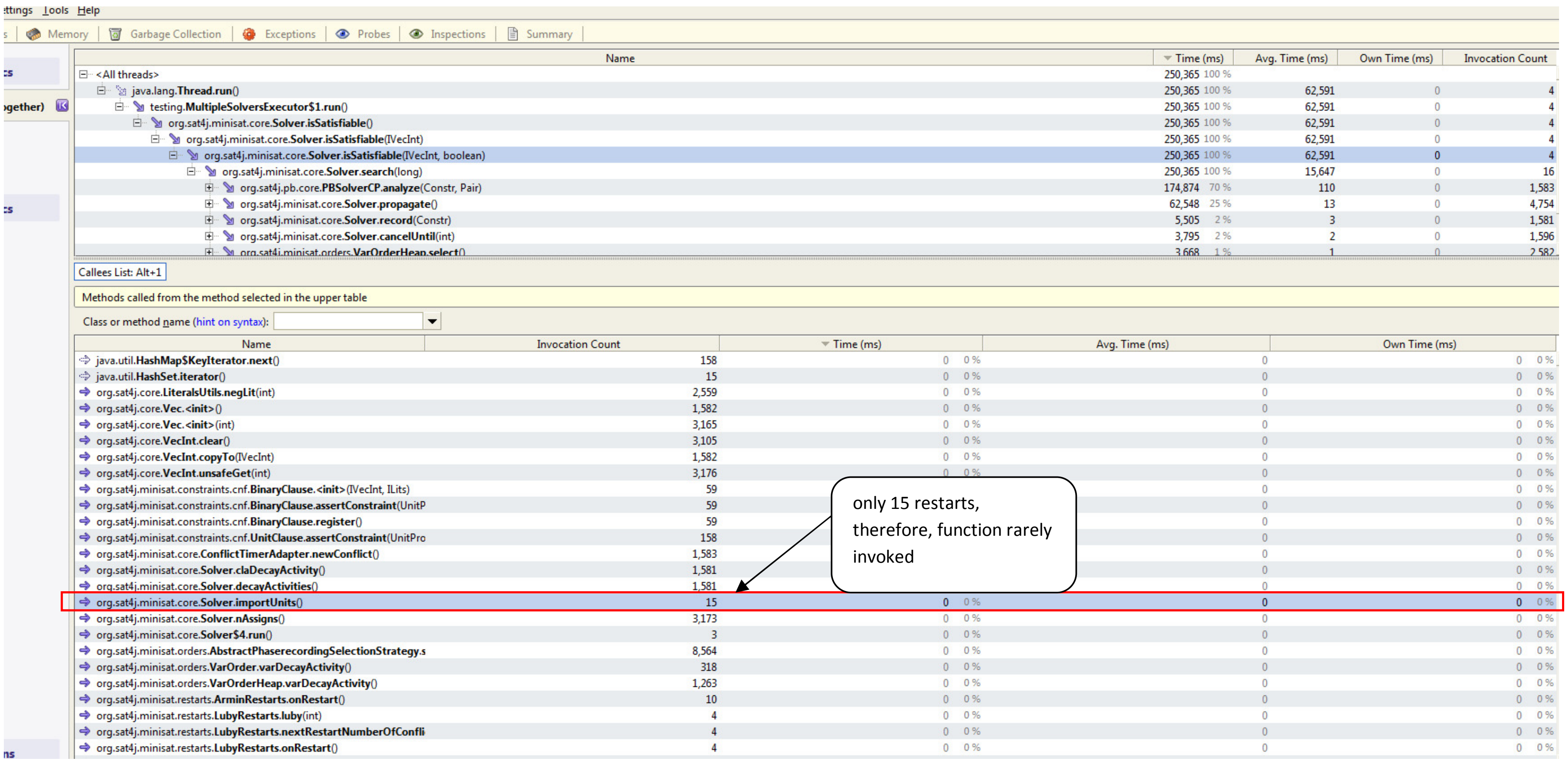
Profiling

Sharing mechanism overhead seems to be negligible, profiler estimated import and export around 0 ms and 0% (compared to other methods at the top of the screen). This happens due to low cost (in terms of "running time") of import and export, and also due to its infrequent invocation (i.e. importing only on restart, and exporting only when encountering unit clause).



