Table 1: Experimental evaluation

Benchmark	Bootstrap		Upgrade check					
Name	Total [s]	Itp [s]	Total [s]	Diff [s]	Itp [s]	**Speedup**	Result	ISR
ABB_A	8.644	0.008	0.04	0.009	0.003	**220**x	SAFE	0/7
ABB_B	6.236	0.009	0.006	0.006	—	**935**x	SAFE	0/9
ABB_C	8.532	0.015	0.059	0.008	0.003	**157**x	SAFE	0/8
VTT_A	0.512	0.001	0.006	0.006	—	**85.5**x	SAFE	0/9
VTT_B	0.514	0.001	0.031	0.006	—	**0.7**x	BUG	1/9
euler_A	12.56	0.099	0.179	0.001	0.016	**70.4**x	SAFE	1/6
euler_B	12.547	0.095	2.622	0.001	0.031	**4.74**x	SAFE	3/5
life_A	13.911	1.366	0.181	0.001	<0.001	**77.0**x	SAFE	0/5
life_B	13.891	1.357	6.774	0.001	—	**0.31**x	BUG	5/5
arithm_A	0.147	0.007	0.355	0.001	—	**0.39**x	BUG	3/3
diskperf_A	0.167	0.001	0.024	0.008	<0.001	**5.79**x	SAFE	0/21
diskperf_B	0.137	0.001	0.062	0.009	—	**2.25**x	BUG	3/21
floppy_A	2.146	0.229	0.422	0.202	<0.001	**5.02**x	SAFE	0/226
floppy_B	2.183	0.237	2.277	0.206	—	**0.82**x	BUG	79/226
kbfiltr_A	0.288	0.011	0.081	0.023	0.001	**3.40**x	SAFE	1/63
kbfiltr_B	0.320	0.009	0.088	0.023	0.001	**1.85**x	SAFE	3/63

Table 1 represents results of the experiments. Each benchmark is shown in a separate row, which summarizes statistics about the initial verification and verification of an upgrade. Time (in seconds) for running the syntactic difference check (**Diff**) and for generation of the interpolants (**Itp**) represents the computational overhead of the upgrade checking procedure, and included in the total running time (**Total**) of `eVolCheck`. Note that interpolation can not be performed at the buggy examples (marked as "—"), for which the corresponded PBMC formula is satisfiable. To show advantages of our upgrade checking approach, for each change we calculated the speedup (**Speedup**) of the upgrade check versus standalone verification of the changed code from scratch, performed only for the sake of comparison reasons and thus not shown in the table. Finally, the posteriori estimation of the upgrade check complexity is shown in the row **ISR** (Invalid Summaries Ratio). This ratio represents the number of invalid summaries (due to the change) with respect to the number of nodes in the call tree of the verified program.

Discussion. Our evaluation demonstrates good performance of `eVolCheck`. In particular, the experiments show high efficiency of upgrade checking for safe upgrades since they result in a small number of refinements (both, upward and downward). This generally leads to a small number of invalidated summaries, as witnesses by the corresponding **ISR** (see, for example, the ABB_n cases, where summaries of all changed functions were proven valid). It is less efficient (for some tests) in case of buggy upgrades, since bugs frequently (as expected) effect larger portions of the program. In classical model checking, confirming the absence of bugs is usually more expensive (since it requires the full state-space search) then detecting the bugs (where the search can be terminated once the bug is detected) and we believe that the fact that `eVolCheck` works so well to confirm safety is very useful for routine analysis of upgrades. In the majority of experiments, the localized upgrade check provided by `eVolCheck`, as expected, outperforms the verification from scratch, which is indicated by **speedup** > 1.