TLP **18** (1): 120, 2018.
doi:10.1017/S1471068417000059

ERRATUM

Lock-free atom garbage collection for multithreaded Prolog - ERRATUM

JAN WIELEMAKER and KERI HARRIS

doi: 10.1017/S1471068416000272, Published by Cambridge University Press, 14th October 2016

Algorithm 4 on page 960 of the above named article (Wielemaker and Harris 2016) is flawed. The issue is illustrated by algorithm 1 (supplementary figure 1). If a thread A detects the condition table too full is false it proceeds adding its atom to the table. If thread B detects the table is (now) too full it starts a resize. The resize allocates a new table and copies the atoms from the old to the new table. If thread A adds the new atom after the copy loop passes its location and before thread B activates the new table the insertion is considered successful, but the new atom is only in the deactivated old table.

The problem was discovered about 18 months after the described algorithm was de-ployed in SWI-Prolog. The issue is triggered infrequently because, typically, thread A will insert the atom in the old table before thread B copies the specific symbol. Finishing the insertion late may happen if thread A is preempted during the . . . marked lines in algorithm 1. In addition, the table is only resized log N times where N is the final number of atoms.

The solution is to introduce a global variable *rehashing* that is set to **true** inside RESIZE_ATOM_TABLE while this function is active and tested to be **false** in the condition at line 33 of algorithm 4.

Thread A	Thread B

if table too full then
  RESIZE_ATOM_TABLE
end if
...
...
Add atom to table
	
	BEGIN RESIZE_ATOM_TABLE
	  allocate new table
	  copy symbols
	  ...
	  activate new table
	END RESIZE_ATOM_TABLE

Supplementary Fig. 1. Thread *A* can insert an atom into the old table after the old table has been copied and before it is activated.

Reference

Wielemaker, J. and Harris, K. 2016. Lock-free atom garbage collection for multithreaded prolog. TPLP 16, 5–6, 950–965.