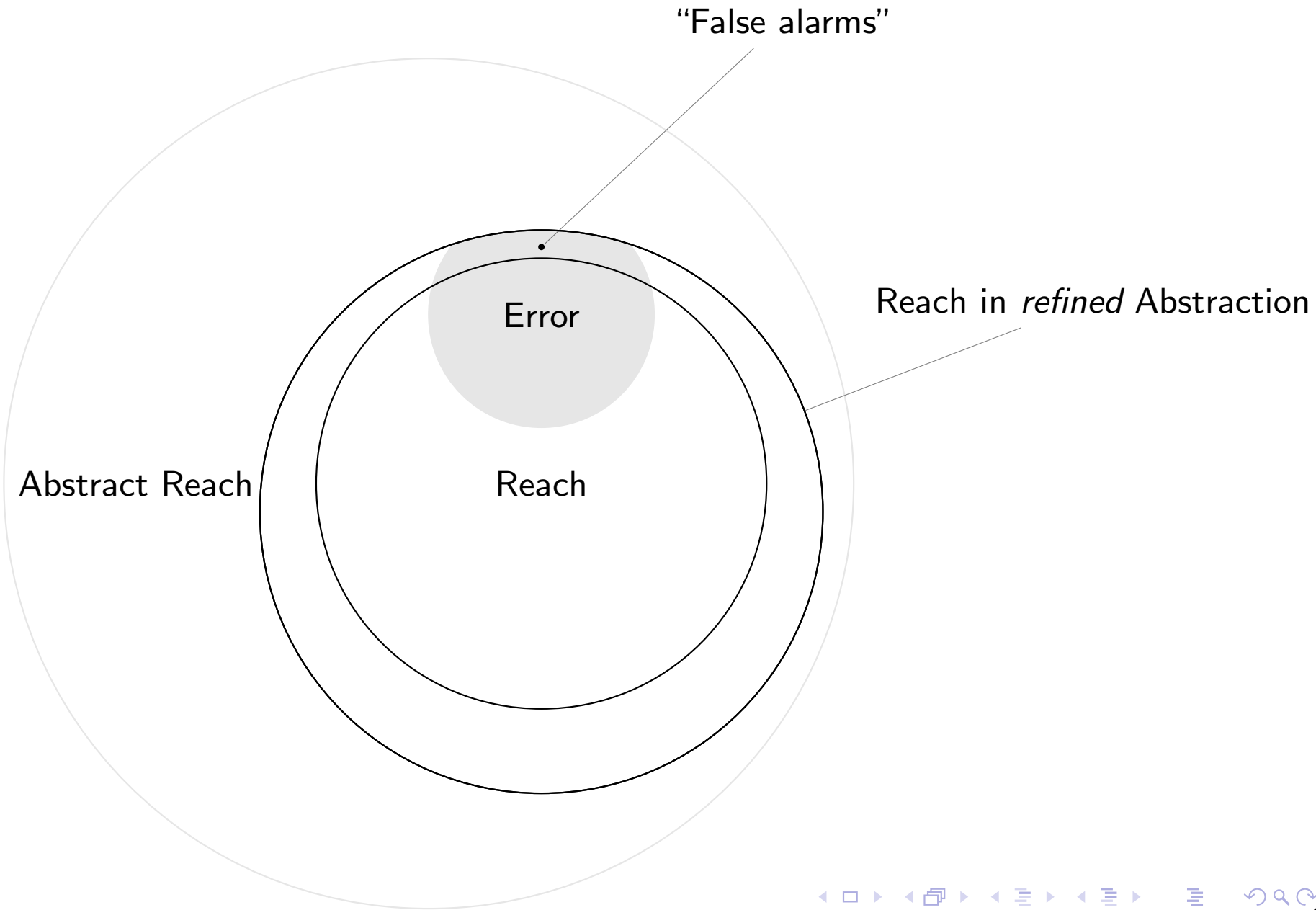
Abstract interpretation: Intuition

Reduce precision, but stay **sound**:



Abstract interpretation: Intuition

Reduce precision, but stay **sound**:



Abstract interpretation: Parity abstraction

$X \subseteq \mathbb{Z}$: set of variable values.

$$\alpha_{\text{parity}}(X \subseteq \mathbb{Z}) = \begin{cases} \bot & \text{if } X = \varnothing \\ 2\mathbb{Z} & \text{if } \varnothing \neq X \subseteq 2\mathbb{Z} \\ 2\mathbb{Z}+1 & \text{if } \varnothing \neq X \subseteq 2\mathbb{Z}+1 \\ \top & \textit{otherwise} \end{cases}$$

State = multiple variables $\rightarrow$ component-wise abstraction.

Fixed?

```
T1() {
  x+=z;
  x+=z;

}
```

```
T2() {
  atomically {
    z++;
    z++;
  }
}
```

```
T3() {
  y1 = f(x);
  y2 = x;
  assert( y1 ≠ y2 );
}

f(x) {
  if(x == 1)
    return 3;
  else if(x == 2)
    return 6;
  else return 5;
}
```

Figure: Example: $T1 \parallel T2 \parallel T3$, f atomic

Fixed?

