The Static Driver Verifier Research Platform

Thomas Ball, Ella Bounimova, Vladimir Levin
Rahul Kumar, and Jakob Lichtenberg

Microsoft Corporation
http://research.microsoft.com/slam/

Abstract. The SDV Research Platform (SDVRP) is a new academic release of Static Driver Verifier (SDV) and the SLAM software model checker that contains: (1) a parameterized version of SDV that allows one to write custom API rules for APIs independent of device drivers; (2) thousands of Boolean programs generated by SDV in the course of verifying Windows device drivers, including the functional and performance results (of the BEBOP model checker) and test scripts to allow comparison against other Boolean program model checkers; (3) a new version of the SLAM analysis engine, called SLAM2, that is much more robust and performant.

1 Introduction

Static Driver Verifier [1] (SDV) is a verification tool included in the Windows Driver Kit (WDK), using SLAM [4] as the underlying analysis engine. SDV comes with support for three classes of drivers: WDM (The Windows Driver Model); KMDF (Kernel Mode Driver Framework); NDIS (Network Driver Interface Specification). For each of these driver classes, SDV provides a number of class-specific components (for example, API rules and an environment model). API rules are expressed in the SLIC language [5] and describe the proper way to use the driver APIs.

The SDV Research Platform (SDVRP) is a new academic release of SDV that contains a number of features that should be useful to the verification research community:

- **Static Module Verification:** SDVRP enables the development of SLIC rules for APIs independent of device drivers, and the application of SDV to modules that use these APIs. With this feature, researchers can use SDV to verify that clients of an API adhere to the API specification.
- **Boolean Program Repository and Test Scripts:** SDVRP contains thousands of Boolean programs generated by SDV in the course of verifying Windows device drivers, including the functional and performance results of running the symbolic model checker BEBOP [3] on these programs.
- **Slam2 Engine:** SDVRP contains a new version of the SLAM analysis engine (SLAM2) that is much more robust and performant than the first version of SLAM. SDV for Windows 7 uses SLAM2.

SDVRP is available from `http://research.microsoft.com/slam/` under a license for academic use.

T. Touili, B. Cook, and P. Jackson (Eds.): CAV 2010, LNCS 6174, pp. 119–122, 2010.

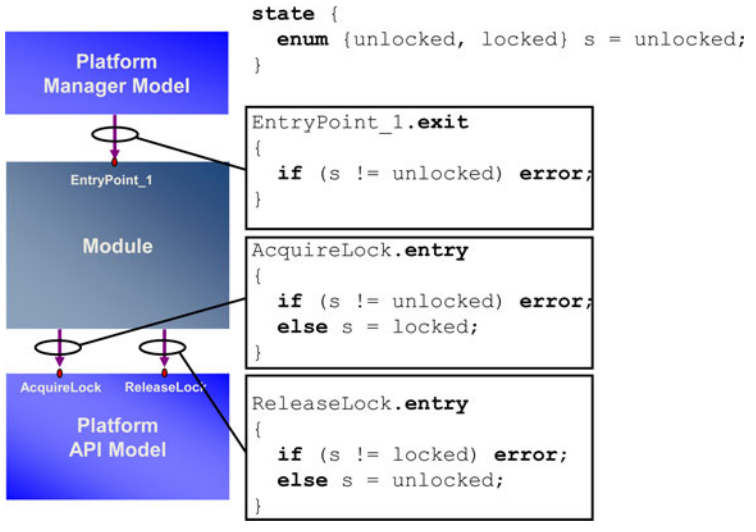

Fig. 1. The interaction between the platform model, a module, and a rule. The platform manager model calls into the entry points of the module. The module itself interacts with the underlying API model, while the rule specifies the safe interactions between the various components.

2 Static Module Verification

In its early days, SDV verified API usage requirements on WDM Drivers. With the success of SDV, came the difficulty of scaling out to other classes of drivers and programs, which in turn motivated the need to parameterize SDV so it could be adapted for other uses.

At a high level, SDV consists of a verification engine, a model of the operating system (platform/environment model), and a set of driver API rules. The verification engine checks whether a user provided driver in the context of the operating system model adheres to the applicable driver API usage rules.

SDVRP generalizes this concept by allowing researchers to provide their own version of the platform model and the API usage rules. Together these two parts comprise a *plugin for static module verification*. The verification engine now checks whether a user provided C module adheres to the plugin API usage rules, in the context of the plugin platform model. This allows SDV to be applied to many other pieces of code besides device drivers.

The platform model itself can be thought of as having two major parts. First, the platform model implements how the platform exercises the module by calling into the module's entry points. This is done by the platform manager model. We can think of this component as the "main" routine of the system. Second, the platform API model provides an implementation of the APIs that the module can use. They are simplified implementations of each platform API that contain behaviors relevant for the verification of associated platform API rules. The

platform model is written using the C language, with one special construct for introducing non-deterministic choice.