Name	Time (ms)		Avg. Time (ms)	Own Time (ms)	Invocation Count
<All threads>	250,365	100 %			
java.lang.**Thread.run**()	250,365	100 %	62,591	0	4
testing.**MultipleSolversExecutor$1.run**()	250,365	100 %	62,591	0	4
org.sat4j.minisat.core.**Solver.isSatisfiable**()	250,365	100 %	62,591	0	4
org.sat4j.minisat.core.**Solver.isSatisfiable**(IVecInt)	250,365	100 %	62,591	0	4
org.sat4j.minisat.core.**Solver.isSatisfiable**(IVecInt, boolean)	250,365	100 %	62,591	0	4
org.sat4j.minisat.core.**Solver.search**(long)	250,365	100 %	15,647	0	16
org.sat4j.pb.core.**PBSolverCP.analyze**(Constr, Pair)	174,874	70 %	110	0	1,583
org.sat4j.minisat.core.**Solver.propagate**()	62,548	25 %	13	0	4,754
org.sat4j.minisat.core.**Solver.record**(Constr)	5,505	2 %	3	0	1,581
org.sat4j.minisat.core.**Solver.cancelUntil**(int)	3,795	2 %	2	0	1,596
org.sat4j.minisat.orders.**VarOrderHeap.select**()	3,668	1 %	1	0	2,582

Callees List: Alt+1

Methods called from the method selected in the upper table

Class or method name (hint on syntax):

Name	Invocation Count	Time (ms)		Avg. Time (ms)	Own Time (ms)	
java.util.**HashMap$KeyIterator.next**()	158	0	0 %	0	0	0 %
java.util.**HashSet.iterator**()	15	0	0 %	0	0	0 %
org.sat4j.core.**LiteralsUtils.negLit**(int)	2,559	0	0 %	0	0	0 %
org.sat4j.core.**Vec.<init>**()	1,582	0	0 %	0	0	0 %
org.sat4j.core.**Vec.<init>**(int)	3,165	0	0 %	0	0	0 %
org.sat4j.core.**VecInt.clear**()	3,105	0	0 %	0	0	0 %
org.sat4j.core.**VecInt.copyTo**(IVecInt)	1,582	0	0 %	0	0	0 %
org.sat4j.core.**VecInt.unsafeGet**(int)	3,176	0	0 %	0	0	0 %
org.sat4j.minisat.constraints.cnf.**BinaryClause.<init>**(IVecInt, ILits)	59			0	0	0 %
org.sat4j.minisat.constraints.cnf.**BinaryClause.assertConstraint**(UnitP	59			0	0	0 %
org.sat4j.minisat.constraints.cnf.**BinaryClause.register**()	59			0	0	0 %
org.sat4j.minisat.constraints.cnf.**UnitClause.assertConstraint**(UnitPro	158			0	0	0 %
org.sat4j.minisat.core.**ConflictTimerAdapter.newConflict**()	1,583			0	0	0 %
org.sat4j.minisat.core.**Solver.claDecayActivity**()	1,581			0	0	0 %
org.sat4j.minisat.core.**Solver.decayActivities**()	1,581			0	0	0 %
org.sat4j.minisat.core.**Solver.importUnits**()	15	0	0 %	0	0	0 %
org.sat4j.minisat.core.**Solver.nAssigns**()	3,173	0	0 %	0	0	0 %
org.sat4j.minisat.core.**Solver$4.run**()	3	0	0 %	0	0	0 %
org.sat4j.minisat.orders.**AbstractPhaserecordingSelectionStrategy.s**	8,564	0	0 %	0	0	0 %
org.sat4j.minisat.orders.**VarOrder.varDecayActivity**()	318	0	0 %	0	0	0 %
org.sat4j.minisat.orders.**VarOrderHeap.varDecayActivity**()	1,263	0	0 %	0	0	0 %
org.sat4j.minisat.restarts.**ArminRestarts.onRestart**()	10	0	0 %	0	0	0 %
org.sat4j.minisat.restarts.**LubyRestarts.luby**(int)	4	0	0 %	0	0	0 %
org.sat4j.minisat.restarts.**LubyRestarts.nextRestartNumberOfConfli**	4	0	0 %	0	0	0 %
org.sat4j.minisat.restarts.**LubyRestarts.onRestart**()	4	0	0 %	0	0	0 %

Sharing efficiency and its influence on # of decisions

1. According to the experiments below imported clauses consist of **around 75%** of clauses unknown to the solver at the point on import.

2. The number of decisions is slightly affected by sharing; the overall improvement is therefore quite small.

Comment: to improve efficiency, we did not prevent solvers from writing clauses to the shared queue multiple times. Therefore, in some cases the number of imported unit clause may contain the same clauses from different solvers.

