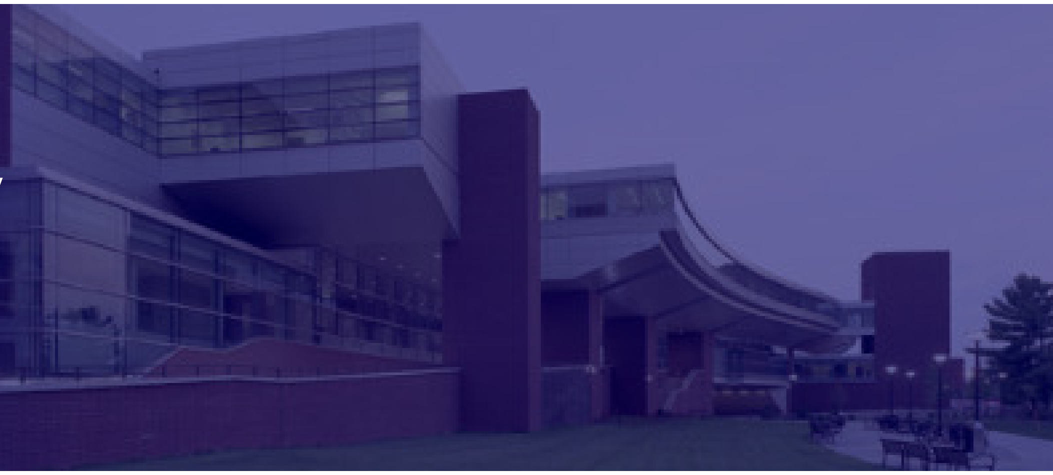


CCured: Type-safe Retrofitting of Legacy Code

By Necula, McPeak, Weimer

Presented By: Philip Koshy

Background

- Circa the 1970s, writing fast code was important
  - This generally required writing assembly code
- UNIX was first written in assembly.
  - They realized they needed something fast and portable.
- C was created by Ken Thompson and Dennis Ritchie as an alternative to assembly
- UNIX was eventually rewritten in C
  - The rest is history

Ken Thompson & Dennis Ritchie

National Medal of Technology,1999

“For co-inventing UNIX and the C programming language”

Why C matters today

- Although application development today is largely done in type safe languages (e.g., Java/C#), there are many legacy C applications and libraries.

- Kernels are still largely written in C.
  - ➢ Linux, Unix, Solaris, Windows

- C code is the foundation for
  - ➢ Billions of dollars of software
    - ➢ Linux kernel is estimated to be worth $700 million in programmer productivity
  - ➢ Millions of lines of code.
    - ➢ Linux kernel has more than 10 million lines of code

What's wrong with C?

- This enormous codebase implicitly comes with all of C's strengths and weaknesses…
- As a design decision in the 1970s, type safety was intentionally sacrificed for flexibility/performance.
  - ➢ At the time, C still needed to win the hearts and minds of assembly programmers.
- The paper says that 50% of CERT advisories (in 2002), were caused by avoidable type safety issues:
  - E.g., Array out-of-bounds, buffer overruns, etc.
- Incorrect pointer usage is at the heart of the problem