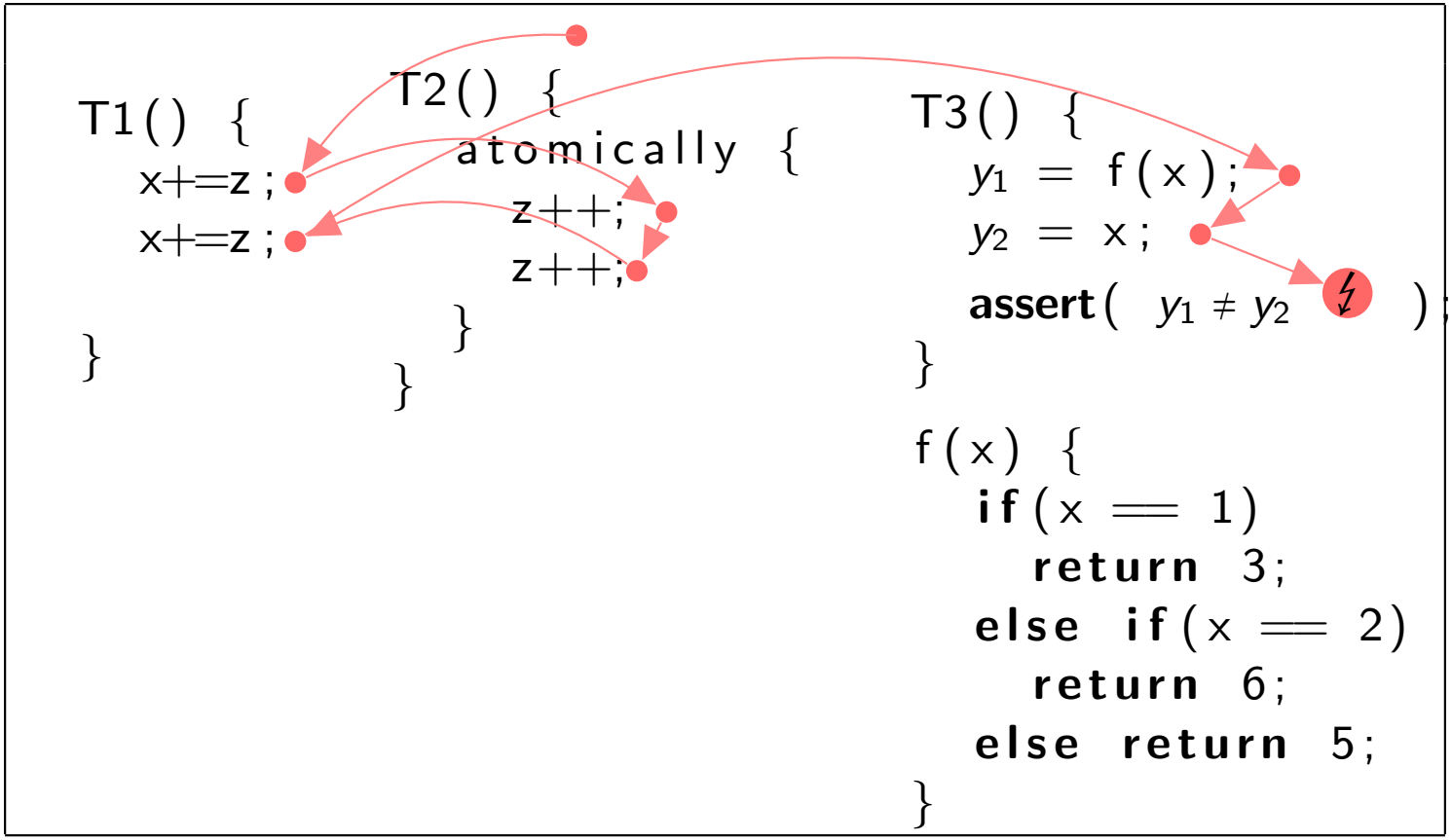


Figure: An interleaving that violates the assertion in the abstraction

Fixed?

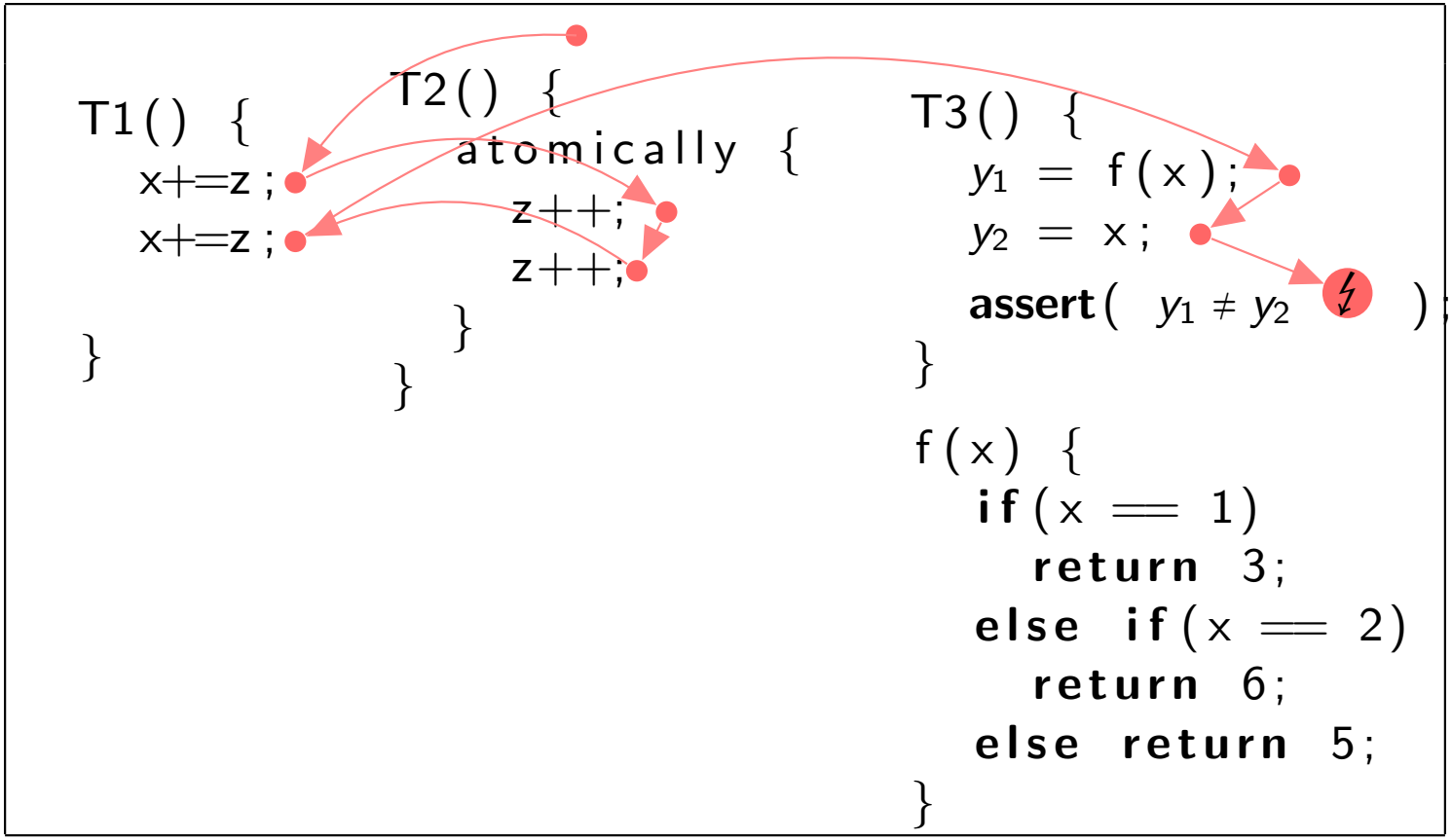


Figure: An interleaving that violates the assertion in the abstraction

False positive!

