Dartagnan: Bounded Model Checking for Weak Memory Models (Competition Contribution)

Hernán Ponce-de-León*[1], Florian Furbach[2],
Keijo Heljanko[3], and Roland Meyer[2]

[1]University of the Bundeswehr Munich, Munich, Germany
[2]TU Braunschweig, Braunschweig, Germany
[3]University of Helsinki and HIIT, Helsinki, Finland

Abstract. Dartagnan is a bounded model checker for concurrent programs under weak memory models. What makes it different from other tools is that the memory model is not hard-coded inside Dartagnan but taken as part of the input. For SV-COMP'20, we take as input sequential consistency (i.e. the standard interleaving memory model) extended by support for atomic blocks. Our point is to demonstrate that a universal tool can be competitive and perform well in SV-COMP. Being a bounded model checker, Dartagnan's focus is on disproving safety properties by finding counterexample executions. For programs with bounded loops, Dartagnan performs an iterative unwinding that results in a complete analysis. The SV-COMP'20 version of Dartagnan works on Boogie code. The C programs of the competition are translated internally to Boogie using SMACK.

1 Overview and Software Architecture

Dartagnan is a bounded model checker for concurrent programs under weak memory models. It expects as input a program P annotated with a reachability condition S, a memory model $\mathcal{M}$, and an unrolling bound k. It recursively unwinds all loops in P up to the bound k. The unwound program is converted into an SMT formula that symbolically represents all candidate executions. The memory model will filter out some candidates using a second formula, we explain this below. Events of a candidate execution model (instances of) program instructions, like memory accesses, local computations, and conditional/unconditional jumps. Edges model relations between events, including *program order* (the order within a thread), *data-dependencies* (an assigned variable is used within an expression), *reads-from* (matching each read with the write from which it takes its value), and *coherence* (the order in which writes commit to the memory).

A memory model can be understood as a predicate over candidate executions that declares some of them valid. We describe memory models in the CAT language [2]. A memory model is defined as a set of relations (those mentioned

* Jury member.

A. Biere and D. Parker (Eds.): TACAS 2020, LNCS 12079, pp. 378–382, 2020.
https://doi.org/10.1007/978-3-030-45237-7_24

SC+atomicity

$\mathsf{com} = \mathsf{co} \cup \mathsf{fr} \cup \mathsf{rf}$ $\mathsf{come} = \mathsf{com} \cap \mathsf{ext}$

$\mathsf{acyclic}\ \mathsf{po} \cup \mathsf{com}$ $\mathsf{empty}\ \mathsf{rmw} \cap (\mathsf{come}\,;(\mathsf{po} \cup \mathsf{com})^*\,;\mathsf{come})$

Fig. 1. CAT model used for SV-COMP'20.

above and others derived as unions, transitive/reflexive closures, compositions, etc.) and constraints over them (emptiness, acyclicity and irreflexivity). Given a memory model, we construct a formula that evaluates to true precisely under the candidate executions that are valid according to the memory model. Figure 1 shows the memory model used for SV-COMP'20. To support atomic blocks, Dartagnan adds a specific edge (rmw) for every pair of events between `VERIFIER_atomic_begin()` and its matching `VERIFIER_atomic_end()` or in a `VERIFIER_atomic_` function. We encode atomicity for sequential consistency (SC) as the empty intersection of rwm and paths starting and ending with an external communication (i.e. between different threads). This means once an atomic block starts, external communications with the block are forbidden until all events in the block have been executed.

Dartagnan comes with a rich assertion language inspired by Herd [1]. Assertions define inequalities over the values of local and global variables. They can be used freely throughout the code, rather than being limited to the end of the execution. Semantically, our assertions do not stop the execution but record the failure and continue. To achieve this, each instructions `assert(exp)` is transformed to a local computation $\mathtt{f} \leftarrow \mathtt{exp}$ where the fresh variable $\mathtt{f} \in \mathtt{F}$ stores the value of `exp` at the corresponding point of the execution. We refer to the formula $\bigvee_{\mathtt{f} \in \mathtt{F}} \neg\mathtt{f}$ as the reachability condition.

The formula for candidate executions of the program, the formula for validity under the given memory model, and the reachability condition together (in conjunction) yield the SMT encoding of the reachability problem at hand. Any solution to the conjunction corresponds to an execution that is valid according to the memory model and violates at least one assertion. Details on the encoding can be found in [8,9].

Dartagnan implements a may-alias analysis to improve pointer precision and a novel relation analysis. The latter technique reduces the SMT encoding to those parts of the relations that might affect the consistency with the memory model, resulting in a considerably smaller formula. Relation analysis improves the performance up to two orders of magnitude [4,5]. We remark that related approaches represent each candidate execution explicitly [1,6]. Thanks to the symbolic representation of executions and static analysis techniques such as relation analysis, Dartagnan is often more efficient [4,5].