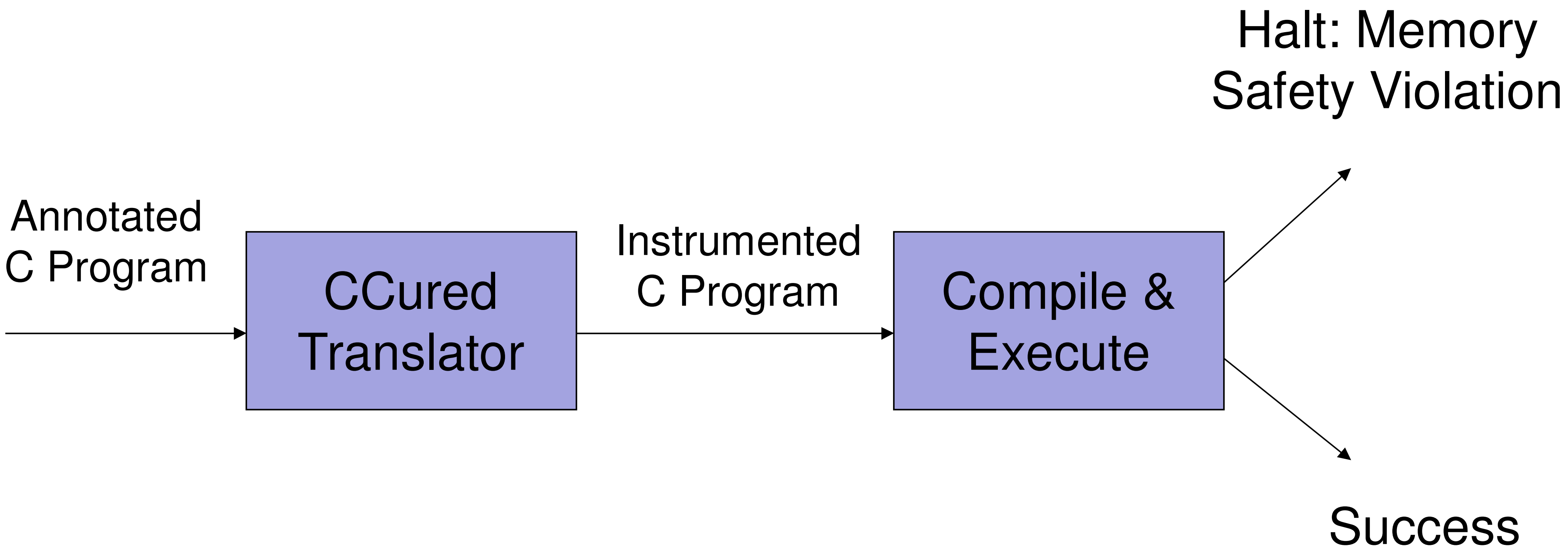
CCured Solution

- Assumption # 1: The majority of pointers in C are used in safe ways, and thus, large portions of legacy programs should be verifiably safe at compile-time.
- With CCured, pointer usage is statically analyzed at compile-time and verified to be type safe.
- For situations where safety cannot be determined at compile time, run-time checks are inserted.

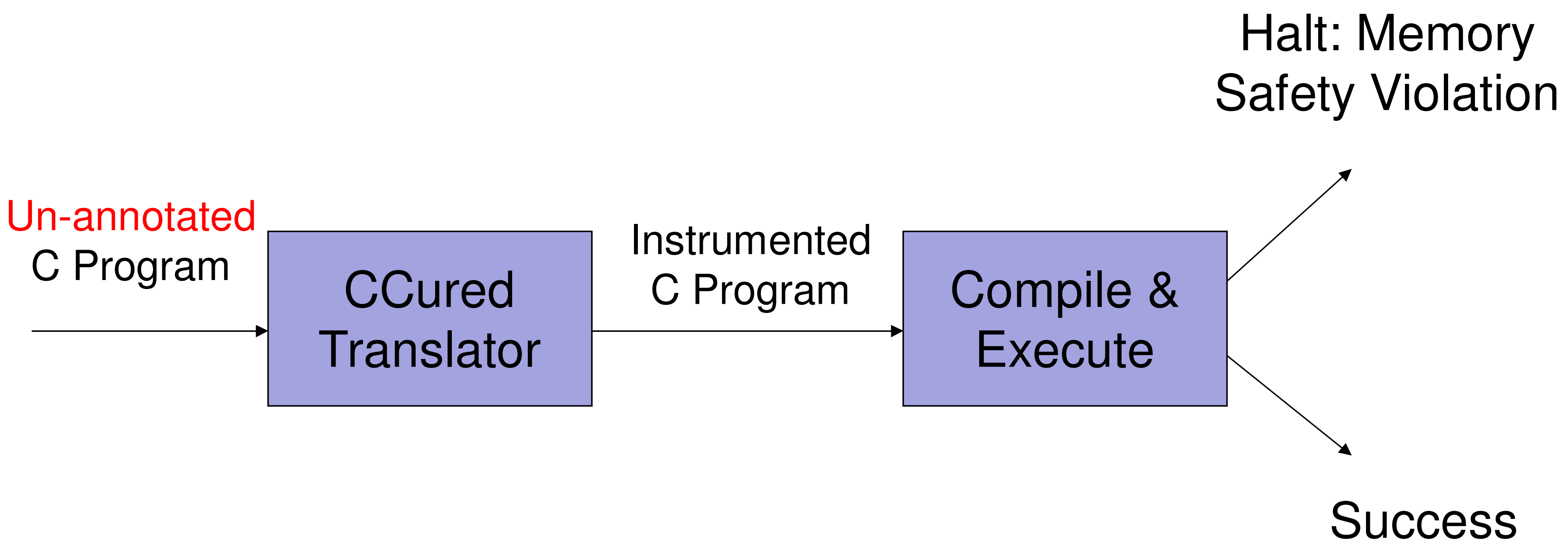
CCured Solution

- Assumption #2: For many, non-critical applications, performance penalties (due to run-time checks) are probably acceptable.
- In performance tests, CCured was between 0 to 150% slower.
  - That’s certainly a wide spread…
  - Is this really acceptable?

