U.S. ARMY

Carnegie Mellon

High Confidence Embedded Systems

MAY 2003

Formal Verification

Edmund Clarke, Daniel Kroening

Hunting Down the Bug

Bridging the Gap between

LEGACY CODE and FORMAL SPECIFICATION

MOTIVATION

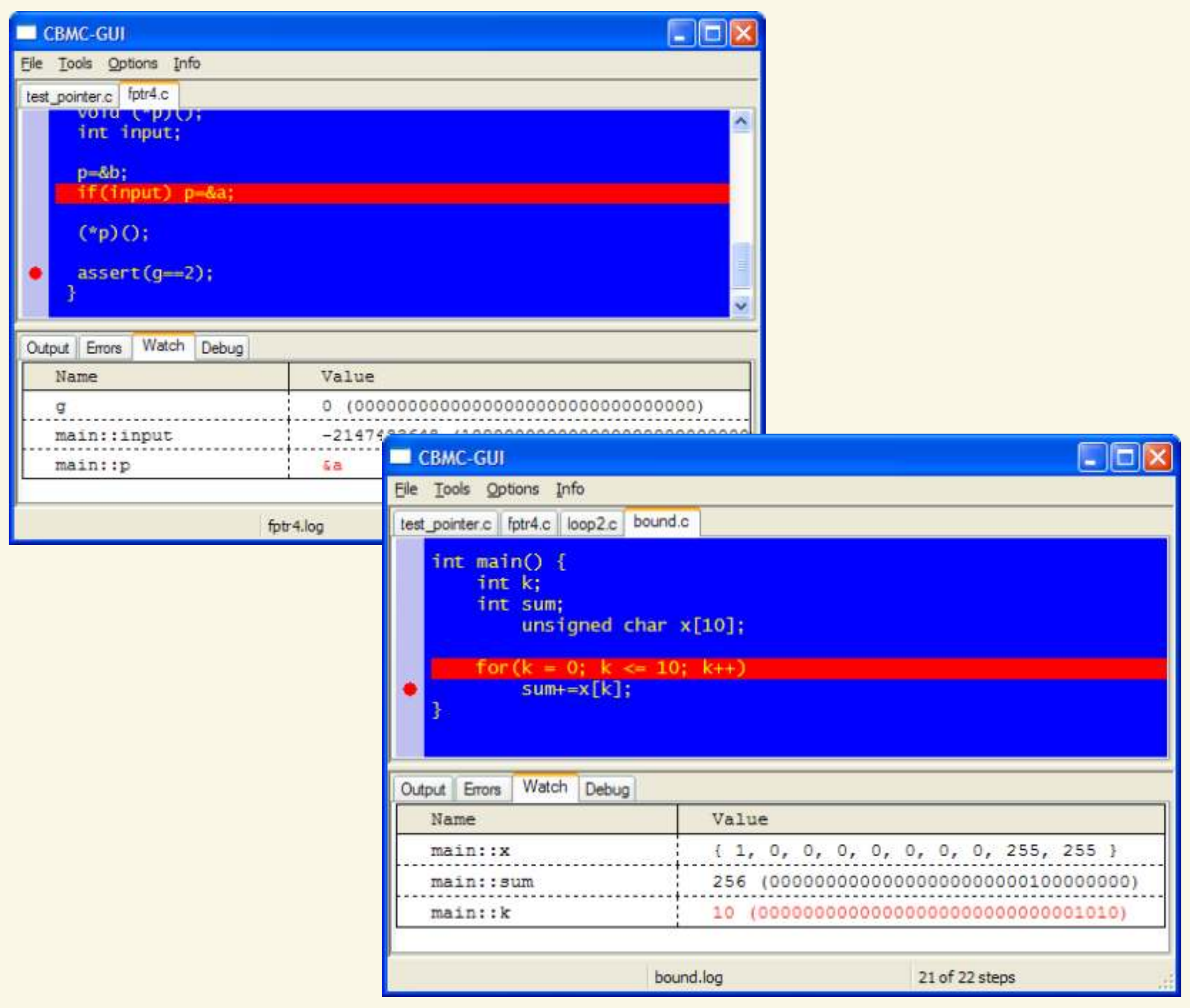
- Software most complex part of today's safety critical embedded systems
- Most embedded systems are legacy designs
- Written in a low level language such as ANSI-C or even assembly language
- Bigger problem than new, high level designs
- Existing tools do not address the verification problem
- Goal: Verify legacy code with respect to
  - A formal specification
  - A high level design
  - Safety properties

> *"Most embedded systems are legacy designs, i.e., written in a low level language such as ANSI-C or even assembly language. These systems are a bigger problem than the new, high level designs, but the existing tools do not address the verification problem."*

ANSI-C BMC

- Problem: Fixpoint computation is too expensive for software
- Idea:
  - Unwind program into equation
  - Check equation using SAT
- Advantages:
  - Completely automated
  - Allows full set of ANSI-C, including full treatment of pointers and dynamic memory
- Properties:
  - Simple assertions
  - Security (Pointers/Arrays)
  - Run time guarantees (WECT)

