

Correctness of Software Safety Policies

PROBLEM

Software safety is a complex and ill-defined notion. Past work has concentrated on specific aspects of software safety (e.g. memory access safety) using formal *safety policies*. But:

- How do we actually build a safety policy?
- How can we ensure there are no errors in the safety policy itself?

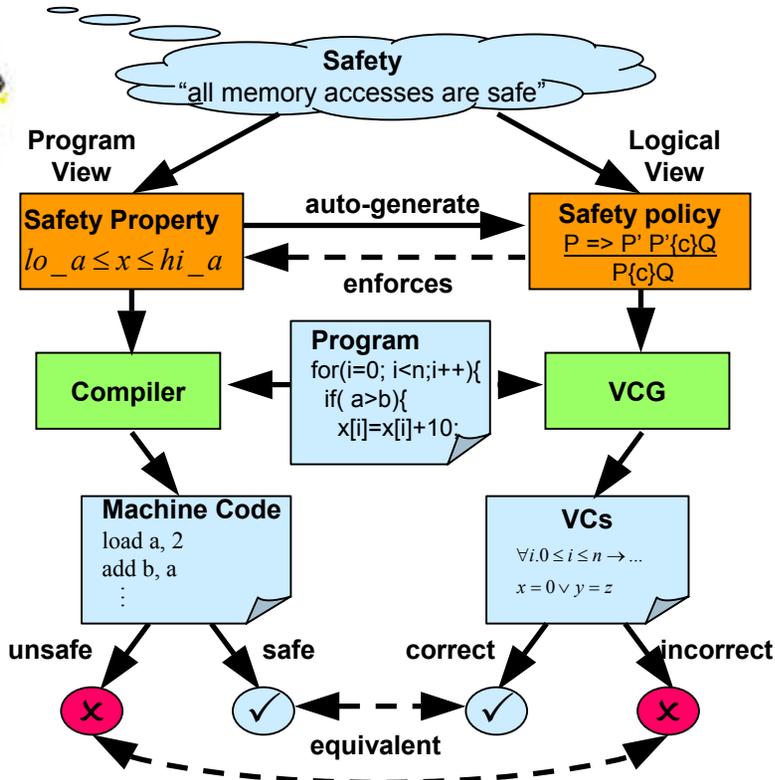


SOLUTION

We distinguish

- *safety properties* -- intuitive but formal definitions of which programs are safe, from
- *safety policies* -- sets of rules enforcing safety.

We can automatically generate a safety policy from a safety property and ensure the correctness of the policy with respect to the property.



TECHNOLOGY

We are developing automated code certification technology to extend our automated code generators.

A generic *Verification Condition Generator* (VCG) then applies an explicit safety policy to a program, returning a set of verification conditions (VCs), which can be checked automatically.

Explanation of Accomplishment



- **POC:** Ewen Denney (ASE Group, Code IC, edenney@email.arc.nasa.gov)
- **Background:** Program certification techniques formally show that programs satisfy certain safety policies. They rely on the correctness of the safety policy which has to be established externally. We have investigated an approach to showing the correctness of safety policies which are formulated as a set of Hoare-style inference rules on the source code level. We have developed a framework which is generic with respect to safety policies and which allows us to establish that proving the safety of a program statically guarantees dynamic safety, i.e. that the program never violates the safety property during its execution.
- **Accomplishment:** We have formalized a selection of safety properties in our framework, and shown that they are sound and complete with respect to a semantic notion of safety. We have developed a generic method of doing this for arbitrary safety properties, thus showing how a safety policy can be automatically derived from a safety property and an operational semantics. This serves as the blueprint for the implementation of a verification condition generator which can be parameterized with different safety policies. The principle difficulty has been finding a general definition of safety property which enables this automatic derivation. We have answered these questions, and thus shown how generic safety policies can be. This is a significant conceptual insight.
Benefits: Safety assurance is a necessary precondition for the application of code generation technology in safety-critical areas. By automating the generation of safety policies, we increase their ease of use and better enable modifications to be made.
- **Future Work:** We will look at the automated generation of safety documents – textual explanations of why a program satisfies a safety policy. This will be useful for Flight Readiness Review (FRR) acceptance of auto-generated software.