Objectives

- Understand the concept of thread interleaving in concurrent systems
- Understand how a system’s schedules can be viewed as a computation tree where a path through the tree corresponds to a particular schedule
- Be able to draw the computation tree for any simple BIR-Lite system
Outline

- The SumToN Example
  - use this simple BIR-Lite system to explain the concepts of schedule and computation tree
- Computation Trees

SumToN

```system SumToN { 
  const PARAM { N = 1 };  
  typealias byte int wrap (0,255);  
  byte x;  
  byte t1;  
  byte t2;  
  thread Thread1() {  
    loc loc0:  
      when x != 0 do { t1 := x; }  
      goto loc1;  
    loc loc1:  
      do { t2 := x; }  
      goto loc2;  
    loc loc2:  
      do { x := t1 + t2; }  
      goto loc0;  
  }  
  thread Thread2() {  
    loc loc0:  
      when x != 0 do { t1 := x; }  
      goto loc1;  
    loc loc1:  
      do { t2 := x; }  
      goto loc2;  
    loc loc2:  
      do { x := t1 + t2; }  
      goto loc0;  
  }  
  thread Thread0() {  
    loc loc0:  
      do { x := 1; }  
      goto loc1;  
    loc loc1:  
      do { assert (x != PARAM.N); }  
      return;  
  }  
} 
```

declare a namespace PARAM with a constant N so that we can easily modify N's value.
system SumToN {
  const PARAM { N = 1; }
  typealias byte int wrap (0,255);
  byte x;
  byte t1;
  byte t2;

  thread Thread1() {
    loc loc0:
      when x != 0 do { t1 := x; }
      goto loc1;
    loc loc1:
      do { t2 := x; }
      goto loc2;
    loc loc2:
      do { x := t1 + t2; }
      goto loc0;
  }

  thread Thread2() {
    loc loc0:
      when x != 0 do { t1 := x; }
      goto loc1;
    loc loc1:
      do { t2 := x; }
      goto loc2;
    loc loc2:
      do { x := t1 + t2; }
      goto loc0;
  }

  thread Thread0() {
    loc loc0:
      do {
        x := 1;
      } goto loc1;
    loc loc1:
      do { assert (x != PARAM.N); }
      return;
  }
}

declare a 'byte' to be an integer with range 0..255 that will 'wrap around' when operated on.

declare three byte-sized variables
system SumToN {
    const PARAM { N = 1 };  
    typealias byte int wrap (0,255);  
    byte x;  
    byte t1;  
    byte t2;  

    thread Thread1() {
        loc loc0:  
            when x != 0 do { t1 := x; }  
            goto loc1;  
        loc loc1:  
            do { t2 := x; }  
            goto loc2;  
        loc loc2:  
            do { x := t1 + t2; }  
            goto loc0;  
    }
}

system SumToN {
    const PARAM { N = 1 };  
    typealias byte int wrap (0,255);  
    byte x;  
    byte t1;  
    byte t2;  

    thread Thread2() {
        loc loc0:  
            when x != 0 do { t1 := x; }  
            goto loc1;  
        loc loc1:  
            do { t2 := x; }  
            goto loc2;  
        loc loc2:  
            do { x := t1 + t2; }  
            goto loc0;  
    }
}

thread Thread0() {
    loc loc0:  
        do {  
            x := 1;  
        } goto loc1;  
    loc loc1:  
        do { assert (x != PARAM.N); }  
        return;  
}

Each thread reads the value of x in t1, then t2, then sums t1 and t2 to get a new value for x.

The "main" thread initializes x, and then asserts that x is not equal to the value of N.
SumToN

system SumToN {
    const PARAM { N = 1 };    
type alias byte int wrap (0,255);
    byte x;
    byte t1;
    byte t2;
    thread Thread1() {
        loc loc0:
            when x != 0 do { t1 := x; }
            goto loc1;
        loc loc1:
            do { t2 := x; }
            goto loc2;
        loc loc2:
            do { x := t1 + t2; }
            goto loc0;
    }
    thread Thread2() {
        loc loc0:
            when x != 0 do { t1 := x; }
            goto loc1;
        loc loc1:
            do { t2 := x; }
            goto loc2;
        loc loc2:
            do { x := t1 + t2; }
            goto loc0;
    }
    thread Thread0() {
        loc loc0:
            do {
                x := 1;
            } goto loc1;
        loc loc1:
            do { assert (x != PARAM.N); }
            return;
    }
}

Note: Arbitrary interleaving can occur between the execution of these two transitions

Assessment

Can the assertion in the SumToN example be violated?

