

Program Language Memory Models: Axiomatic

October 21, 2026

First, draw an *event structure*, a graph containing all the reads and writes in the program with operations from the same thread connected by program order (**po**) edges. Then, try to apply the rules of the memory model to complete the event structure into an *execution*. In language-level models, we distinguish between *atomic* and *non-atomic* memory operations.

1 SC-DRF with locks

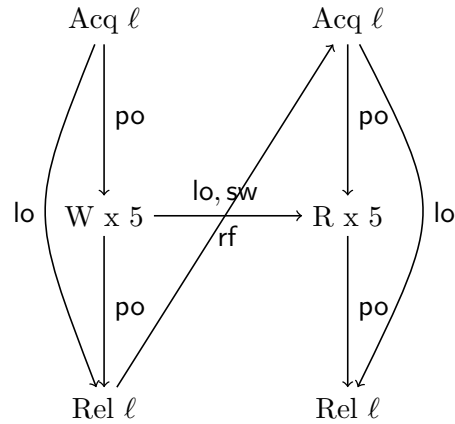
1. For each lock ℓ , there is a total order lo_ℓ on all operations to ℓ , consistent with **po**, where each Acq is followed by a Rel and vice versa.
2. If $\text{Rel } \ell \xrightarrow{\text{lo}} \text{Acq } \ell$, then $\text{Rel } \ell \xrightarrow{\text{sw}} \text{Acq } \ell$.
3. $\text{hb} := \text{po} \cup \text{sw}$ is acyclic. If there are two conflicting operations a and b such that $a \not\prec_{\text{hb}} b$ and $b \not\prec_{\text{hb}} a$, the execution is racy.
4. “No intervening writes”: For each location ℓ , there is a modification order mo_ℓ that is a total order on writes to ℓ . If $w_1 \xrightarrow{\text{rf}} r$ and $w_1 <_{\text{mo}_\ell} w_2$, then $w_2 \not\prec_{\text{hb} \cup \text{mo}_\ell} r$. Equivalently, there is a reads-before order rb_ℓ such that if $w_1 \xrightarrow{\text{rf}} r$ and $w_1 \xrightarrow{\text{mo}_\ell} w_2$ then $r \xrightarrow{\text{rb}_\ell} w_2$, and $\text{hb} \cup \text{mo}_\ell \cup \text{rb}_\ell$ is acyclic for each ℓ .

Example:

```

acquire(a); || acquire(a);
*x = 5;      || y = *x;
release(a); || release(a);
    
```

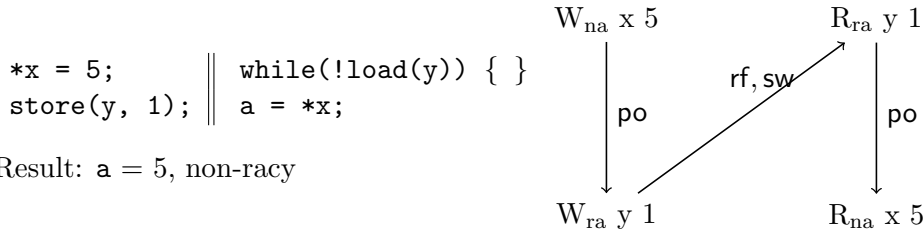
Result: $y = 5$



2 RA+NA

1. There is a read-from (rf) edge to each read from the write it reads from (i.e., a write to the same location with the same value). If the read and write are atomic, this is also a synchronizes-with (sw) edge.
2. $\text{hb} := \text{po} \cup \text{sw}$ is acyclic. If there are two conflicting non-atomic operations a and b such that $a \not\prec_{\text{hb}} b$ and $b \not\prec_{\text{hb}} a$, the execution is racy.
3. “No intervening writes”: For each location ℓ , there is a modification order mo_ℓ that is a total order on writes to ℓ . If $w_1 \xrightarrow{\text{rf}} r$ and $w_1 <_{\text{mo}_\ell} w_2$, then $w_2 \not\prec_{\text{hb} \cup \text{mo}_\ell} r$. Equivalently, there is a reads-before order rb_ℓ such that if $w_1 \xrightarrow{\text{rf}} r$ and $w_1 \xrightarrow{\text{mo}_\ell} w_2$ then $r \xrightarrow{\text{rb}_\ell} w_2$, and $\text{hb} \cup \text{mo}_\ell \cup \text{rb}_\ell$ is acyclic for each ℓ .

Example:



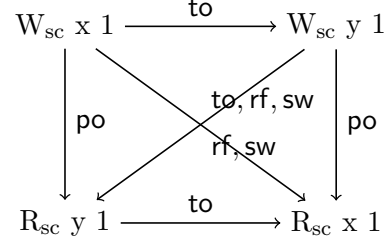
3 SC+NA

1. There is a total order **to** on all *atomic* operations, consistent with **po**.
2. There is a read-from (rf) edge to each read from the write it reads from (i.e., a write to the same location with the same value). If the read and write are atomic, this is also a synchronizes-with (sw) edge, and must be from the immediately preceding write to ℓ in **to**.
3. $\text{hb} := \text{po} \cup \text{sw}$ is acyclic. If there are two conflicting non-atomic operations a and b such that $a \not\prec_{\text{hb}} b$ and $b \not\prec_{\text{hb}} a$, the execution is racy.
4. “No intervening writes”: For each location ℓ , there is a modification order mo_ℓ that is a total order on writes to ℓ . If $w_1 \xrightarrow{\text{rf}} r$ and $w_1 <_{\text{mo}_\ell} w_2$, then $w_2 \not\prec_{\text{hb} \cup \text{mo}_\ell} r$. Equivalently, there is a reads-before order rb_ℓ such that if $w_1 \xrightarrow{\text{rf}} r$ and $w_1 \xrightarrow{\text{mo}_\ell} w_2$ then $r \xrightarrow{\text{rb}_\ell} w_2$, and $\text{hb} \cup \text{mo}_\ell \cup \text{rb}_\ell$ is acyclic for each ℓ .

Example:

```
store(x, 1); || store(y, 1);
r1 = load(y); || r2 = load(x);
```

Result: r1 = 1, r2 = 1



4 RA+RLX+NA

1. There is a read-from (rf) edge to each read from the write it reads from (i.e., a write to the same location with the same value). If the read and write are RA, this is also a synchronizes-with (sw) edge.
2. $\text{hb} := \text{po} \cup \text{sw}$ is acyclic. If there are two conflicting non-atomic operations a and b such that $a \not\prec_{\text{hb}} b$ and $b \not\prec_{\text{hb}} a$, the execution is racy.
3. “No intervening writes”: For each location ℓ , there is a modification order mo_ℓ that is a total order on writes to ℓ . If $w_1 \xrightarrow{\text{rf}} r$ and $w_1 <_{\text{mo}_\ell} w_2$, then $w_2 \not\prec_{\text{hb} \cup \text{mo}_\ell} r$. Equivalently, there is a reads-before order rb_ℓ such that if $w_1 \xrightarrow{\text{rf}} r$ and $w_1 \xrightarrow{\text{mo}_\ell} w_2$ then $r \xrightarrow{\text{rb}_\ell} w_2$, and $\text{hb} \cup \text{mo}_\ell \cup \text{rb}_\ell$ is acyclic for each ℓ .

Example:

```
a = atomic_load(x, rlx); || b = atomic_load(y, ra);
atomic_store(y, 1, ra); || atomic_store(x, 1, rlx);
```

Result: a = 0, b = 1, non-racy (but a = 1 is disallowed)