Idealized CCured Workflow



Realistic CCured Workflow



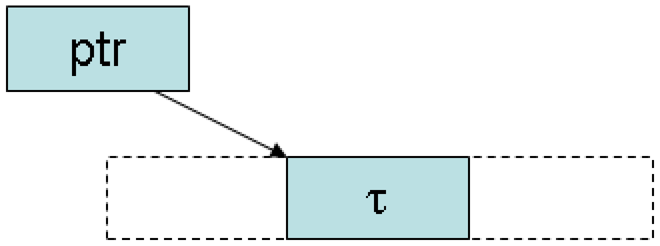
Pointer Usage

Most pointer usage is ‘safe.’ These just need to be checked before dereferencing:

```
int* p = (int*)malloc( sizeof(int) ); // // What if malloc() fails?

if( p == NULL )
          return -1;

*p = 3;

printf( "p is %d\n", *p );
```

SAFE Pointers

SAFE pointer to type τ



Check if the pointer is NULL

If the pointer != NULL, we can dereference it.

This check can be performed statically with CCured.

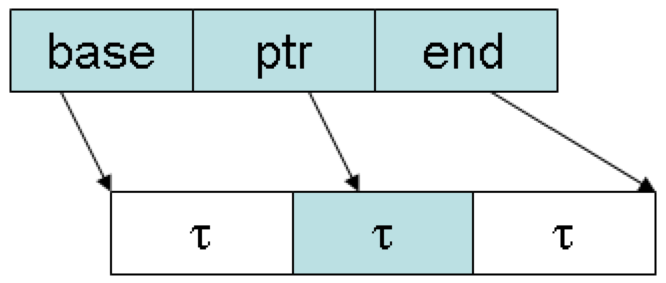
Pointer Usage

It’s possible to perform arithmetic operations on a pointer before dereferencing.

```
int i;
int* array = (int*)malloc( 5 * sizeof(int) );

if( array == NULL )
        return -1;

for( i = 0; i < 5; i++ )
        array[i] = i;

printf( "array[2] is %d\n", *(array + 2) );  // What if we accidently
                                             // step out of bounds?
```

SEQuence Pointers

SEQ pointer to type τ



- In addition to checking if pointer != NULL:
- A "SEQuence" pointer is checked to make sure arithmetic expressions do not move outside an expected bound.
- This check can also be performed statically with CCured.
- The bounds data ('base' and 'end') is stored as metadata alongside the pointer. This creates "fat pointers."

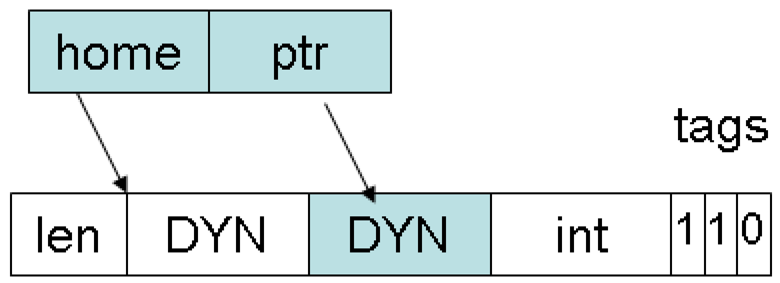
Pointer Usage

We can cast pointers to other types of pointers!

```
int* testValue = (int*)malloc( sizeof(int) );
*testValue = 1;

char* lsb = (char*)testValue; // On the rhs, we cast an int* to a char*
                              // The statically declared type of the lhs
                              // is misleading, due to this cast.
if( *lsb == 1 )
        printf("This is a little-endian system\n");
else
        printf("This is a big-endian system\n");
```

DYNamic (aka WILD) Pointers

DYN pointer



- Any pointer that can point to a heterogeneous type is considered WILD.
- Any pointer obtained through a WILD pointer (either through assignment or deference) must be inferred as WILD.
- This check is be performed at run-time with CCured.
- Note the additional metadata.

