0.1 From Examples to Precise Rules

Notations

- T_i denotes thread number i.
- $T_i \equiv T_j$ means both threads have same code.
- w_i^j is the j^{th} event in thread i which is a write.
- r_i^j is the j^{th} event in thread i which is a read.

A few definitions for our use

Definition 1. *Program Order(*po*) Total order between events in the same thread. Respects the execution order between events in the same thread.*

Definition 2. *Symmetric Memory Order (*smo*) A strict partal order between writes in a set of symmetric threads.*

Perhaps should put examples for the above defintion.

Definition 3. *Reads-From (*rf*) Binary relation that links a read to a write from which its value comes. Note that for our purpose, this relation is functional. For example, if a read r_i^j gets its read value from write w_k^l, then we have the relation.*

$$w_k^l \xrightarrow[rf]{} r_i^j$$

Name to be decided We consider a set of symmetric threads $T_1 \equiv T_2 \equiv ... \equiv T_n$. Each of these threads have exactly one read event, and multiple write events, all to the same memory, say x.

Each write in the above threads are involved in a symmetric order, such that.

$$\forall i \in [0, n-1] \; . \; w_i^j \xrightarrow[smo]{} w_{i+1}^j$$

Where j denotes the j^{th} event in any of the threads, which is a write.

Using the above setup, our intention is to explore lesser execution graphs leveraging the symmetry that can result due to swapping of thread identities. For this, we enforce a restriction on possible $\xrightarrow[rf]{}$ relations that are to be considered *valid*. A valid $\xrightarrow[rf]{}$ relation is one that respects the following **irreflexivity constraints**.

$$rf; po; smo$$
$$rf^{-1}; smo; po$$

We can add here more as we go about to prove completeness.

Place examples to show how our analysis through examples satisfy the above irreflexivity constraint.

Soundness of the rules above To prove soundness, we first define the following:

Definition 4. *Implied Write Order(*iwo*) These are the orders implied between any two writes, which can be derived as*

$$w_i^j; rf; po; w_k^j$$
$$w_i^j; po; rf^{-1}; w_k^j$$

When the implied write order and symmetric memory order form a cycle, that execution for us is considered invalid. This means the irreflexivity constraint can be also put as :

$$smo; iwo$$

Some observations on this:

- No write order is implied when a read reads from its own thread's write.
- Implied write orders between two symmetric threads are reversed when they are swapped.
- Either all implied write orders between two threads are compliant with symmetric order, or they are all not. (do an example based analysis, show that it would otherwise violate coherence).

Case for two threads

- For every implied write order against symmetric write order, swapping the thread identities reverses the implied write order, thus maintaining the irreflexivity constraint.
- Because all implied write orders between threads are either compliant or not, swapping thread identities does indeed give us an execution compliant with our irreflexivity constraints.
- Hence, for every execution not covered by our rules, there is a symmetric one covered.

Case for three or more threads Method 1:

- I see it as a simple case of bubble sort. Not sure how to show it though.
- Construct a total order for each set of equal writes using implied write orders and the symmetric memory order.
- Then the entire process of constructing a symmetric execution that respects our constraint is just a sorting problem.
- Note that write orders not implied can be freely set by us to respect symmetric memory order.
- How do we show this?

Method 1: For writes above and below

- Proof by contradiction
- Suppose all write orders above reads are sorted as per our need.

- Then for writes below, firstly, if an implied write order is not respected, then either the writes above also not respected or it is a free order.
- If it is not respected, this contradicts our assumption.
- If it is free, then we can freely swap the two threads, fixing the implied wrtie order of the write below the read.
- The above writes will still be in order, because the above write order between those two threads swapped was free, and smo is total w.r.t the equal writes of each symmetric thread.