Abstract interpretation: Interval abstraction

$X \subseteq \mathbb{Z}$: set of variable values.

$$\alpha_{\text{interval}}(X \subseteq \mathbb{Z}) = [\min X, \max X] \subseteq \mathbb{Z}$$

Q: More precise than parity?

Abstract interpretation: Interval abstraction

$X \subseteq \mathbb{Z}$: set of variable values.

$$\alpha_{\text{interval}}(X \subseteq \mathbb{Z}) = [\min X, \max X] \subseteq \mathbb{Z}$$

Q: More precise than parity?
A: Yes!

Abstract interpretation: Interval abstraction

$X \subseteq \mathbb{Z}$: set of variable values.

$$\alpha_{\text{interval}}(X \subseteq \mathbb{Z}) = [\min X, \max X] \subseteq \mathbb{Z}$$

Q: More precise than parity?
A: Yes! *Refine* our (parity) abstraction to interval.

Fixed!

```
T1() {
    x+=z;
    x+=z;
}
```

```
T2() {
  atomically {
    z++;
    z++;
  }
}
```

```
T3() {
  y1 = f(x);
  y2 = x;
  assert( y1 ≠ y2 );
}

f(x) {
  if(x == 1)
    return 3;
  else if(x == 2)
    return 6;
  else return 5;
}
```

Figure: Example: $T1 \parallel T2 \parallel T3$, f atomic: Safe under interval abstraction

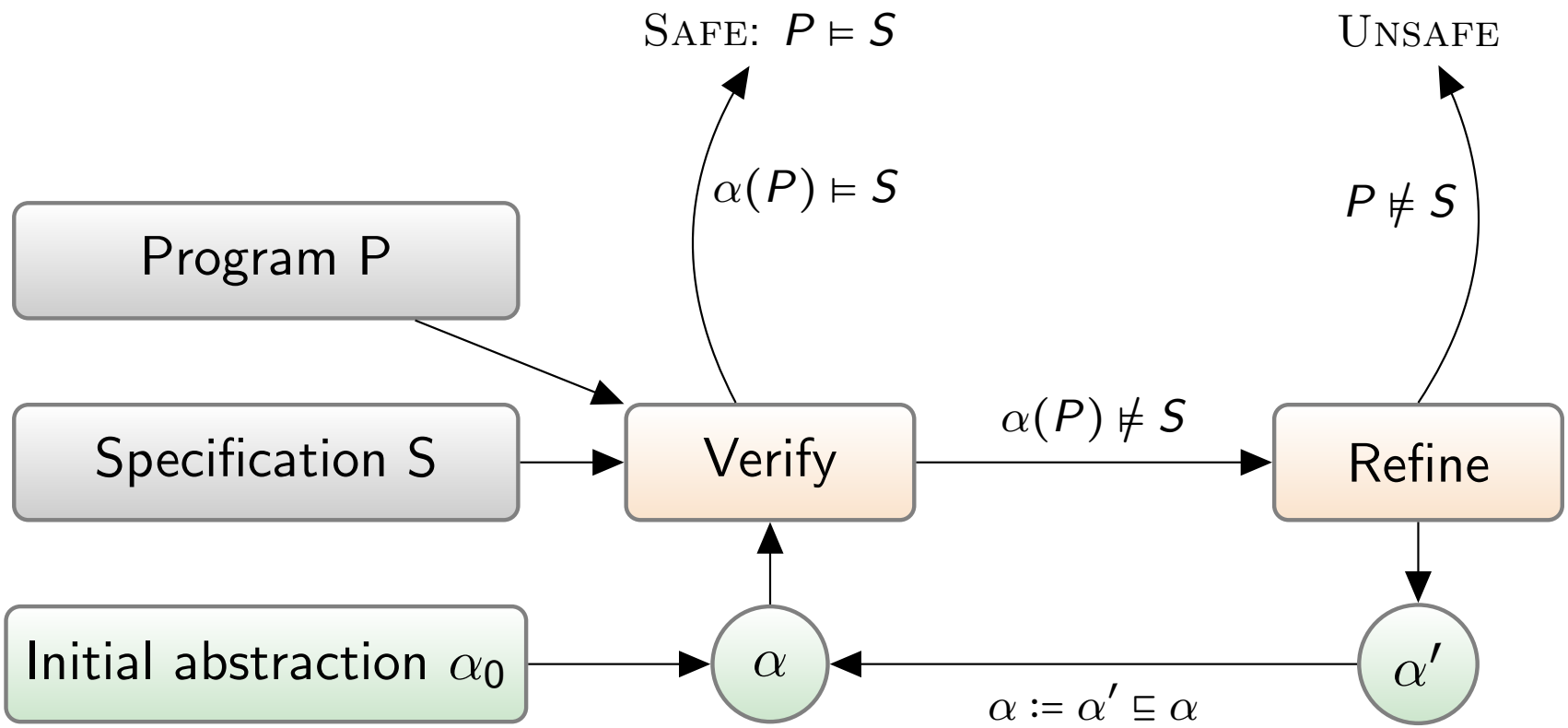
What we just did

Check safety (in abstraction)