A contrived example

```
int *_1 *_2 a;                      // array
int i;                              // index
int acc;                            // accumulator
int *_3 *_4 p;                      // elem ptr
int *_5 e;                          // unboxer
acc = 0;
for (i=0; i<100; i++) {
   p = a + i;                       // ptr arith
   e = *p;                          // read elem
   while ((int) e % 2 == 0) {       // check tag
      e = * (int *_6 *_7) e;        // unbox
   }
   acc += ((int)e >> 1);            // strip tag
}
```

A contrived example

```
1  int *_1 *_2 a;                      // array
2  int i;                              // index
3  int acc;                            // accumulator
4  int *_3 *_4 p;                      // elem ptr
5  int *_5 e;                          // unboxer
6  acc = 0;
7  for (i=0; i<100; i++) {
8      p = a + i;                      // ptr arith
9      e = *p;                         // read elem
10     while ((int) e % 2 == 0) {      // check tag
11         e = * (int *_6 *_7) e;      // unbox
12     }
13     acc += ((int)e >> 1);           // strip tag
14 }
```

a = SEQ

Pointer arithmetic on Line 8

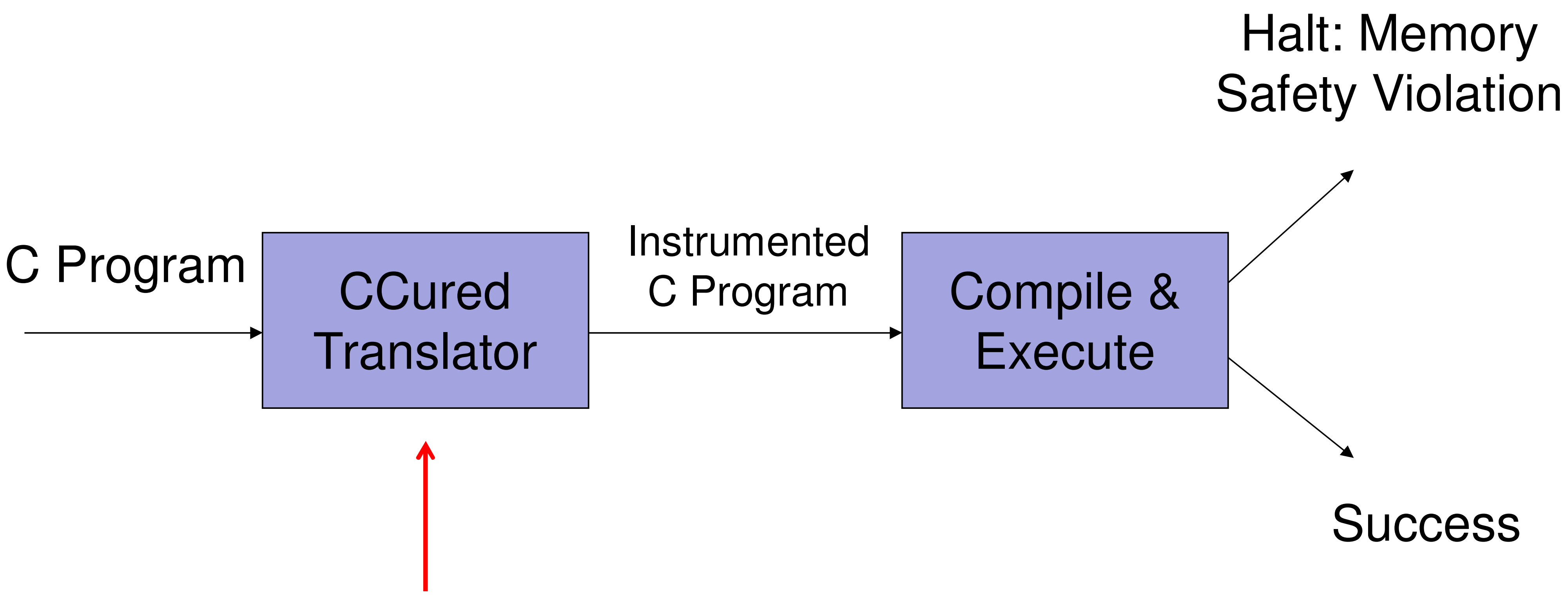
p = SAFE

Simple dereference on line 9

e = WILD

Line 5 says it declared as type (int*) but it is cast in Line 11 as (int**)

