Relaxed Stateless Model Checking, RA model

May 2017

1 Notations

• \(e_{i,j}^{w,x}\) denotes jth event of ith thread, w indicates that this is a write event and x indicates that it operates on variable x.

• similarly we define : \(e_{i,j}^{r,x}\) denotes jth event of ith thread, r indicates that this is a read event and x indicates that it operates on variable x.

2 Data structures associated with each event

• for each event \(e_{i,j}\) we want to maintain a \(hb\) (happens before) order and \(co\) (coherence order), we do this by storing two kinds of vector clocks :

  – \(\text{predecessor}\{x\}\{i\}\) stores the the last index of an event on variable x of ith thread which comes before e in poUrF\(Uco\)\(_x\).

  – \(\text{successor}\{x\}\{i\}\) stores the first index of an event on variable x of ith thread which comes after e in poUrF\(Uco\)\(_x\).

  – \(\text{predecessor}\{hb\}\{i\}\) stores the the last index of an event on any variable of ith thread which comes before e in poUrF.

  – \(\text{successor}\{hb\}\{i\}\) stores the first index of an event on any variable of ith thread which comes after e in poUrF.

• the intuitive idea behind these data structures and rest of our implementation is that, the information of addition of new edge in the trace is conveyed to some event of each thread. Thus when we are required to get the events which are hidden from some event we only need to look all the events which belong to the same thread and since events belonging to same thread have a strict total order, we get well defined range queries to answer questions like which events are hidden from some event. These range queries can be performed efficiently using segment trees.
3 Updating vector clocks on addition of new edges in trace

- **NOTE**: although we are saying that we are going to maintain the vector clocks on addition of each new edge, this may not be true at every instant. we are storing updates indirectly or lazily, so each time we want to find, say \( \text{predecessor}[x]\{i\} \) we are going to spend \( O(m^*\log(n)) \) time rather than \( O(1) \) time to get the exact value, where \( n \) is the number of events in that thread and \( m \) is the number of threads.

- suppose we add an edge from an event \( e_{i,j} \) to \( e'_{i',j'} \), due to the addition of this edge we want to somehow tell all events which are before \( e \) about the new \( \text{successors} \) introduced due to this edge and tell all the events who follow event \( e' \) (i.e. \( \text{successors} \) of \( e' \)) about the new \( \text{predecessors} \) introduced.

- this can be done efficiently using the data structures introduced earlier.

- suppose we observe a new write event \( e_{w,x}^{i,j} \). this write event induces only one edge in the trace i.e. the po edge between \( e_{i,j-1} \) and \( e_{i,j} \). now here we need to do two kind of updates,
  1. update \( \text{predecessor} \) vector clock of \( e_{i,j} \)
  2. update \( \text{successor} \) of \( e_{i,j-1} \) and its \( \text{predecessors} \)

To do (1), we copy vector clocks of \( e_{i,j-1}^{y} \) with few changes, i.e.

\[
\text{predecessor}_{i,j}\{hb\}\{i\} = j - 1
\]

and,

\[
\text{predecessor}_{i,j}\{x\} = \max(\text{predecessor}_{i,j}\{hb\}, \text{predecessor}_{i,j-1}\{x\})
\]

but we have to be careful here since as we have said earlier the vector clocks of \( e_{i,j-1} \) might be outdated thus we update its information,

\[
\text{predecessor}_{i,j-1} = \max_{k=0}^{n-1} (\text{predecessor}_{i,k})
\]

this can be done in \( m^*\log(n) \) time using segment trees with vectors as their nodes, where \( m \) is the number of processes and \( n \) is number of events.

To do (2), first we need to update successor vector-clock of \( e_{i,j-1} \) as \( \text{successor}_{i,j-1}^{y}\{hh\}\{i\} = j \) and \( \text{successor}_{i,j-1}\{x\}\{i\} = j \), then we go to all the events pointed in \( \text{predecessors}_{i,j-1}\{hh\} \), \( \text{predecessors}_{i,j-1}\{x\} \) and update their \( \text{successors}\{hh\} \), \( \text{successors}\{x\} \) respectively according to the new event added.

- now suppose we observe a read event \( e_{r,x}^{i,j} \); this read event firstly induces a po edge i.e. between \( e_{i,j-1} \) and \( e_{i,j} \).
the updates corresponding to this edge is similar to the updates done in
previous section with one change that we don’t do

\[ \text{predecessor}_{i,j} \{ x \} = \text{max} \left( \text{predecessor}_{i,j} \{ \text{hb} \}, \text{predecessor}_{i,j-1} \{ x \} \right) \]

rather we just do

\[ \text{predecessor}_{i,j} \{ x \} = \text{predecessor}_{i,j-1} \{ x \} \]

here we are supposing that we have maintained the set of maximal writes
which can be done using the method suggested by Phong.

now all the writes which come after \( \text{predecessor}_{i,j} \{ x \} \) are the possible
candidates for the observed read.

- now suppose we choose an event \( e^{w,x}_{i',j'} \) as the read-from for the read. This
  introduces one rf edge and few cox edges in the trace. updates related to
  the rf edge can be done similar to previous procedure.

for updates related to each edge between a maximal write \( e^{w,x}_{i'',j''} \) (say e")
and \( e^{w,x}_{i,j} \) (say e’) we have to perform two kinds of updates :

1. update successors of e’’ and its predecessors
2. update predecessors of e’ and its successors

before we perform any operation we must update the vector clocks of
\( e'' \) and e’ by doing \( m^* \text{log}(N) \) range query as suggested earlier.

to do (1) first we update \( \text{successor}_{i'',j''} \{ x \} \) as

\[ \text{successor}_{i'',j''} \{ x \} = \min \left( \text{successor}_{i'',j''} \{ x \}, \text{successor}_{i',j'} \{ x \} \right) \]

then we go to all the events which are pointed in \( \text{predecessor}_{i'',j''} \{ \text{hb} \} \) and
\( \text{predecessor}_{i'',j''} \{ x \} \) and update their \( \text{successor} \{ x \} \) vector clock similarly
as above. Note here we only update \( \text{successor} \{ x \} \) since the new edge was
a cox edge.

to do (2) we perform similar operations with the change that instead
of updating \( \text{successors} \) of the \( \text{predecessors} \) of e'' we would be updating
\( \text{predecessors} \) of \( \text{successors} \) of e’