Avoid the interleaving or **refine** the abstraction

Repeat

Verifier – Checking Safety



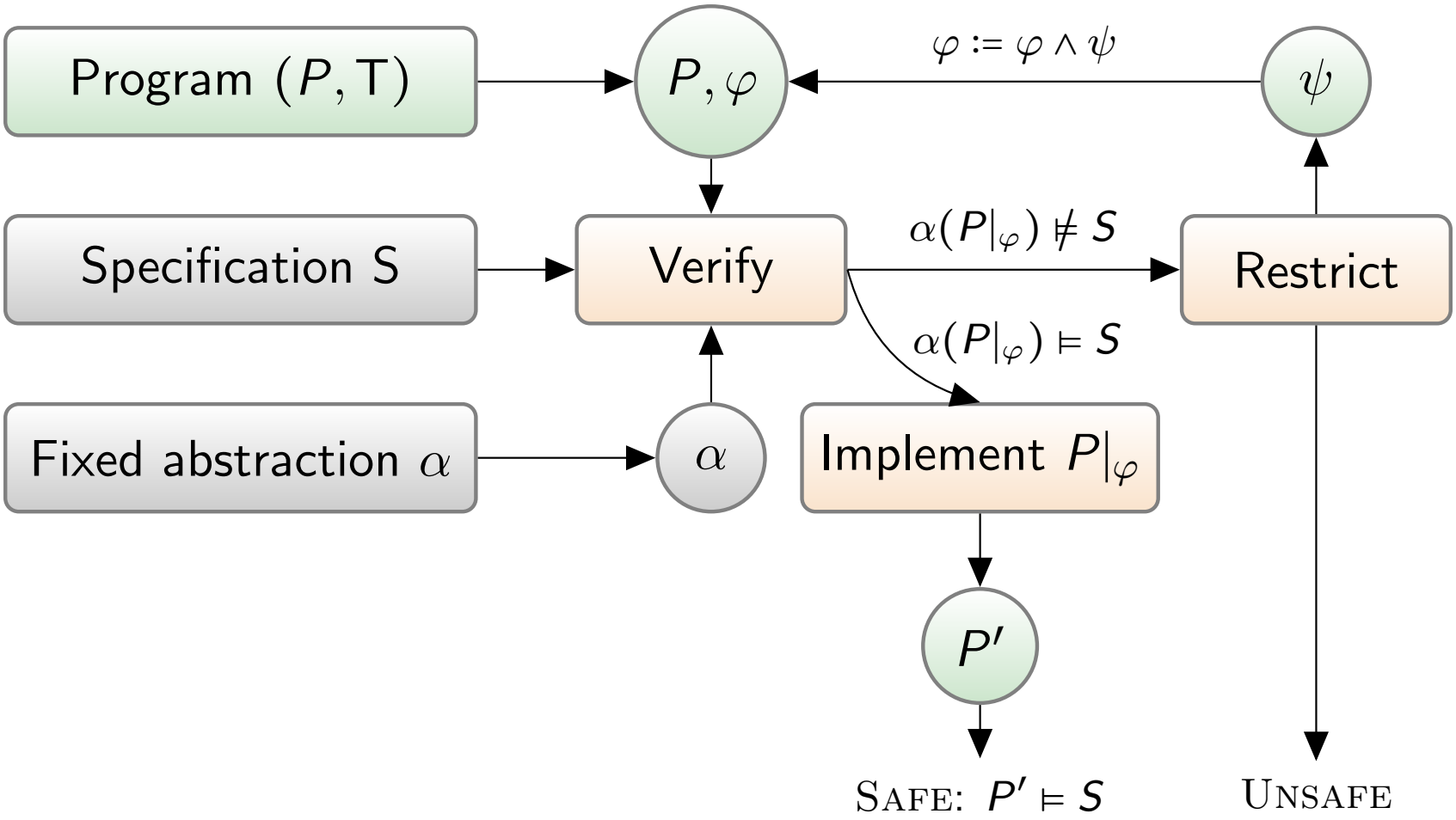
What we just did

Check safety (in abstraction)

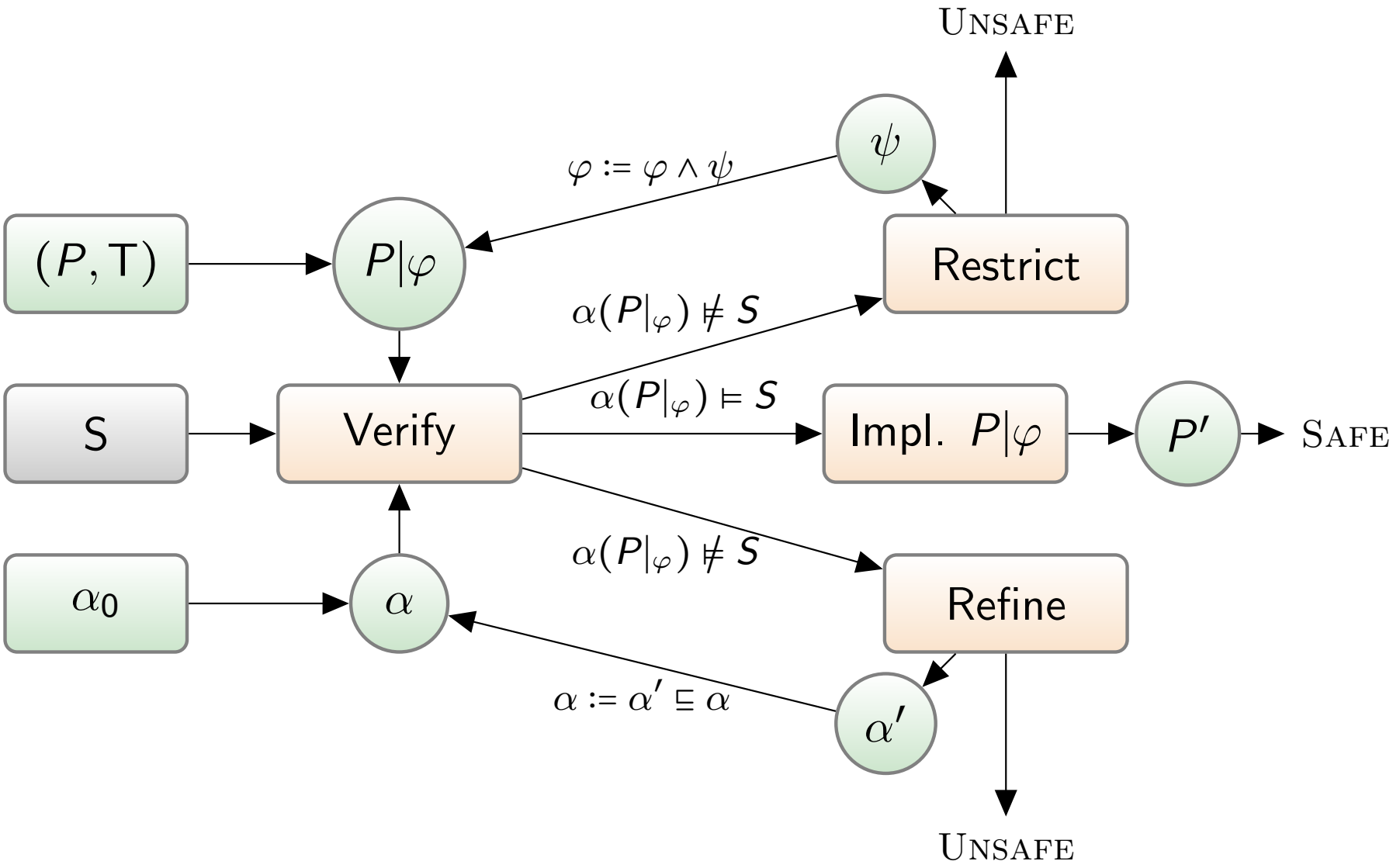
Avoid the interleaving or **refine** the abstraction

