Thread Synchronization: Too Much Milk

Safety and Liveness

Properties defined over an execution of a program

- Safety: "nothing bad happens"
  - holds in every finite execution prefix
    - Windows™ never crashes
    - No patient is ever given the wrong medication
    - A program never terminates with a wrong answer
- Liveness: "something good eventually happens"
  - no partial execution is irremediable
    - Windows™ always reboots
    - Medications are eventually distributed to patients
    - A program eventually terminates

A Really Cool Theorem

"Every property defined on an execution of a program is a combination of a safety property and a liveness property"

(Alpern and Schneider, 1987)

Too Much Milk!

Jack	Jill
☐ Look in the fridge: out of milk	
☐ Leave for store	☐ Look in fridge: no milk
☐ Arrive at store	☐ Leave for store
☐ Buy milk	☐ Arrive at store
☐ Arrive at home: put milk away	☐ Buy milk
	☐ Arrive at home: put milk away
	☐ Oh no!

Formalizing "Too Much Milk"

- Shared variables
  - "Look in the fridge for milk" - check a variable
  - "Put milk away" - update a variable
- Safety
  - At most one person buys milk
- Liveness
  - If milk is needed, eventually somebody buys milk

Solution #1: Leave a note

- If you find a note from your roommate don't buy!
  - Leave note ≈ lock
  - Remove note ≈ unlock

Jack/Jill

```
if (noMilk) {
  if (noNote) {
    leave Note;
    buy milk;
    remove Note
  }
}
```

Solution #1: Leave a note

- If you find a note from your roommate don't buy!
  - Leave note ≈ lock
  - Remove note ≈ unlock
- Safe?

Jack/Jill

```
if (milk == 0) {
  if (note==0) {
    note = 1;
    milk++;
    note = 0;
  }
}
```

T1

```
if (milk == 0) {
```

Switch

T2

```
if (milk == 0) {
  if (note==0) {
    note = 1;
    milk++;
    note = 0;
  }
}
```

Switch

T1

```
if (note==0) {
  note = 1;
  milk++;
  note = 0;
}
```

Oh no!

Solution #1: Leave a note

- If you find a note from your roommate don't buy!
  - Leave note ≈ lock
  - Remove note ≈ unlock
- Safe?
- This "solution" makes the problem worse!
  - sometime works, sometime doesn't

Jack/Jill

```
if (milk == 0) {
  if (note==0) {
    note = 1;
    milk++;
    note = 0;
  }
}
```

Solution #2: Colors

Jack

```
Leave Blue note
if (noPinknote) {
  if (noMilk) {
    buy milk;
  }
}
Remove Blue note
```

Jill

```
Leave Pink note
if (noBluenote) {
  if (noMilk) {
    buy milk;
  }
}
Remove Pink note
```

Solution #2: Colors

Jack

```
    noteA = 1;
A1  if (noteB == 0) {
A2    if (milk == 0) {
A3      milk++;
      }
    }
    noteA = 0;
```

Jill

```
    noteB = 1;
    if (noteA == 0) {      B1
      if (milk == 0) {     B2
        milk++;            B3
      }                    B4
    }                      B5
    noteB = 0;
```

Proof of Safety

By contradiction:
Suppose Jack and Jill both buy milk
Consider the state of variables (noteB,milk) at A_1

- Case 1: noteB == 1

  Impossible, since Jack ends up buying milk

- Case 2: noteB == 0, milk > 0

  Impossible. milk > 0 is a stable property, so Jack would fail test A_2 and never buy milk

- Case 3: noteB == 0, milk == 0

  Impossible. Jill cannot be executing in B_1-B_5. Since notaA==1, then Jill will not pass B_1

Solution #2: Colors

Jack

```
    noteA = 1;
A1  if (noteB == 0) {
A2    if (milk == 0) {
A3      milk++;
      }
    }
    noteA = 0;
```

Jill

```
    noteB = 1;
    if (noteA == 0) {      B1
      if (milk == 0) {     B2
        milk++;            B3
      }                    B4
    }                      B5
    noteB = 0;
```

Proof of Liveness

Not Live!

Solution #3

Jack

```
    noteA = 1;
    while (noteB == 1) {
        ;
    }
A1  if (milk == 0) {
      milk++;
    }
    noteA = 0;
```

Jill

```
    noteB = 1;
    if (noteA == 0) {
      if (milk == 0) {
        milk++;
      }
    }
    noteB = 0;
```

Proof of Safety

Similar to previous case

Proof of Liveness

Jill will eventually sets noteB = 0
Jack will then reach line A_1
  if Jack finds milk, done
  If still no milk, Jack will buy it

Too Much Milk: Lessons

- Last solution works, but it is really unsatisfactory:
  - Complicated; proving correctness is tricky even for the simple example
  - Inefficient: while thread is waiting, it is consuming CPU time
  - Asymmetric: hard to scale to many threads
  - Incorrect(?) : instruction reordering can produce surprising results

A better way

- How can we do better?
  - Define higher-level programming abstractions (shared objects, synchronization variables) to simplify concurrent programming
    - lock.acquire() - wait until lock is free, then grab it • atomic
    - lock.release() - unlock, waking up a waiter, if any • atomic

Jack/Jill/even Dame Dob!

```
Kitchen::buyIfNeeded() {
    lock.acquire():
    if (milk == 0) {
            milk++;
        }
    lock.release();
}
```

  - Use hardware to support atomic operations beyond load and store