講演番号	032

Model Checking of Automotive Software with CBMC

Shinya Miharu [1)] **Wenhung Huang** [1)]

1) DENSO CORPORATION.
1-1 Showa-cho, Kariya, Aichi, 448-8661, Japan

KEY WORDS:**Software and its underlying technologies, modeling language, formal methods**, concurrency bug detection, functional safety

Recent advances in software-defined vehicles (SDVs) have significantly increased the scale and complexity of automotive software. As a result, improving development efficiency while maintaining software quality has become an important challenge. However, current verification processes still rely heavily on manual work, which increases verification costs and the risk that defects escape to later development phases. Automotive software typically decomposes functionality into multiple tasks and utilizes hardware interrupts to ensure real-time performance. If the design or implementation is inadequate, concurrency bugs such as data races on shared variables and undefined behavior may occur. Because these bugs depend on execution timing and involve many possible execution paths, detecting them through conventional testing is difficult. Static analysis tools are widely used in automotive development, but their effectiveness is often limited by the large number of false positives they generate, increasing the effort required for manual triage. Model checking systematically explores execution paths and can detect defects that are difficult to reproduce through testing, but applying it to large industrial software systems remains challenging due to scalability issues and the computational cost of exhaustive analysis. This study investigates the applicability of model checking to automotive software development using CBMC (C Bounded Model Checker), which can directly analyze C and C++ source code with relatively low modeling cost. The study addresses the following research questions: RQ1: What usage scenario enables practical application of model checking in automotive software development? RQ2: Can model checking detect concurrency bugs missed by existing verification processes? RQ3: Can it be operated with fewer false positives and acceptable operational cost?

To address the scalability challenge, we define a practical usage scenario for automotive development processes. In this scenario, the analysis scope is limited to concurrency bugs caused by interrupt timing, which are frequently observed in automotive software. Under this assumption, periodic tasks are assumed to execute in their specified order, while the timing of interrupt occurrence is exhaustively verified. Furthermore, model checking is applied during the software integration test phase, and verification is performed at the module level within a single application rather than analyzing the entire application at once. These constraints reduce the state space and enable practical analysis time. To evaluate the proposed scenario, we applied CBMC to production automotive software that contained a concurrency bug which had escaped detection in the existing development process and was discovered in a later phase. The results addressing the research questions are as follows. For RQ1, the proposed usage scenario significantly reduces computational cost. When verification was attempted on the entire application, the analysis did not complete after more than 40 hours and exceeded available system memory. In contrast, when verification was performed at the module level within the application, the analysis completed in approximately 0.2 hours with 2.7 GB of RAM usage. For RQ2, CBMC successfully detected the concurrency bug contained in the production software under the proposed analysis scenario, demonstrating that model checking can identify concurrency defects missed by the existing verification process. For RQ3, we compared the number of warnings produced by CBMC with those from an existing currently used static analysis tool when analyzing the same module. The currently used static analysis tool reported approximately 500 warnings, whereas CBMC reported 28 warnings, corresponding to about 7% of the warnings reported by the currently used static analysis tool. However, manual effort is still required to classify warnings. In our trial, the triage process required approximately 2.5 hours per warning, resulting in an estimated total effort of about 70 hours. Analysis of the reported cases revealed several contributing factors, including incomplete modeling of RTOS APIs, unconstrained input values caused by tool specifications, and the inability to analyze libraries implemented in assembler. Future work includes increasing validation studies using production automotive software and conducting benchmark-based evaluations to further assess applicability. We also plan to investigate methods for reducing false positives and improving the efficiency of warning triage.

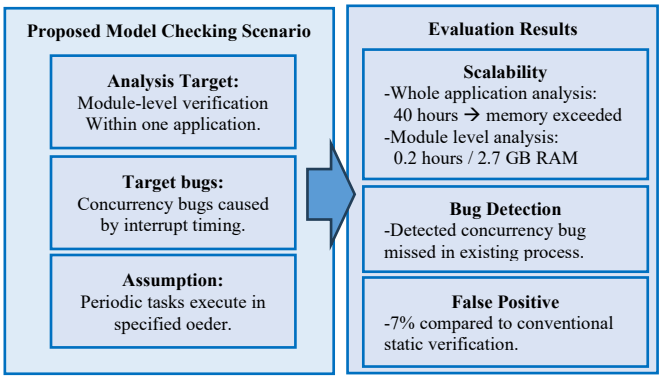

Fig.1 Proposed scenario and verification results