Repeat

(Synchronization) Synthesizer – Avoiding Bad Traces



AGS: Verifier+Synthesizer



AGS pseudocode

Input: Program P, Specification S, Abstraction function α.
Output: Program P' satisfying S.

procedure $\mathrm{AGS}(P, S, \alpha)$
    $\varphi := \mathsf{T}$
    repeat
        if $\exists \pi \in \mathsf{AbstractTraces}(P|_\varphi, \alpha) : \pi \notin S$ **then**
            if $\mathsf{ShouldAvoid}(\pi, \alpha) \vee \neg \mathsf{CanRefine}(\alpha, \pi)$ **then**
                if $\mathsf{CanAvoid}(\pi)$ **then**
                    $\varphi := \varphi \wedge \mathsf{Avoid}(\pi)$
                else abort
            else
                $\alpha := \mathsf{Refine}(\alpha, \pi)$
        else
            return $\mathsf{Implement}(P|_\varphi)$

AGS pseudocode

Input: Program P, Specification S, Abstraction function α.
Output: Program P' satisfying S.

procedure $\mathrm{AGS}(P, S, \alpha)$
    $\varphi := \mathsf{T}$
    repeat
        if $\exists \pi \in \mathsf{AbstractTraces}(P|_\varphi, \alpha) : \pi \notin S$ **then**
            if $\mathsf{ShouldAvoid}(\pi, \alpha) \vee \neg \mathsf{CanRefine}(\alpha, \pi)$ **then**
                if $\mathsf{CanAvoid}(\pi)$ **then**
                    $\varphi := \varphi \wedge \mathsf{Avoid}(\pi)$
                else abort
            else
                $\alpha := \mathsf{Refine}(\alpha, \pi)$
        else
            return $\mathsf{Implement}(P|_\varphi)$

AGS pseudocode

procedure $\textsc{Avoid}(\pi = \pi_1 \dots \pi_n)$

$$\rho := \bigvee_{i,j \in [n] \wedge i<j} \left\{ [\pi_i, \pi_j] \mid \wedge \exists t : \mathrm{T}(\pi_i) = \mathrm{T}(\pi_j) = t \wedge \forall l \in]i, j[: \mathrm{T}(\pi_l) \neq t \right\}$$

return ρ

A context switch in π:

Scheduling a thread t $(T(\pi_i) = t)$

executing any other thread(s) $(T(\pi_{i+1}), \dots, T(\pi_{j-1}) \neq t)$

and back again $(T(\pi_j) = t)$.

Disjunction $\vee$: "disable at least one of these".