Below is a sample of numbers (based on subset of SAT-UNSAT collection involving sharing of units):

		sharing unit clauses					**no sharing**		
file name	**result**	**time (sec)**	**# of decisions (A)**	**# of learnt units**	**# of imported units**	**# of unknown at import**	**time (sec)**	**# of decisions (B)**	**A-B**
normalized-22s.smv.opb	UNSAT	0.271	2680	6	3	0	0.285	2680	0
normalized-37s.smv.opb	UNSAT	0.436	254	1	5	0	0.444	254	0
normalized-dlx1c.rwmem.ucl.opb	UNSAT	1.411	7120	1	3	0	1.432	7120	0
normalized-dlx1c.rwmem1.ucl.opb	UNSAT	1.123	6733	1	1	0	1.221	6733	0
normalized-elf.rf6.ucl.opb	UNSAT	0.008	3	0	0	0	0.003	3	0
normalized-elf.rf7.ucl.opb	UNSAT	0.266	158	0	0	0	0.277	51	107
normalized-elf.rf8.ucl.opb	UNSAT	0.414	387	1	1	0	0.346	387	0
normalized-ooo.burch_dill.4.accl.ucl.opb	UNSAT	0.58	3037	0	0	0	0.626	3037	0
normalized-ooo.rf6.ucl.opb	UNSAT	0.365	369	0	2	0	0.369	369	0
normalized-ooo.rf7.ucl.opb	UNSAT	0.562	1029	0	0	0	0.579	1029	0
counting-easier-php-012-010.sat05-1172.reshuffled-07.opb	UNSAT	787.207	321703	1	1	1	781.602	324628	-2925
itox_vc1033.opb	SAT	8.483	101114	10	1	1	8.691	103178	-2064
normalized-cache.inv10.ucl.opb	UNSAT	0.206	143	0	2	1	0.207	48	95
normalized-dlx1c.ucl.opb	UNSAT	0.761	3628	0	1	1	0.756	3455	173
normalized-ooo.burch_dill.2.accl.ucl.opb	UNSAT	135.208	40652	3	1	1	114.483	37542	3110
normalized-ooo.rf8.ucl.opb	UNSAT	1.47	2384	0	1	1	1.506	2496	-112
eq.atree.braun.7.unsat.opb	UNSAT	9.601	19598	0	2	2	9.769	20053	-455
eq.atree.braun.8.unsat.opb	UNSAT	85.778	75589	3	2	2	91.291	78774	-3185
normalized-cache-ibm-q-unbounded.Ih2arity.ucl.opb	SAT	1.281	4070	0	2	2	1.276	4070	0

normalized-25s.smv.opb	UNSAT	0.188	1735	0	3	3	0.21	1735	0
normalized-43s.smv.opb	UNSAT	7.653	17552	1	5	3	11.939	18279	-727
itox_vc1044.opb	SAT	9.112	98969	8	4	4	21.27	361846	-262877
normalized-cache-ibm-q-unbounded.Icl2arity.ucl.opb	UNSAT	10.616	7193	0	4	4	12.584	7555	-362
normalized-elf.rf10.ucl.opb	UNSAT	12.924	7072	0	6	5	13.851	8335	-1263
normalized-ooo.rf9.ucl.opb	UNSAT	8.068	11933	2	5	5	8.07	9353	2580
itox_vc909.opb	SAT	8.411	157107	14	9	7	13.825	264183	-107076
itox_vc1130.opb	SAT	27.658	477220	26	9	8	24.63	196925	280295
itox_vc1138.opb	SAT	27.194	382360	26	11	9	29.833	452689	-70329
mizh-sha0-35-5.opb	SAT	301.406	1512613	10	16	13	368.759	1295633	216980
normalized-cache-ibm-q-unbounded.Ic22arity.ucl.opb	UNSAT	217.792	35574	3	13	13	160.888	18967	16607
mizh-sha0-35-3.opb	SAT	772.169	2210009	8	17	17	411.566	2064940	145069
mizh-sha0-35-4.opb	SAT	471.1	2153530	9	20	18	737.868	2497195	-343665
total time		**2909.722**					**2830.456**		
total # of decisions			**7663518**					**7793542**	
total diff. in # of decisions									**-130024**

Entire collection results:

total time		**11996.023**					**12448.22**		

As can be seen, the running time is improved by 4% (from ~12500 to ~12000), and the number of decision improved by 1.67% (from ~7793000 to ~7663000).

Conclusions

- Sharing mechanism overhead seems to be tiny, in fact negligible.
- Most of the units learnt are not known to the solver at the point of import.
- The number of decisions is slightly affected by sharing of units, and accordingly, the improvement in total time is quite low (4%).
- As a final remark, in our experiments, we've seen that sharing of more than a unit clause degrades the performances. Implementing the sharing mechanism for clauses with size > 1 involves modification of the internal data structures SAT4j uses (as opposed to sharing a unit clause which is more straight forward).
- Although the total improvement of sharing clauses is quite low, it demonstrates that there is a room for further investigation of this mechanism.