Figure 2 shows the overall architecture of Dartagnan. It reads programs written in the litmus format of Herd [1] or the intermediate verification language Boogie [7]. For the competition, C programs are compiled to LLVM and then

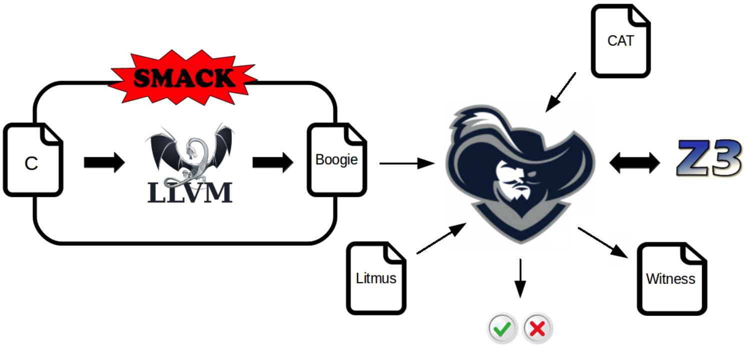

Fig. 2. DARTAGNAN's architecture.

translated internally to BOOGIE using the SMACK tool [11]. The SMT solver is Z3 [3]. When a violation is found, DARTAGNAN returns a witness execution.

2 Strengths and Weaknesses

The main strength of DARTAGNAN is its fully configurable memory model. Unfortunately, in SV-COMP'20 there is no category for verification tasks under weak memory models. On the SV-COMP'20 benchmarks, DARTAGNAN reports only one incorrect result, being beaten in that aspect only by CPACHECKER, DIVINE, LAZY-CSEQ and YOGAR-CBMC; three of them category winners. The incorrect result is related to the use of pointer arithmetic which is currently not supported by our alias analysis.

Its main strength is also its main weakness: DARTAGNAN's performance cannot quite match that of other verifiers that were developed specifically for sequential consistency. DARTAGNAN performs particularly poor on benchmarks with big atomics blocks. This is the case for most of the verification tasks in the `pthread-wmm` group which represent 83% of the ConcurrencySafety category. The problem is that DARTAGNAN adds rmw edges for all pairs in an atomic block. This results in a large encoding (even using relation analysis) and highly impacts its performance.

3 Tool Setup and Configuration

Besides the program to be verified, DARTAGNAN expects a CAT file containing the memory model of interest. For SV-COMP'20, this is the extension of sequential consistency given in Figure 1. The tool is run by executing the following command:

```
$ java -jar dartagnan/target/dartagnan-V-jar-with-dependencies.jar
  -cat <CAT file> -i <program file> [options]
```

Placeholder `V` is the tool version (currently 2.0.5) and `options` is used to configure the unrolling bound, the alias analysis, and the fixpoint encoding. The full list of options can be found on the project website (see Section 4).

To make sure not to miss a violation, the competition version of Dartagnan implements an iterative approach. Initially, the bounded model checking algorithm is called with an unrolling bound of one. If it finds a violation or can prove that all loops have been unrolled completely (this is done using unwinding assertions), the verification process terminates with a conclusive answer. If not, Dartagnan increases the bound by one and repeats the process. For program with an infinite state space, our tool does not terminate.

Dartagnan participates in the ConcurrencySafety category. No specification file is required. The artifact is available on [10]. To reproduce the results of the competition, the tool can be executed with the following wrapper script:

```
$ Dartagnan-SVCOMP.sh <program file>
```

4 Software Project and Contributors

The project home page is https://github.com/hernanponcedeleon/Dat3M. Dartagnan is open source software distributed under the MIT license.

Acknowledgement: We thank Dirk Beyer and Philipp Wendler for their help during the process of integrating Dartagnan into the competition framework. We also thank Natalia Gavrilenko for her contributions to the development of the bounded model checking engine of the tool [4,5].

References

1. The herdtools7 tool suite. https://github.com/herd/herdtools7.
2. Jade Alglave, Patrick Cousot, and Luc Maranget. Syntax and semantics of the weak consistency model specification language CAT. *CoRR*, abs/1608.07531, 2016.
3. Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
4. Natalia Gavrilenko. Improving scalability of bounded model checking for weak memory models. Master's thesis, Aalto University, Department of Computer Science, 2019.
5. Natalia Gavrilenko, Hernán Ponce de León, Florian Furbach, Keijo Heljanko, and Roland Meyer. BMC for weak memory models: Relation analysis for compact SMT encodings. In *CAV*, volume 11561 of *LNCS*, pages 355–365. Springer, 2019.
6. Stella Lau, Victor B. F. Gomes, Kayvan Memarian, Jean Pichon-Pharabod, and Peter Sewell. Cerberus-BMC: A principled reference semantics and exploration tool for concurrent and sequential C. In *CAV*, volume 11561 of *LNCS*, pages 387–397. Springer, 2019.
7. K. Rustan M. Leino. This is Boogie 2. 2008.
8. Hernán Ponce de León, Florian Furbach, Keijo Heljanko, and Roland Meyer. Portability analysis for weak memory models. PORTHOS: One tool for all models. In *SAS*, volume 10422 of *LNCS*, pages 299–320. Springer, 2017.

9. Hernán Ponce de León, Florian Furbach, Keijo Heljanko, and Roland Meyer. BMC with memory models as modules. In *FMCAD*, pages 1–9. IEEE, 2018.
10. Hernán Ponce de León, Florian Furbach, Keijo Heljanko, and Roland Meyer. Replication package for the Dartagnan tool for SVCOMP 2020. http://dx.doi.org/10.5281/zenodo.3678318, February 2020.
11. Zvonimir Rakamaric and Michael Emmi. SMACK: Decoupling source language details from verifier implementations. In *CAV*, volume 8559 of *LNCS*, pages 106–113. Springer, 2014.