Figure 1 shows the interaction between the platform model, the module, and a rule. The platform manager model calls into the entry points of the module. The module itself interacts with the underlying API model, while the rule monitors the interactions between the various components. SLIC [5] rules allow declaration of state as well as state transitions based on API events (call/return). When SDV finds a rule violation, it constructs an error trace that passes through the platform model, the module, and the rule.

Along with the three highly developed plugins for existing driver platform models, SDVRP also comes with a minimal plugin. All of these are available for use, modification, and cloning for research purposes.

3 Boolean Program Repository

SDV/SLAM generates Boolean programs that represent abstractions of C programs, where each Boolean variable represents a predicate on the state of the C program. Boolean programs are an interesting object of study because they admit efficient symbolic model checking, despite the fact that they have recursive procedures. BEBOP [3] is SLAM's model checker for Boolean programs. SDV runs on many drivers, for each driver checking many SLIC rules. A single run of SDV on a driver against a rule can generate many Boolean programs, one for each iteration of the counterexample guided abstraction refinement (CEGAR) process, which successively refines the Boolean program. The SDVRP contains the Boolean programs generated by SDV when run on the drivers in the WDK, as well as the functional and performance results of running BEBOP on these programs. Furthermore, the SDVRP contains the set of test scripts used to generate the results, so that others may easily substitute other Boolean program model checkers in place of BEBOP.

4 Slam2 Engine

SLAM2 improves the precision, reliability, maintainability and scalability of the original SLAM verification engine (SLAM1). SDV 2.0, released with the Windows 7 WDK, uses SLAM2. For SDV 2.0, the true bugs/total bugs ratio is 90-98% on Windows 7 Microsoft drivers, depending on the class of driver. The number of non-useful results (timeouts, "don't know" results) has been reduced greatly. In particular, for drivers shipped as WDK samples, it is 3.5% for WDM drivers and 0.02% for KMDF drivers.

Comparing SLAM2 to SLAM1, on WDM drivers SLAM1 had 19.7% false defects (31/157 reported defects), while SLAM2 had 0.4% (2/512). On WDM drivers, SLAM1 had 6% "give-up" runs (285/4692), while SLAM2 had 3.2% (187/5727). On KMDF drivers SLAM1 had 25% false defects (75/300), while SLAM2 had 0% (0/271). On KMDF drivers SLAM1 had 1% "give-up" runs (31/3111), while SLAM2 had 0.004% (2/5202)

SLAM2 implements a CEGAR loop, which consists of the following main components: a predicate abstraction module, a model checker, and an error trace validation/predicate discovery module. SLAM2 has a new field-sensitive alias analysis with improved precision and performance, and uses the Z3 state-of-the-art SMT solver [6] and new axiomatization of pointer aliasing [2].

The changes in SLAM2 are mostly related to the abstraction, trace validation and predicate discovery functionalities of the CEGAR loop. An abstract intermediate representation (IR) of the input program is introduced as an interface between the low-level IR representing the C program and the CEGAR loop, which permits independence from the input language and from the granularity of abstraction (single statement or multiple statements).

The error trace validation algorithm in SLAM2 is bi-directional with respect to the error trace, combining forward symbolic execution of the error trace (strongest postconditions) with backwards symbolic execution (weakest preconditions). As a result, significant optimization is achieved on long error traces often encountered in the runs on Windows Device Drivers.

Forward execution computes data about the trace (procedure call graph, variable values at each step, pointer aliasing, etc.). The data is used to perform simple feasibility checks on the trace and to optimize subsequent backwards execution and predicate discovery algorithms. Backwards execution is optimized by taking into account data about the trace discovered on the forward pass (for example, program-point-specific pointer aliasing).

SLAM2 implements a new algorithm for discovering Boolean predicates, which is a part of the backwards execution pass. The algorithm is iterative and progresses (on an as-needed basic) from generating a small set of new predicates via computationally cheaper techniques, towards larger sets of predicates via more expensive discovery algorithms.

References

1. Ball, T., Bounimova, E., Cook, B., Levin, V., Lichtenberg, J., McGarvey, C., Ondrusek, B., Rajamani, S.K., Ustuner, A.: Thorough static analysis of device drivers. In: EuroSys, pp. 73–85 (2006)
2. Ball, T., Bounimova, E., de Moura, L., Levin, V.: Efficient evaluation of pointer predicates with z3 smt solver in slam2. Technical Report MSR-TR-2010-24, Microsoft Research (2010)
3. Ball, T., Rajamani, S.K.: Bebop: A symbolic model checker for Boolean programs. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN 2000. LNCS, vol. 1885, pp. 113–130. Springer, Heidelberg (2000)
4. Ball, T., Rajamani, S.K.: The SLAM toolkit. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 260–264. Springer, Heidelberg (2001)
5. Ball, T., Rajamani, S.K.: SLIC: A specification language for interface checking. Technical Report MSR-TR-2001-21, Microsoft Research (2001)
6. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)