Avoid vs. What we did

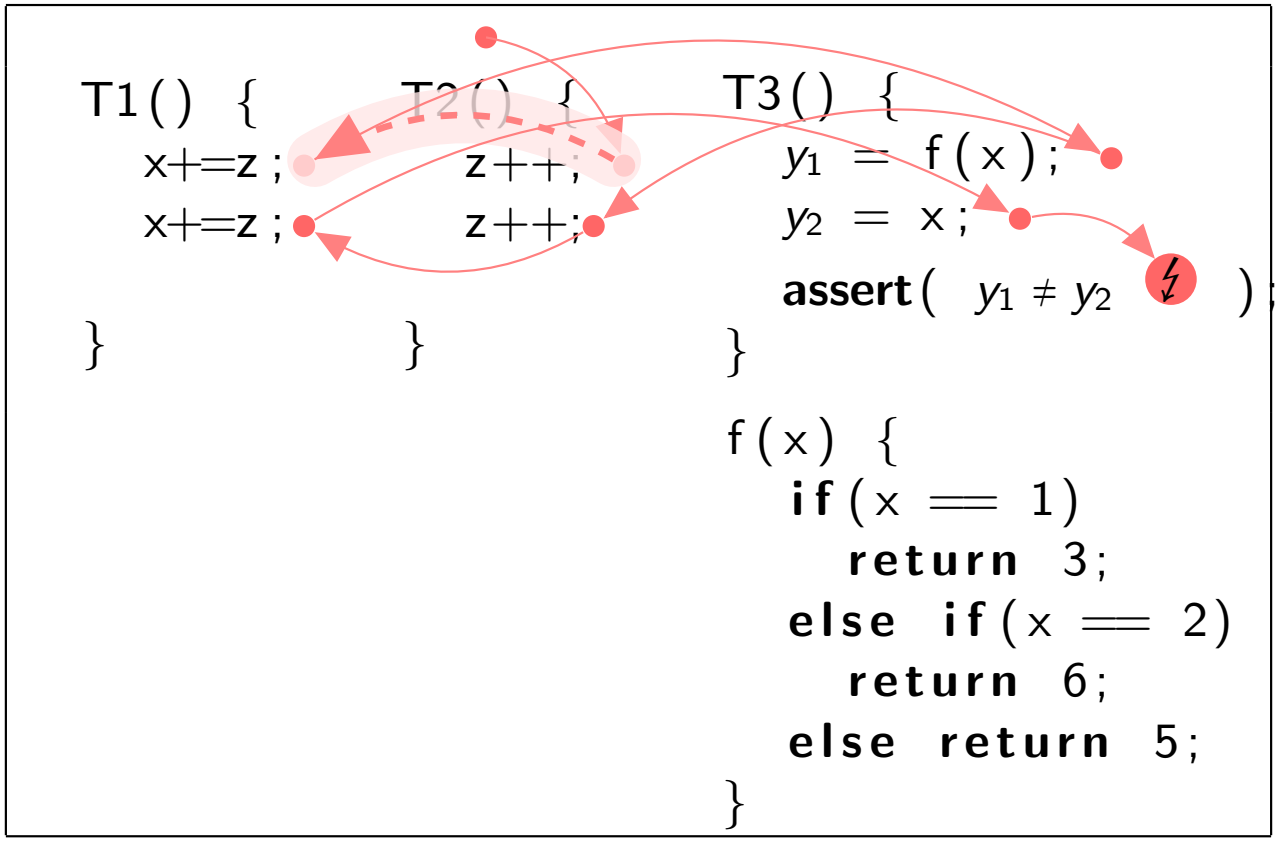


Figure: Avoiding the interleaving: Disabling a context switch

$$\pi = T2_1\ T1_1\ T3_1\ T2_2\ T1_2\ T3_2\ T3_3$$

Avoid - Example

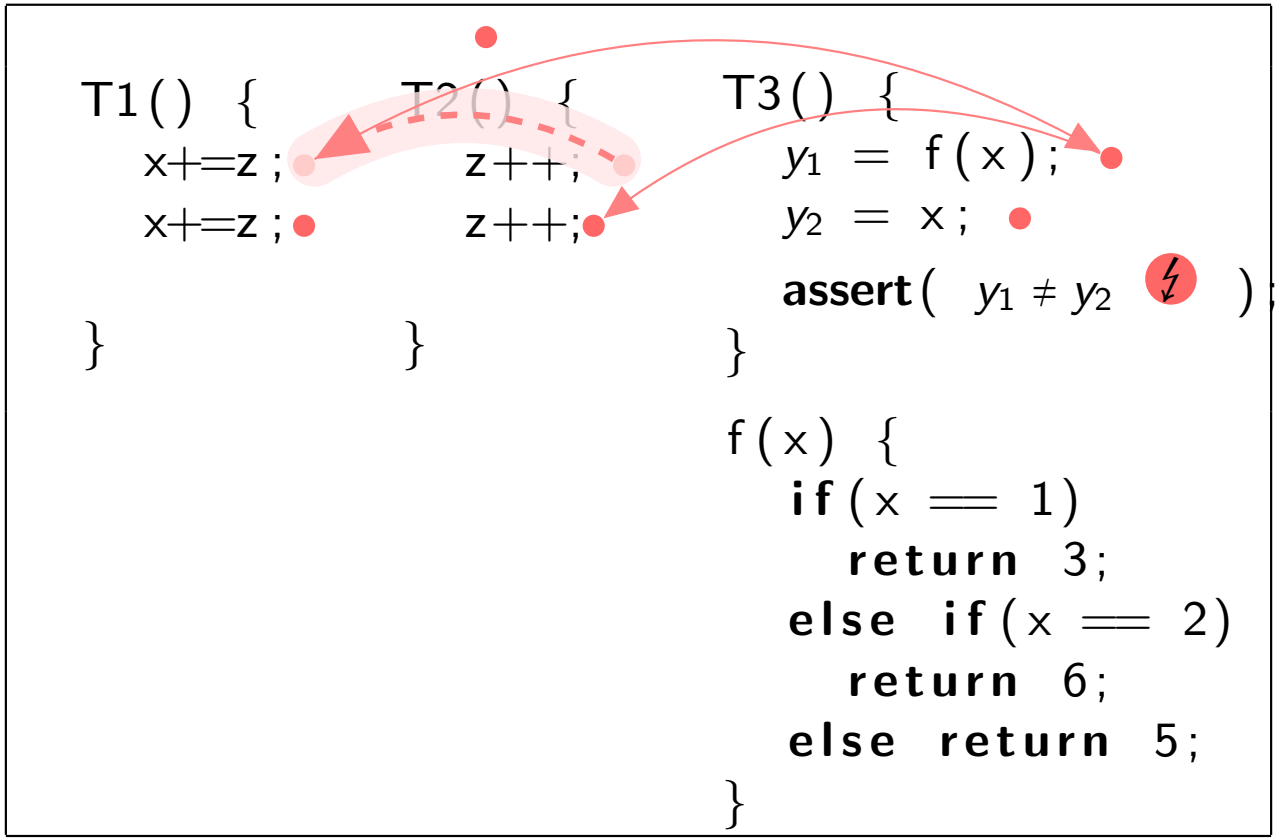


Figure: Avoiding the interleaving: Disabling at least one context switch

$$\pi = T2_1\ T1_1\ T3_1\ T2_2\ T1_2\ T3_2\ T3_3$$

Avoid - Example

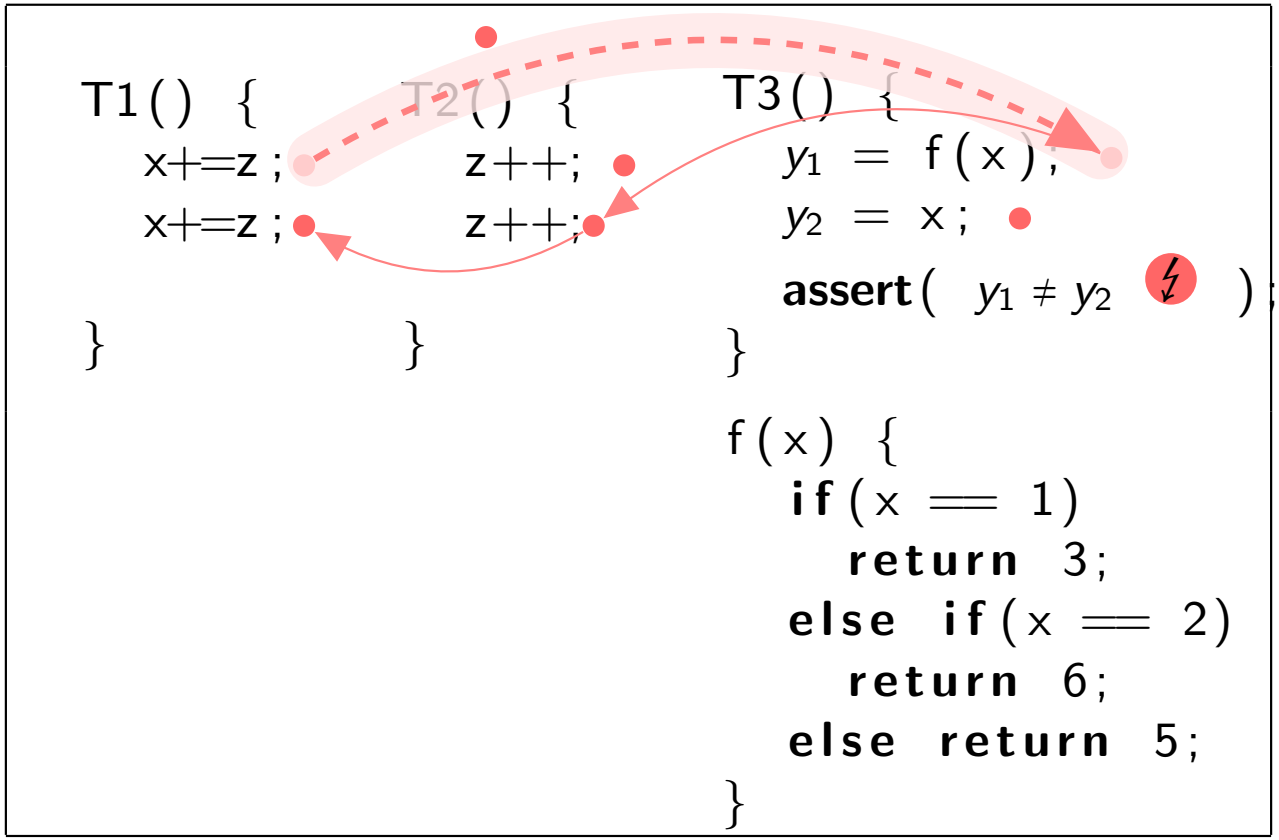


Figure: Avoiding the interleaving: Disabling at least one context switch

$$\pi = T2_1\ T1_1\ T3_1\ T2_2\ T1_2\ T3_2\ T3_3$$

Avoid - Example

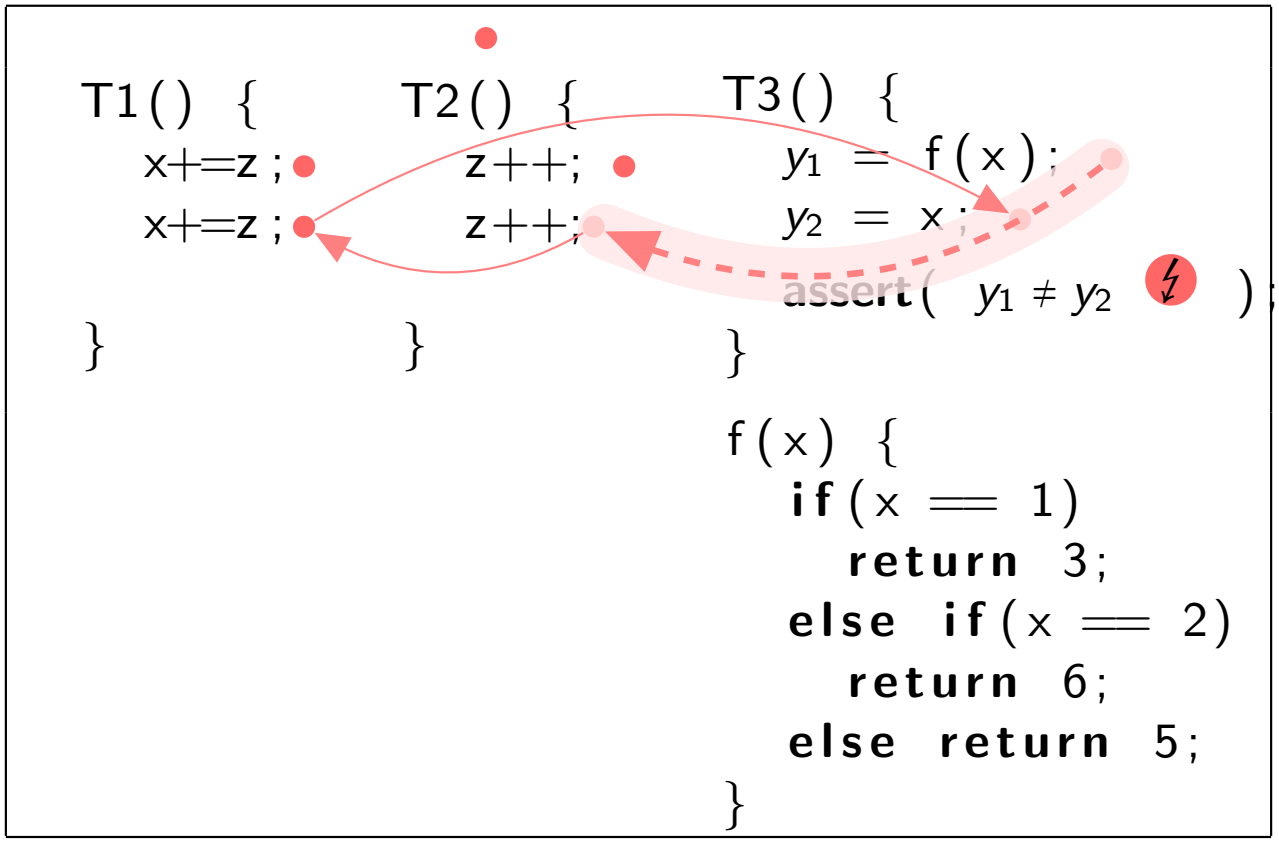


Figure: Avoiding the interleaving: Disabling at least one context switch

$$\pi = T2_1\ T1_1\ T3_1\ T2_2\ T1_2\ T3_2\ T3_3$$

Avoid - Example

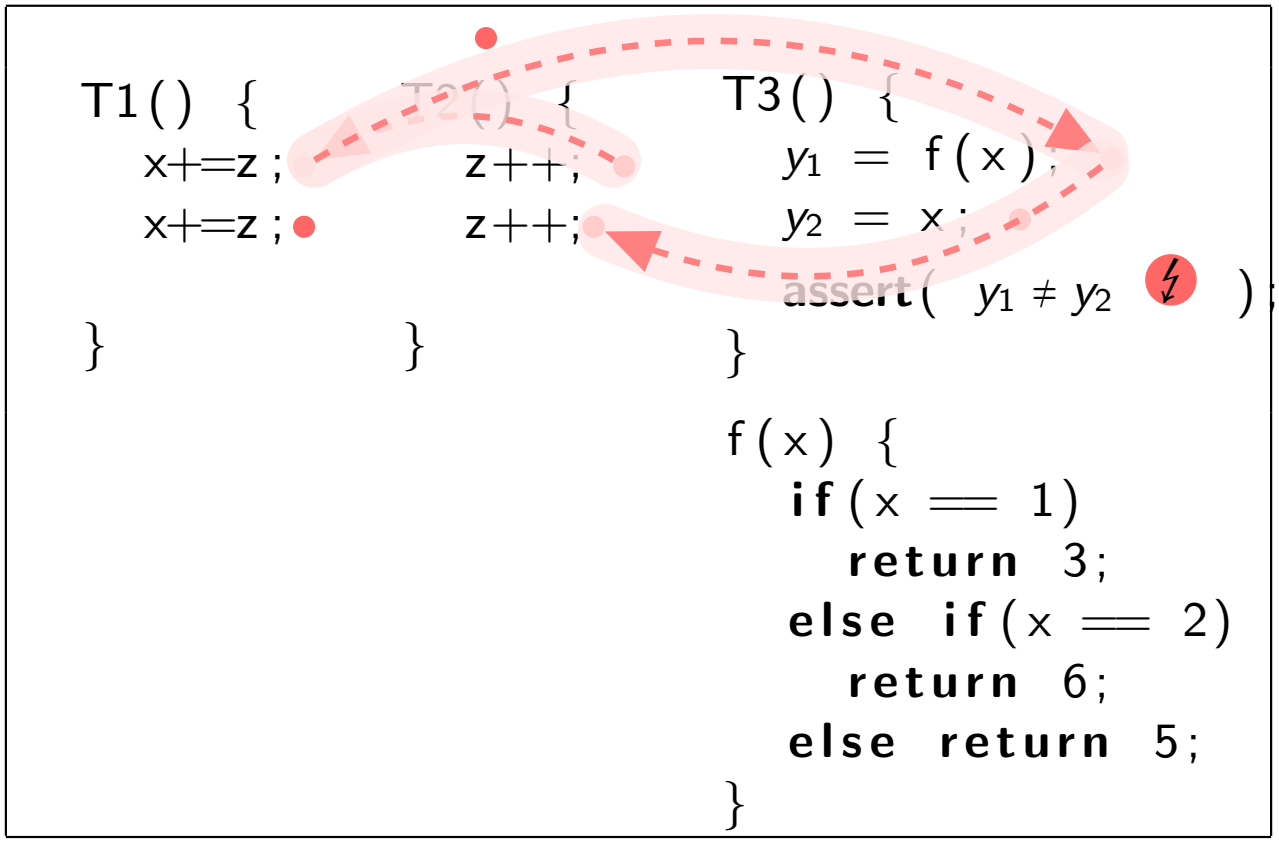