Realistic CCured Workflow

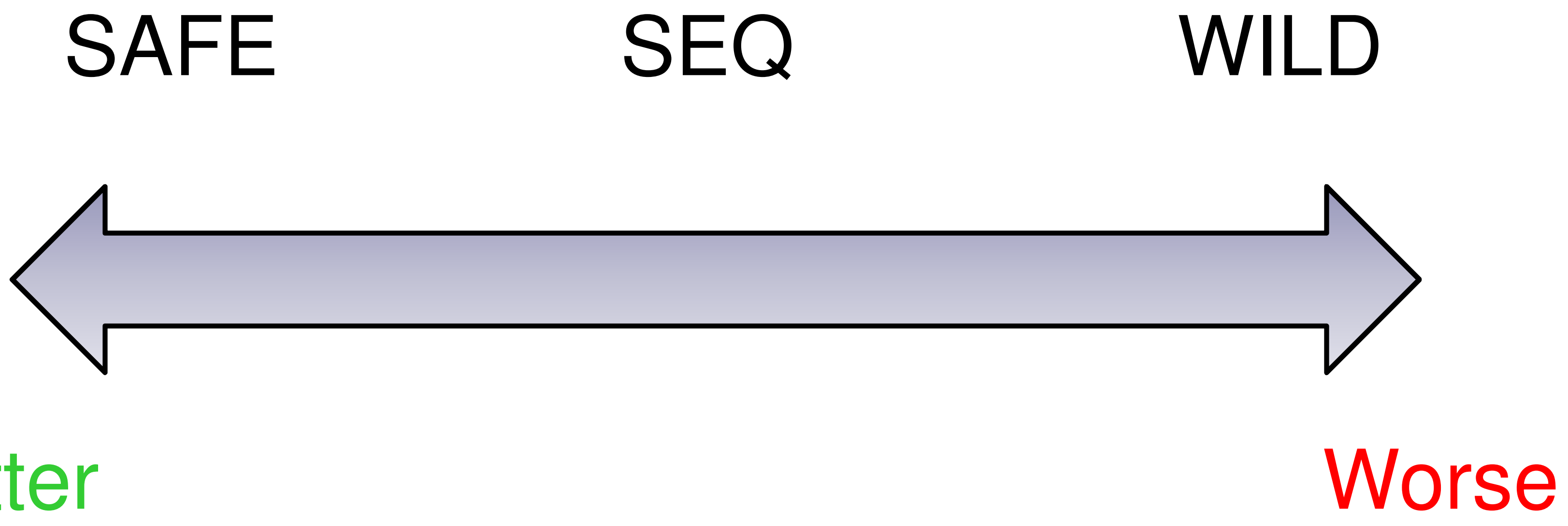


How does CCured infer the pointer type at this stage?

