Definition of Event Reordering and Data-races

Tobias Hoffmann ✉
Albert-Ludwigs-Universität Freiburg, Germany

1 Definition

Program & Threads

A *concurrent program* P consists of a set of *threads* $T = \{t_1, ..., t_n\}$.

Action & Event

An *event* $e = (i, a)$ is any *action* a performed by a thread t_i which potentially influences or is influenced by the execution of other threads. An action can be:

- $read(x)$: Read from variable x
- $write(x)$: Write to variable x
- $lock(l)$: Acquire lock l
- $unlock(l)$: Release lock l
- $fork(j)$: Create child thread t_j
- $join(j)$: Await child thread t_j to terminate

Hereby is x the identifier of a *global variable*, l the identifier of a *lock* and j is the index of a *child thread* t_j.

Trace of a program

The *trace* $\sigma = \{e_1, e_2, ..., e_m\}$ of a program P is the temporally ordered sequence of events of the threads T in P for a potential execution of P.

Last-Write of a Read Action

For a read event $e = (i, read(x))$, we define the *last-write* of e as the last event $e' = (j, write(x))$ that wrote to x before e by any thread. We write $lastwrite(e) := e'$.

Reordering & Happens-Before Relation

The relation *happens-before* between any two events $e = (i, a)$ and $e' = (j, a')$, written $e \prec e'$, is defined if any of the following conditions are met:

- $i = j$ and a happens before a' in t_i
- $a = fork(j)$
- $a' = join(i)$
- $i \neq j$ and $a = unlock(l)$ and $a' = lock(l)$

A *valid reordering* $r : \{1, ..., m\} \to \{1, ..., m\}$ of the trace $\sigma = \{e_1, ..., e_m\}$ of a program P is a bijective function that preserves the *happens-before* relation between all events:

$$\forall k \in \{1, ..., m\} \, \forall l \in \{1, ..., m\} : e_k \prec e_l \Rightarrow e_{r(k)} \prec e_{r(l)}$$

Data Race

A *data race* exists for a read event $e_k = (i, read(x))$ if there is a valid reordering r of the program's trace $\sigma = \{e_1, ..., e_m\}$ that changes the last-read of it:

$$lastread(e_k) \neq lastread(e_{r(k)})$$