Figure: Avoiding the interleaving: Disabling at least one context switch

$$\text{Avoid}(\pi) = [T2_1, T2_2] \vee [T1_1, T1_2] \vee [T3_1, T3_2]$$

Experiments – Concurrent Defragmentation

```
Barrier = F1 = F2 = 0;
Region = 2;
Defragment() {
/* Pick a Region */
L1: i = Region;
L2: Region = i - 1;
L3: empty = i - 2;
L4: if (i >= N) goto L14;
/* has free entry? */
L5: b = Pages[i];
L6: if (!b && empty <= 0)
L7: empty = i;
/* Copy Entry */
L8: if (b && empty > 0) {
L9: Pages[empty] = true;
L10: empty += 2;
L11: Pages[i] = false;
}
L12: i += 2;
L13: goto L4;
/* Barrier Synch */
L14: Barrier += 1; F1 = 0;
L15: if (F1 == 1)
goto L16;
if (Barrier == 2) {
Barrier = 0; F2 = 1;
Region = 2;
goto L16;
}
goto L15;
L16: goto L1;
}
Update() {
/* Pick a Region */
L1: j = Region;
L2: Region = j - 1;
L3: b = Pages[j];
/* Update the Page */
L4: if (!b)
Pages[j] = true;
/* Barrier Sync */
L5: Barrier += 1; F2 = 0;
L6: if (F2 == 1)
goto L7;
if (Barrier == 2) {
Barrier = 0; F1 = 1;
Region = 2;
goto L7;
}
goto L6;
L7:
}
main() {
Defragment() || Update();
}
```

Figure: Defragment cleans while Update allocates. Spec: Disjoint access

Experiments Summary

Concurrent defragmentation, Double buffering, 3D Grid computation

All experiments ran in under 10 minutes

Non-trivial programs

AGS is useful.

Limitations – AGS and Deadlocks

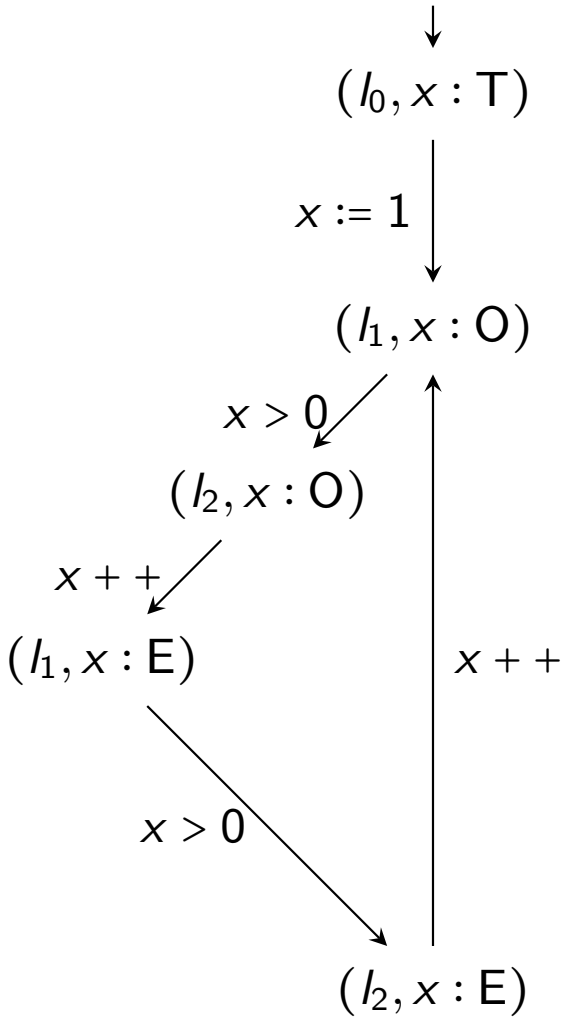
Let's apply AGS to $S \parallel T$:

```
S  {down(b);
    [up(c);
    [down(b);  }
```

```
T  { [up(b);
     [down(c);
     [up(b);
     [assert(b == 1);  }
```

Figure: Example problematic program

Abstract interpretation: Example



l_0 : x = 1;
l_1 : **while** (x>0)
l_2 : x++;