Inference Algorithm

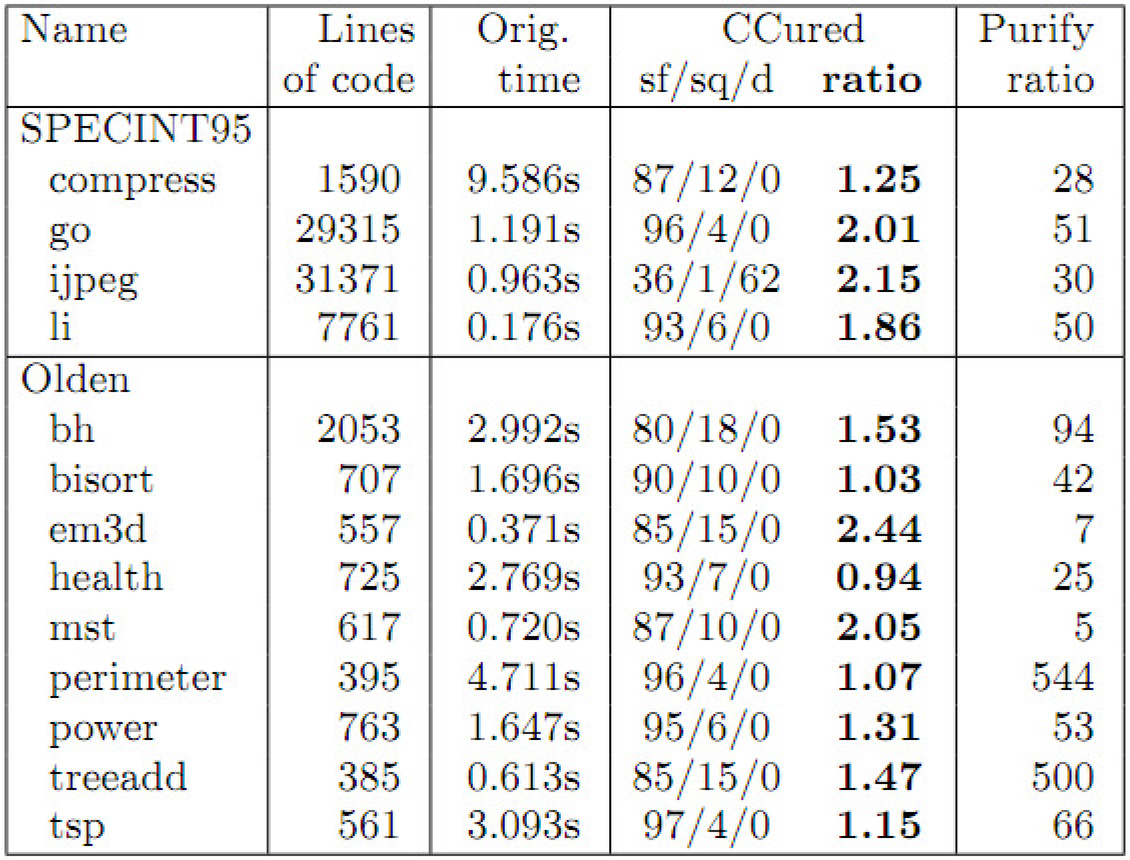
- Inference involves solving a constraint problem
- Any pointer obtained through a WILD pointer (either through assignment or deference) must be inferred as WILD.
  - WILD pointers propagate quickly through programs in this way.
- Otherwise, it is either SEQ or SAFE.
  - If the pointer under consideration is involved in any pointer arithmetic, it is SEQ
  - Otherwise, it is SAFE.

Performance Characteristics



This inference algorithm attempts to maximize the number of SAFE and SEQ pointers.

Performance Results



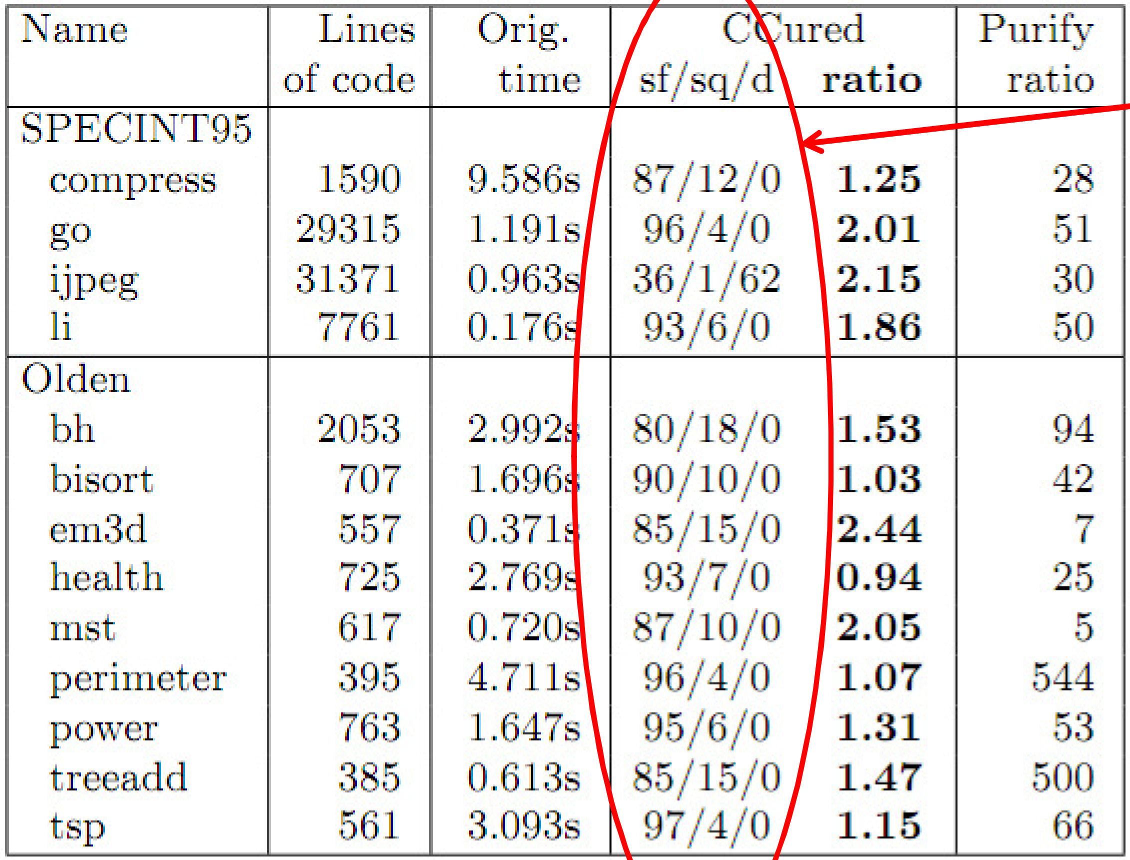
Name	Lines of code	Orig. time	CCured sf/sq/d	CCured **ratio**	Purify ratio
SPECINT95					
compress	1590	9.586s	87/12/0	**1.25**	28
go	29315	1.191s	96/4/0	**2.01**	51
ijpeg	31371	0.963s	36/1/62	**2.15**	30
li	7761	0.176s	93/6/0	**1.86**	50
Olden					
bh	2053	2.992s	80/18/0	**1.53**	94
bisort	707	1.696s	90/10/0	**1.03**	42
em3d	557	0.371s	85/15/0	**2.44**	7
health	725	2.769s	93/7/0	**0.94**	25
mst	617	0.720s	87/10/0	**2.05**	5
perimeter	395	4.711s	96/4/0	**1.07**	544
power	763	1.647s	95/6/0	**1.31**	53
treeadd	385	0.613s	85/15/0	**1.47**	500
tsp	561	3.093s	97/4/0	**1.15**	66