- Answering this question requires us to reason about possible schedules (i.e., orderings of instruction execution)

- Let’s try to find schedules that cause the assertion to be violated for various values of N...
SumToN Assertion Violation

Violating schedule for $N = 1$

(initial values)

$0:0 \rightarrow [0, 0, 0, x = 0, t1 = 0, t2 = 0]$

$0:1 \rightarrow [1, 0, 0, x = 1, t1 = 0, t2 = 0]$

$0:1 \rightarrow [2, 0, 0, x = 1, t1 = 0, t2 = 0]$  \[violation\]

...that was easy!

---

SumToN Assertion Violation

Violating schedule for $N = 2$

(initial values)

$0:0 \rightarrow [0, 0, 0, x = 0, t1 = 0, t2 = 0]$

$1:0 \rightarrow [1, 0, 0, x = 1, t1 = 0, t2 = 0]$

$1:1 \rightarrow [1, 1, 0, x = 1, t1 = 1, t2 = 0]$

$1:2 \rightarrow [1, 0, 0, x = 2, t1 = 1, t2 = 1]$

$0:1 \rightarrow [1, 0, 0, x = 2, t1 = 1, t2 = 1]$  \[violation\]
SumToN Assertion Violation

Violating schedule for $N = 2$

(Initial values)

0:0 $\rightarrow$ [0, 0, 0, x = 0, t1 = 0, t2 = 0]

2:0 $\rightarrow$ [1, 0, 1, x = 1, t1 = 1, t2 = 0]

2:1 $\rightarrow$ [1, 0, 2, x = 1, t1 = 1, t2 = 1]

2:2 $\rightarrow$ [1, 0, 0, x = 2, t1 = 1, t2 = 1]

0:1 $\rightarrow$ [1, 0, 0, x = 2, t1 = 1, t2 = 1]

---

Yet Another

SumToN Assertion Violation

Violating schedule for $N = 2$

(Initial values)

0:0 $\rightarrow$ [0, 0, 0, x = 0, t1 = 0, t2 = 0]

2:0 $\rightarrow$ [1, 0, 1, x = 1, t1 = 1, t2 = 0]

2:1 $\rightarrow$ [1, 1, 1, x = 1, t1 = 1, t2 = 0]

2:2 $\rightarrow$ [1, 1, 0, x = 2, t1 = 1, t2 = 1]

0:1 $\rightarrow$ [1, 1, 0, x = 2, t1 = 1, t2 = 1]

---

Another

Another
We can think of the possible schedules (execution traces) as forming a computation tree...

First example trace (schedule)
Computation Tree

- We can think of the possible schedules (execution traces) as forming a computation tree...

Computation Tree

- We can think of the possible schedules (execution traces) as forming a computation tree...

Typo in figure

Typo in figure
Computation Tree

- We can think of the possible schedules (execution traces) as forming a computation tree...

For You To Do...

- [book exercise] Consider a modification to the SumToN system where N is set to 3. Is there a schedule that causes the assertion to be violated in this modified system? If so, give the schedule.

- [book exercise] For the SumToN system, does there exist a value of N from 0 and 255 such that assertion cannot be violated?

- Draw a computation tree five levels deep for the Dining Philosophers of example Figure ?? from the Book (i.e., the computation tree should represent the first five instructions of every possible schedule).
Summary

- The actions/transitions of a concurrent system can be interleaved in arbitrary ways.
- The implementation's scheduler is responsible for choosing a particular ordering of transitions.
- When verifying systems, one often abstracts away from the particular scheduling strategy and considers the choices between enabled transitions to be made non-deterministically.
- The collection of execution traces of a system can be visualized as a computation tree.
- A path through the computation tree corresponds to a particular execution (i.e., schedule).