CVC5-gg at the SMT Competition 2021

Clark Barrett[1], Alex Ozdemir[1], Andres Nötzli[1], Andrew Reynolds[2],
Cesare Tinelli[2], Amalee Wilson[1], and Haoze Wu[1]

[1]Stanford University
[2]The University of Iowa

***Abstract*—This paper is a description of the CVC5-gg SMT solver as entered into the cloud and parallel tracks at the 2021 SMT Competition. CVC5-gg instruments CVC5 to split problems recursively into independent subproblems using a theory-agnostic approach. It uses CVC5 to solve the subproblems and gg to distribute the subproblems across different cores and machines. CVC5-gg is currently work in progress and our submission is an early prototype.**

Overview

CVC5-gg consists of four main components: (a) an abstract divide-and-conquer algorithm, (b) a *base solver* that attempts to solve queries, (c) a *splitter* that divides queries into easier ones, and (d) an infrastructure which schedules and executes these tasks. The splitter and the base solver are both based on the SMT solver CVC5 [2], the successor of CVC4 [4]. The executor is gg [5].

Divide-and-Conquer SMT Solving

Divide-and-conquer SMT solving recursively solves SMT formulae by splitting the original formula into multiple independent sub-problems. The algorithm attempts to solve the original formula F using a *base solver* within some *initial timeout* t_0. If that timeout is exceeded, then the system invokes a *splitter* which is given F and must split the formula into d *cubes* $C_1, \ldots C_d$, which encode a partitioning of the search space. More precisely, F and $(F \wedge C_1) \vee \cdots \vee (F \wedge C_d)$ must be equisatisfiable and, as a consequence, if there exists a satisfiable sub-problem $F \wedge C_i$, then F is satisfiable. Conversely, if F is unsatisfiable, then all sub-problems $F \wedge C_i$ are unsatisfiable. After the splitter completes, the system adds sub-problems $F \wedge C_1$ through $F \wedge C_d$ to its queue, to be recursively solved with an increased timeout $f \cdot t$, where t is the timeout of the problem before this split and f is a timeout growth factor. Through recursion, hard queries are split into ever finer sub-problems attempted with exponentially increasing timeouts.

The system is configured by t_0 (the initial timeout), f (the timeout growth factor), d (the number of sub-problems per split), and d_0 (the number of splits to perform before the initial solve attempt). In our implementation, we set $t_0 = 60$ seconds, $f = 1.5$, $d = 4$, and $d_0 = 16$. We use the base solver, splitter, and task executor described in the sections below.

Solving Queries

To solve the sub-problems, CVC5-gg invokes CVC5 with increasing timeouts. Currently, CVC5-gg uses CVC5 in its default configuration and does not retrieve any additional information other than the satisfiability of a given sub-problem.

Splitting Queries

CVC5-gg includes an instrumented version of CVC5 that is used to split a problem into d sub-problems by computing the aforementioned cubes C_i. CVC5 is a CDCL($\mathcal{T}$)-based SMT solver and the splitter instruments the core search to generate sub-problems in a theory-agnostic way. CDCL($\mathcal{T}$)-based SMT solvers use a SAT solver to produce satisfying assignments at the propositional level. The theory solvers then find lemmas, conflicts, and propagations based on the conjunction of theory literals that is extracted from such an assignment. To split a given problem, the splitter instruments CVC5 to intercept calls to theory solvers. It collects the literals $l_1, \ldots, l_m$ that correspond to the *decisions* that the current propositional assignment is based on. If the number of literals m is greater or equal to $log_2(d)$ where d is the desired number of sub-problems, then the splitter outputs a new cube $l_1 \wedge \ldots \wedge l_n$ (i.e., a conjunction of the first n literals) and asserts the lemma $\neg(l_1 \wedge \ldots \wedge l_n)$. The lemma forces CVC5's procedure to backtrack and skip to a different part of the search space. Intuitively, the search space can be skipped because it will later be investigated by solving the sub-problem. If the number is smaller, then there is a risk that the splitter cannot produce the requested number of sub-problems. To solve this issue, the splitter does not output a cube in that case, but instead lets CVC5 proceed with its invocation of the theory solvers. If these solvers produce lemmas, the lemmas potentially result in additional decisions, e.g., if they are splitting lemmas, and, as a result, additional cubes. If the theory solvers find conflicts, then the current part of the search space is uninteresting and no sub-problem needs to be generated. After the splitter produces $n-1$ sub-problems, it generates a sub-problem that corresponds to the remaining search space by generating the final cube $\neg C_1 \vee \ldots \vee \neg C_{n-1}$ to ensure that the n sub-problems cover the full search space.

Instrumenting CVC5 instead of implementing an independent splitter has several advantages: CVC5-gg can reuse existing infrastructure to represent and parse SMT problems, trivial sub-problems and trivially entailed literals in cubes are skipped by construction, and lemmas generated by theory solvers are automatically used to generate theory-relevant cubes without modifying the theory solvers themselves. The approach skips trivial sub-problems when CVC5 detects conflicts on partial

assignments and cubes do not include literals that are trivially entailed by the other literals in the cube because they would be propagated during CVC5's search and thus not count as a decision.

Scheduling and Executing Tasks

CVC5-gg uses gg [5] to define and execute a divide-and-conquer search. gg is a tool for dynamically defining dependency graphs of tasks and executing them in parallel. gg dependency graphs comprise of *values* (simple files) and *thunks* (tasks) which run a single executable or shell script on some input files and produce some output files. The input files may be values or outputs of other thunks. The outputs may be values or thunks. In the latter case, these thunks are added to the graph, allowing it to dynamically grow.

CVC5-gg describes its divide-and-conquer search algorithm using the pygg library [3]: a Python interface to gg. Thus, its solver is ultimately driven by a short python script which describes the primitive tasks (splitting, solving, and merging solutions) as thunks using pygg.

During dependency graph evaluation, the gg runtime uses a configurable storage engine and a configurable list of execution engines. An execution engine is responsible for evaluating individual thunks; implemented engines include the local machine, AWS Lambda, a remote machine running a gg execution server, and Google Cloud Engines. The storage service is responsible for storing values and thunks until they are needed by an executor; implemented services are built around AWS S3 and Redis.

For the SMT Competition 2021, CVC5-gg uses a gg runtime configured as follows. Its storage service is a Redis datastore running on the master node of the cluster. Its execution engines are gg execution servers running on all nodes of the cluster. The master node additionally runs the driver of the gg runtime which manages the dependency graph and schedules tasks.

Conclusion

While CVC5-gg is only an early prototype of a divide-and-conquer SMT solver, we believe the approach to be promising. The divide-and-conquer architecture permits both theory-agnostic splitters (which can be easily applied to new theories) and theory-specific splitters (which can be optimized to the dynamics of a specific theory, or even specific family of benchmarks). Our work on CVC5-gg is ongoing.

Acknowledgments

Like CVC5, CVC5-gg is supported in part by the organizations listed on CVC5's website [1].

References

[1] cvc5 acknowledgments. https://cvc5.github.io/acknowledgements.html, 2020.

[2] CVC5 website. https://cvc5.github.io/, 2021.

[3] Py-gg. https://github.com/gg-project/gg/tree/next/tools/pygg, 2021.

[4] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[5] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019*, pages 475–488, 2019. URL https://www.usenix.org/conference/atc19/presentation/fouladi.