LEAP results

September 1, 2013

This document describes the results we have obtained applying LEAP to a set of examples. The results are divided into two sections. The first section presents the sequential examples, mainly based on skiplists. The second section presents the concurrent examples based on a ticketing system and lock-coupling lists.

Each section presents two tables. The first table describes each invariant candidate and the decision procedures used in order to verify them. The second table presents the required analysis time for each invariant candidate.

Consider the first table of each section. When dealing with concurrency, we say that a parametrized formula is a formula parametrized by some thread identifiers. The number of thread identifiers parametrizing an invariant candidate is what we call the *index* of the formula. This number is shown in the second column of the tables.

Depending on the proof rules we apply (B-INV, P-INV, SP-INV, ...) and the length of the program, a number of verification conditions (VCs) are generated. The number of generated VCs that needs to be verified as valid for each invariant candidate is shown in the third column. A feature of LEAP is the possibility of applying tactics in order to simplify VCs. Applying a tactic on a single VC may simplify it, or it may produce many new VCs that subsumes the original VC. However, the idea is that all new VCs obtained through the application of tactics are simpler to analyze than the original one. Thus, the number of actual final VCs (or queries as we call them) passed to the solvers may be more than the original number of VCs. The number of final queries is depicted in the forth column of the table.

The final eight columns of the table contain the number of calls performed to each decision procedure in order to solve all the queries. We use the following decision procedures:

- A FO decision procedure, which reasons only about first order logic and information regarding the program counter. This DP usually solves simple queries in almost no time, speeding up the verification process.
- A Num decision procedure, which is capable of reasoning about Presburger Arithmetic with sets of integers.
- A TLL decision procedure, capable of dealing with concurrent lock-coupling single-linked lists.
- A TSL decision procedure, which deals with skiplists of arbitrary height.
- A set of TSLK_1, TSLK_2, TSLK_3 and TSLK_4 decision procedures, capable of reasoning about skiplists with a fixed number of levels (1, 2, 3 and 4 respectively).

In all cases not involving the use of TSL, the sum of calls performed to all decision procedures is equal to the number of generated queries. On the other hand, in the case of the skiplist example involving TSL, notice that only the sum of calls performed to FO and TSL is equal to the number of generated queries. This means that each query is solved through first order logic reasoning or, if not possible, through a call to the TSL decision procedure. However, because of how TSL decision procedure works, a single call to TSL may require multiple calls to Num (when guessing arrangements) as well as to simpler decision procedures such as TSLK_1, TSLK_2, TSLK_3 and TSLK_4.

Now, consider the second table on each section. These tables show the time required in order to analyze all generated VCs for each invariant candidate. The first column presents the name of the invariant candidate. The second column depicts the number of generated VCs (which matches with the first table). Then, we present three columns showing the least, most and average time it takes to analyze a single VC. The sixth column shows the total time employed by the decision procedures to analyze all VCs. The final column presents the total time required by LEAP in order to completely analyze and verify the invariant candidate. This value is greater than the one shown in the previous column as it includes, in addition to the time employed by the decision procedures, the time required to parse the program, the invariant candidates, generate the VCs, apply tactics, transform the intermediate formulas, etc. All the times shown in these tables are in seconds.

1 Sequential examples

	formula info			#calls performed to each DP							
	index	#vc	#query	FO	Num	TLL	TSL	TSLK_1	TSLK_2	TSLK_3	TSLK_4
skiplist_1	0	77	119	87	–	–	–	32	–	–	–
region_1	0	77	119	92	–	–	–	27	–	–	–
next_1	0	77	79	60	–	–	–	19	–	–	–
order_1	0	77	79	54	–	–	–	25	–	–	–
value_1	0	77	79	72	–	–	–	7	–	–	–
skiplist_2	0	79	137	90	–	–	–	–	47	–	–
region_2	0	79	122	95	–	–	–	–	27	–	–
next_2	0	79	82	63	–	–	–	–	19	–	–
order_2	0	79	82	57	–	–	–	–	25	–	–
value_2	0	79	82	75	–	–	–	–	7	–	–
skiplist_3	0	80	154	92	–	–	–	–	–	62	–
region_3	0	80	124	97	–	–	–	–	–	27	–
next_3	0	80	84	65	–	–	–	–	–	19	–
order_3	0	80	84	59	–	–	–	–	–	25	–
value_3	0	80	84	77	–	–	–	–	–	7	–
skiplist_4	0	81	171	?	–	–	–	–	–	?	–
region_4	0	81	126	99	–	–	–	–	–	27	–
next_4	0	81	86	67	–	–	–	–	–	19	–
order_4	0	81	86	61	–	–	–	–	–	25	–
value_4	0	81	86	79	–	–	–	–	–	7	–
skiplist	0	80	560	532	8760	–	28	47	94	46	16
region	0	80	1346	1299	6502	–	47	107	232	116	–
next	0	80	1346	1327	345	–	19	21	34	18	–
order	0	80	2136	2074	7418	–	62	212	442	182	3
value	0	80	82	79	9	–	3	2	1	–	–
levels	0	80	80	66	14	–	14	–	–	–	–

