Concurrency

Franck van Breugel

March 20, 2018

1 Race conditions and data races

A *race condition* is a flaw that occurs when the timing or ordering of events affects a program's correctness. Generally speaking, some kind of external timing or ordering non-determinism is needed to produce a race condition.

A *data race* happens when there are two memory accesses in a program where both

- target the same location,
- are performed concurrently by two threads,
- are not all reads (at least one is a write),
- are not synchronization operations.

Many race conditions are due to data races, and many data races lead to race conditions. However, we can have race conditions without data races and data races without race conditions.

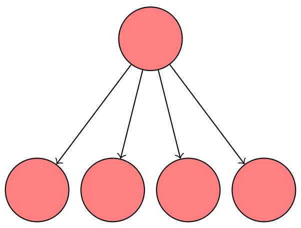
1. Give an example that has both a data race and a race condition.

2. Give an example that has a race condition but does not have a data race.

3. Give an example that has a data race but does not have a race condition.

2 SimpleRaceDetector

Develop a listener that detects data races (only for non-static attributes that are not arrays).



There is a data race if in the first instruction of one transition thread t_1 reads field f of object o and in the first instruction of another transition thread t_2 reads field f of object o and $t_1 \neq t_2$.

Fill in the dots.

```
for each branching state
  reads ← empty set
  writes ← empty set
    if the branching is caused by concurrency
      for each outgoing transition
        if the first instruction is a read by thread $t$
           of field $f$ of object $o$
          if ...
            ...
          ...
      if the first instruction is a write by thread $t$
          of field $f$ of object $o$
        if ...
          ...
        ...
```

3 Stack

1. What are the two operations of the abstract data type Stack?

2. We implement the stack as a singly linked list of nodes. Each node contains an element and a reference to the next node. The variable **`top`** refers to the first node of the linked list and is initially undefined (null).

   How can we implement the push operation?

3. How can we implement the pop operation?

4. The operation CAS(variable, expected, new) *atomically*

    - loads the value of variable,
    - compares that value to expected,
    - assigns new to variable if the comparison succeeds, and
    - returns the old value of variable.

    How can we implement the push operation?

5. How can we implement the pop operation?