Figure 8: CCured versus original performance. The measurements are presented as ratios, where 2.00 means the program takes twice as long to run when instrumented with CCured. The "sf/sq/d" column show the percentage of (static) pointer declarations which were inferred `SAFE`, `SEQ` and `DYNAMIC`, respectively.

Before performing these tests, the authors applied CCured to the actual test suite (SPECINT95).

They found and fixed several previously undetected bugs.

Performance Results



Name	Lines of code	Orig. time	CCured sf/sq/d	CCured ratio	Purify ratio
SPECINT95					
compress	1590	9.586s	87/12/0	**1.25**	28
go	29315	1.191s	96/4/0	**2.01**	51
ijpeg	31371	0.963s	36/1/62	**2.15**	30
li	7761	0.176s	93/6/0	**1.86**	50
Olden					
bh	2053	2.992s	80/18/0	**1.53**	94
bisort	707	1.696s	90/10/0	**1.03**	42
em3d	557	0.371s	85/15/0	**2.44**	7
health	725	2.769s	93/7/0	**0.94**	25
mst	617	0.720s	87/10/0	**2.05**	5
perimeter	395	4.711s	96/4/0	**1.07**	544
power	763	1.647s	95/6/0	**1.31**	53
treeadd	385	0.613s	85/15/0	**1.47**	500
tsp	561	3.093s	97/4/0	**1.15**	66

Figure 8: CCured versus original performance. The measurements are presented as ratios, where 2.00 means the program takes twice as long to run when instrumented with CCured. The "sf/sq/d" column show the percentage of (static) pointer declarations which were inferred `SAFE`, `SEQ` and `DYNAMIC`, respectively.

Their initial assumption that most pointers are used in a 'safe' way seem to be validated here.

CCured breaks legitimate code

- Due to metadata being stored in “fat pointers,” programmer assumptions about memory may be invalidated.
  - ➢ E.g., sizeof() will no longer works as expected on pointers
- CCured uses its own garbage collection
  - ➢ free()’s are ignored
- Will not work with libraries unless they are recompiled with CCured
  - ➢ If we are dealing with legacy code/libraries, can we assume we have the source code?

CCured breaks legitimate code

```
int* a = (int*)malloc( sizeof(int) );
*a = 5;

// Store the address of ‘a’ into a regular variable
unsigned long addressOfA = (unsigned long)a;

// Cast the variable back to an address and then dereference
int b = *((int*)addressOfA);

printf( "b is %d\n", b );
```