	# queries	individual vc time (sec.)			total dp time (sec.)	total analysis time (sec.)
		fastest	slowest	average		
skiplist$_1$	77	0.001	0.208	0.011	0.864	2.060
region$_1$	77	0.001	0.128	0.008	0.628	3.044
next$_1$	77	0.001	0.032	0.003	0.236	1.560
order$_1$	77	0.001	0.056	0.008	0.624	4.684
value$_1$	77	0.001	0.012	0.002	0.176	1.024
skiplist$_2$	79	0.001	2.856	0.070	5.556	7.580
region$_2$	79	0.001	1.036	0.039	3.048	6.248
next$_2$	79	0.001	0.076	0.006	0.472	1.944
order$_2$	79	0.001	0.888	0.020	1.552	7.748
value$_2$	79	0.001	0.012	0.001	0.056	0.992
skiplist$_3$	80	0.001	398.597	9.516	761.316	766.764
region$_3$	80	0.001	17.609	0.371	29.690	33.746
next$_3$	80	0.001	0.208	0.010	0.760	2.412
order$_3$	80	0.001	6.300	0.092	7.392	15.793
value$_3$	80	0.001	0.016	0.002	0.120	1.044
region$_4$	81	?	?	?	TO	TO
region$_4$	81	0.001	246.967	4.547	368.279	373.227
next$_4$	81	0.001	0.252	0.013	1.088	2.840
order$_4$	81	0.001	30.370	0.398	32.226	42.910
value$_4$	81	0.001	0.020	0.001	0.112	1.008
skiplist	80	0.001	22.185	0.718	57.468	58.556
region	80	0.001	78.717	2.204	176.307	185.276
next	80	0.001	0.612	0.035	2.812	10.617
order	80	0.001	19.057	0.526	42.079	70.132
value	80	0.001	0.028	0.001	0.080	0.484
levels	80	0.001	0.004	0.001	0.036	0.748

2 Concurrent examples

	formula info			#calls performed to each DP							
	index	#vc	#query	FO	Num	TLL	TSL	TSLK_1	TSLK_2	TSLK_3	TSLK_4
mutex	2	19	19	18	1	–	–	–	–	–	–
minticket	1	19	19	18	1	–	–	–	–	–	–
notsame	2	19	19	18	1	–	–	–	–	–	–
activelow	1	19	19	17	2	–	–	–	–	–	–
mutexS	2	19	19	18	1	–	–	–	–	–	–
minticketS	1	19	19	17	2	–	–	–	–	–	–
notsameS	2	19	19	18	1	–	–	–	–	–	–
activelowS	1	19	19	17	2	–	–	–	–	–	–
list	0	61	610	472	–	138	–	–	–	–	–
order	1	121	121	69	–	52	–	–	–	–	–
lock	1	121	121	82	–	39	–	–	–	–	–
next	1	121	121	68	–	53	–	–	–	–	–
region	1	121	147	120	–	27	–	–	–	–	–
disj	2	181	181	177	–	4	–	–	–	–	–

	# queries	individual vc time (sec.)			total dp time (sec.)	total analysis time (sec.)
		fastest	slowest	average		
mutex	19	0.001	0.008	0.002	0.032	0.080
minticket	19	0.001	0.004	0.001	0.012	0.028
notsame	19	0.001	0.004	0.001	0.024	0.088
activelow	19	0.001	0.004	0.001	0.008	0.028
mutex	19	0.001	0.012	0.002	0.036	0.088
minticketS	19	0.001	0.008	0.001	0.012	0.040
notsameS	19	0.001	0.004	0.001	0.020	0.084
activelowS	19	0.001	0.004	0.001	0.008	0.032
list	61	0.001	9.075	0.250	15.225	16.420
order	121	0.001	0.200	0.020	2.300	3.930
lock	121	0.001	0.024	0.005	0.588	1.380
next	121	0.001	1.024	0.024	2.856	5.170
region	121	0.001	8.160	0.076	9.210	10.333
disj	181	0.001	0.020	0.001	0.244	0.864