SCREENSHOT



```
void (*p)();
int input;

p=&b;
if(input) p=&a;

(*p)();

assert(g==2);
}
```

```
Name            Value
g               0 (00000000000000000000000000000000)
main::input     -2147
main::p         &a
```

fptr4.log

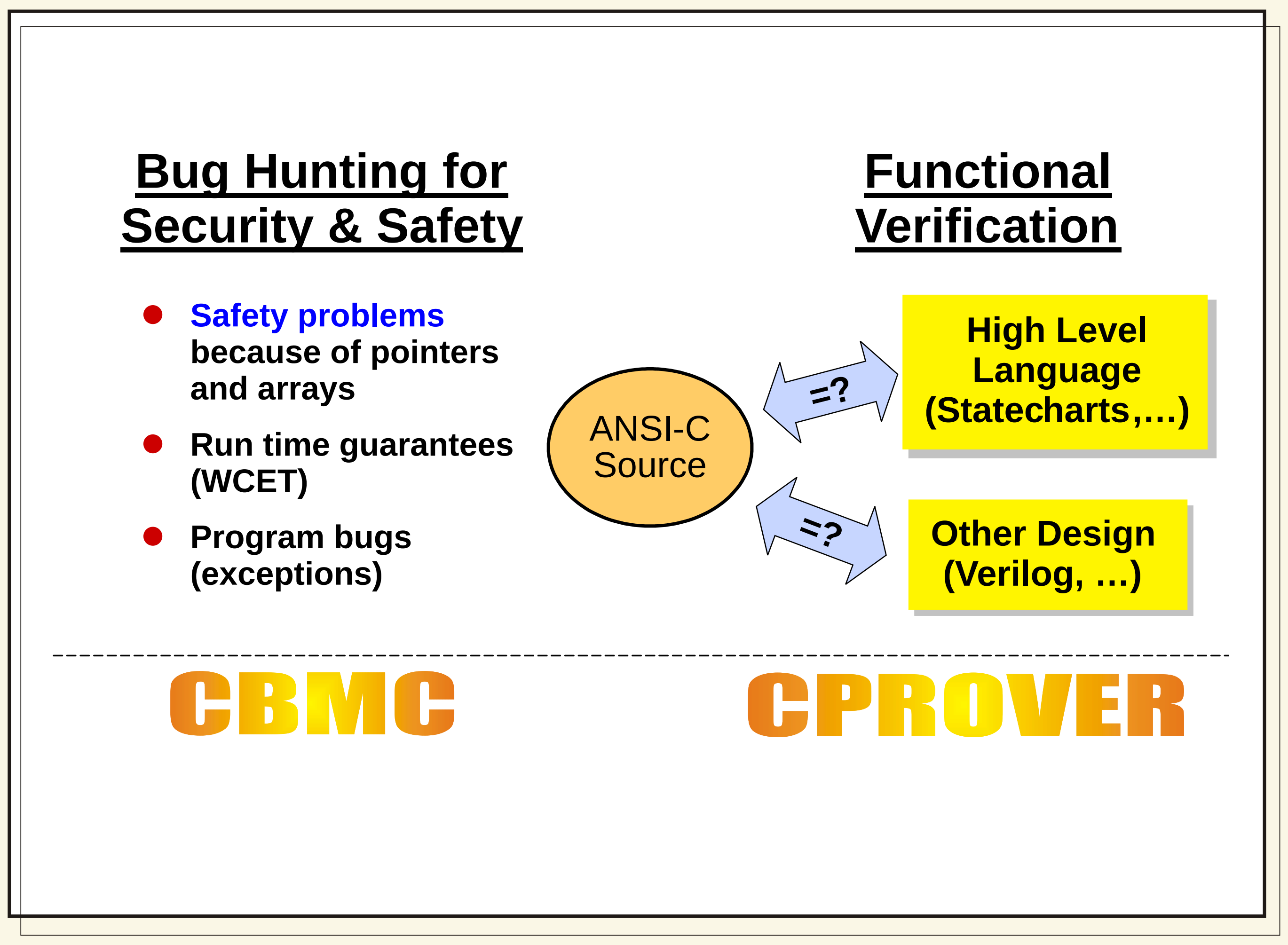
```
int main() {
    int k;
    int sum;
        unsigned char x[10];

    for(k = 0; k <= 10; k++)
        sum+=x[k];
}
```

```
Name            Value
main::x         { 1, 0, 0, 0, 0, 0, 0, 0, 255, 255 }
main::sum       256 (00000000000000000000000100000000)
main::k         10 (00000000000000000000000000001010)
```

bound.log — 21 of 22 steps

CBMC-GUI
File Tools Options Info
test_pointer.c fptr4.c loop2.c bound.c
Output Errors Watch Debug

TOOL OVERVIEW



Bug Hunting for Security & Safety

- Safety problems because of pointers and arrays
- Run time guarantees (WCET)
- Program bugs (exceptions)

Functional Verification

ANSI-C Source
=?
High Level Language (Statecharts,…)
=?
Other Design (Verilog, …)
CBMC
CPROVER

PVS Properties

- Motivation:
  - assertions often not expressive enough
  - E.g.: complex computations
  - One wants specification that can be inspected

    ⇨ We use PVS language
- Problems:
  - Basically everything about PVS language is undecidable, including type consistency
  - PVS language highly compact due to overloading
  - Requires complex resolver and type checker
- Good news: both resolver and type checker implemented (first time outside of PVS!)

D O N E

- Implemented tool that automatically detects
  - Buffer overruns
  - Pointer bugs
  - Worst Case Execution time
  - No false positives, no false negatives!
- Tool takes ANSI-C as input
  - Support for all integer operators
  - Support for complex language features such as side effects
- GUI for easy use by developer
  - Looks and feels like debugger

CURRENT PROJECT

- Verify Safety Properties of a part of a train controller provided by GE
  - Termination / WCET
  - Correctness of pointer constructs
  - The code uses two channels for redundancy: Check that they compute the same result
  - Arithmetic consistency checks, involving multiplication and division
- The code contains x86 assembly language

FUTURE WORK

- Interval abstraction for floating point aritmetic
- Concurrent ANSI-C programs (SpecC)
- Object oriented languages (C++, Java)
- Statechart-like specification language