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**, **Stack**Analyzer, 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. Abstract interpretation amounts to performing a program's computations using value descriptions or abstract values in place of concrete values. One reason for using abstract values instead of concrete ones is computability: to ensure that analysis results are obtained in finite time. Another reason is to obtain results that hold for every program execution and all possible inputs.

aiT, **Stack**Analyzer, and **Astrée** can not only be applied in the late development stage of validation and verification. They also can be smoothly integrated into the development process to detect bugs and errors early when the costs for a fix are lowest and the benefit highest.

Certification and Qualification

Abstract interpretation based tools like **aiT**, **Stack**Analyzer, 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.

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, 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.

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

AbsInt

aiT

timing verification

is your program always ...

... fast enough?

Efficient Verification of Timing Behavior

aiT WCET Analyzers statically compute tight bounds for the worst-case execution time (WCET) of tasks in real-time systems. Testing by repeatedly measuring the execution time of a task is not only tedious, but also typically not safe. It is often impossible to prove that the conditions determining maximum execution time have been taken into account.

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. Valuable development time needs no longer to be spent for extensive timing testing.

aiT can be tightly integrated with many state-of-the-art model-based development tools. aiT's reporting and result visualization features provide valuable feedback for optimizing the worst-case performance or for investigating the timing behavior of code.

aiT has been used successfully for certifying safety-critical software, e.g., according to DO-178B/Level A.

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.

Current safety standards like DO-178B/C, ISO-26262, IEC-61508, or EN-50128, require ensuring that no stack overflows can occur. **Stack**Analyzer proves the absence of stack overflows. AbsInt's Qualification Support Kits enable a tool qualification up to the highest criticality levels.

Astrée

verifying the absence of runtime errors

did you fix all ...

... runtime errors?

Astrée finds all potential runtime errors in C programs. It analyzes safety-critical structured C programs and is also applicable to automatically generated C code.

Astrée is sound. If the analysis does not detect any errors, the absence of runtime errors has been proven. Its powerful analysis engine, flexible parametrization, and comfortable GUI efficiently lead to precise results.

Astrée detects all out-of-bound array accesses, divisions by zero, invalid pointer dereferences and manipulations, arithmetic overflows, floating-point overflows and invalid operations, etc. Floating-point rounding errors are precisely taken into account.

Astrée proves user-defined assertions and detects unreachable code, shared variables, and accesses to uninitialized variables.

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

Always on the safe side with AbsInt.

AbsInt