Abstract Interpretation

verification tool technology

Contemporary safety standards (DO-178B, DO-178C, IEC-61508, ISO-26262, EN-50128, etc.) require identifying potential functional and non-functional hazards and demonstrating that the software does not violate the relevant safety goals.
Especially for non-functional program properties – timing, memory usage, absence of runtime errors – tests and measurements can be ineffective, inefficient, and expensive: identifying safe end-of-test criteria is typically undecidable since failures usually occur in corner cases and full test coverage cannot be achieved.

Abstract interpretation based program analysis tools like aiT, StackAnalyzer, and Astrée provide a solution to this problem. Static program analysis is a widely used technique to automatically determine runtime properties of a given program without actually executing it. Abstract interpretation is a semantics based framework for static program analysis that enables the systematic derivation of provably correct analyses: Astrée never fails to report a potential runtime error, aiT never under-estimates the worst-case execution time, StackAnalyzer never under-estimates the worst-case stack usage.

All tools can be smoothly integrated in the development process to detect bugs and errors early when the costs for a fix are lowest and the benefit highest. Dedicated plugins provide a seamless integration in continuous integration frameworks and model-based code generators.

Certification and Qualification

Abstract interpretation based tools like aiT, StackAnalyzer, and Astrée are formal verification tools providing 100% complete and reliable results and are therefore perfectly suited to be used for certification.

The tool qualification process is widely simplified by qualification support kits (QSKs). They specify the tool requirements and the verification test plan and contain an extensible test package. They also provide scripts to execute all test cases and to evaluate and document the results. AbsInt's tool development and software quality process is detailed in a Qualification Software Life Cycle Data report.

Safety and Efficiency.

Always on the safe side with AbsInt

AbsInt Angewandte Informatik GmbH

Tel.: +49 681-383-600
Fax: +49 681-383-6020
info@AbsInt.com
www.AbsInt.com

Founded in 1998, AbsInt is a privately-held company located in Saarbrücken, Germany.
AbsInt provides advanced development tools and tools for validation, verification, and certification of safety-critical software.

AbsInt's tools are designed to:

- enhance software safety,
- speed up time-to-market,
- lower testing and validation costs,
- improve software efficiency.

Consultancy and Services

AbsInt offers consultancy and services in the areas of program analysis, compiler technology, and program validation and verification. AbsInt's specialists perform code analyses of your software projects as a service according to your requirements.

Customers

Our customers belong to the most respected and innovative companies from avionics, automotive, railway, energy, communication, and healthcare technology sectors.

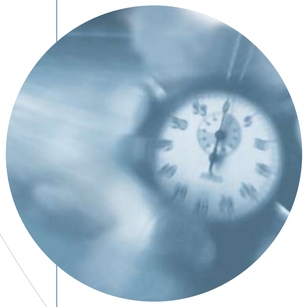
AbsInt's tools are used to validate safety-critical software and to optimize embedded applications. Software products optimized and validated by AbsInt's tools are in daily use by millions of people.

AbsInt

Timing Tools

timing verification - timing optimization

is your program...



...fast enough?

aiT WCET Analyzers statically compute tight bounds for the worst-case execution time (WCET) of tasks in real-time systems. aiT statically analyzes a task's intrinsic cache and pipeline behavior based on formal cache and pipeline models. This enables correct and tight upper bounds to be computed for the worst-case execution time. These bounds are valid for all inputs and each execution of a task. The correct timing behavior can be guaranteed.

TimeWeaver computes worst-case execution time estimates based on real-time traces. TimeWeaver combines observed code snippet execution times with aiT's method to compute longest execution paths.

TimingProfiler statically computes worst-case execution time estimates without the need to repeatedly provide test inputs, execute, and measure. TimingProfiler helps you to identify application parts that cause unsatisfactory execution times. It can be used very early in the development process, when measurements on physical hardware are costly or plain impossible. A typical use is constantly monitoring timing behavior during software development.

StackAnalyzer

memory usage validation

now a thing of the past:

stack overflow

StackAnalyzer automatically determines the worst-case stack usage of the tasks in safety-critical applications. The analysis results are valid for all inputs and each task execution.

Stack memory has to be allocated statically by the programmer. Underestimating stack usage can lead to serious runtime errors which can be difficult to find.

StackAnalyzer directly analyzes binary executables, exactly as they are executed in the final system, and takes the effect of all assembly code, library functions, function pointers, and recursions into account.

Astrée

verifying the absence of runtime errors

did you fix all...

...runtime errors?

Astrée finds all potential runtime errors in C programs. Examples are division by zero, invalid pointer accesses, overflows, data races, and deadlocks. Additionally, Astrée can prove user-defined assertions and detects unreachable code, non-terminating loops, and accesses to shared variables. Floating-point rounding errors are precisely taken into account.

Astrée automatically takes the OS configuration of ARINC-653, OSEK, and AUTOSAR projects into account and can precisely analyze all potential interactions between concurrent threads.

Astrée offers a rule checker which can check for violations of coding rules, supporting MISRA C:2004, MISRA C:2012 incl. Amendment 1, SEI CERT C, CWE, and ISO/IEC 17961:2013.

Astrée is sound. If the analysis does not detect any errors, the absence of runtime errors has been proven.

Astrée has been used successfully to analyze large-scale safety-critical software with zero false alarms.

CompCert

verified compilation

you can trust...

...your compiler!

CompCert is a formally verified optimizing C compiler. Its intended use is compiling safety-critical and mission-critical software written in C and meeting high levels of assurance.

Unlike any other production compiler, CompCert is formally verified, using machine-assisted mathematical proofs, to be exempt from miscompilation issues. In other words, the code it produces is proved to behave exactly as specified by the semantics of the source C program.

"The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors."

Excerpt from: Yang, Chen, Eide, and Regehr. Finding and understanding bugs in C compilers. PLDI 2011

Using the CompCert C compiler is a natural complement to applying formal verification techniques (static analysis, program proof, model checking) at the source-code level. The correctness proof of CompCert guarantees that all safety properties verified on the source code automatically hold for the generated code as well.

Always on the safe side with AbsInt

AbsInt