**State of the Art in Program Analysis**

- Loop statements are an indispensable construct of imperative programming languages.
- Their expressive power entails undecidable problems, as famously shown by Turing.
- Thus, efficacy of automated program analysis can only be demonstrated on practical problems.

**Challenge**

No established notion of difficulty with regard to handling loops in automated program analysis.

Hence, the occurrence of "simple" and "difficult" loops in practice has not yet been investigated.

Few key insights have vastly increased the scalability of automated program analysis to tackle additional decidable cases from whole new application areas, e.g. for model checking (a verification technique):

- 1962: Inception of model checking
- 1990: Symbolic model checking: Hardware and embedded systems
- 2000: Bounded model checking & CEgar: small pieces of software such as device drivers

**Syntactically Terminating Loops**

**Well-behaved Increment**

Compute the set of possible increments of variable \( i \) along all paths using data-flow analysis.

**Well-behaved Exit Condition**

Syntactically match exit conditions against a set of well-behaved predicates.

---

**Reservation:** Report number of "simple" loops to characterize benchmarks. Currently empirical evaluation is insufficiently comparable, due to the wide range of benchmark programs used.

**Simple Loops**

- We systematically weaken syntactic termination criteria along three dimensions:
  - Control flow: The exiting block may be on any path.
  - Loop-invariance: Allow predicates in > 1 variable.
  - Well-formedness: Predicate is well-formed, but eventual truth not implied by increments.

- We obtain twelve simple loop classes:

**Results**

![Fig. 1: Distribution of various loop classes.](image1)

![Fig. 2: Experimental results on SPEC CPU2006.](image2)

**Contributions**

- We show a termination proof by minimal information and define the class of syntactically terminating loops.
- We systematically weaken syntactic termination criteria to obtain simple loops, a loop pattern commonly used by programmers.
- We show that simple loops indeed represent the majority of loops in our benchmarks.
- We show the various simple loop classes indeed capture the difficulty of automated program analysis.
- We sketch research directions and further applications of loop patterns.

**Programs as Human Artifacts**

- Prevalent treatment of programs: mathematical object
- How to choose good parameterizations? → Leverage structure imposed by programmer!

```java
for (int i = 0; i < N; i++) {
    int b = 0;
    if (f(i)) {
        b = 1;
    } else {
        b = 0;
    }
    // ...
}
```

**Insight**

Use patterns of data flow and control flow to extract such intuitive notions from programs.

These notions are not reflected in the programming language, but attributed by human readers of the program!