

# Program Transformation

Friday, March 06, 2009  
12:55 PM

Also called program rewriting.

Input C Program -> Rewriter or transformer -> "Bug Free" Program (free from particular bugs that rewriter looks for)

## Desired Properties

- Shouldn't break working code
- Shouldn't affect performance or as small of an effect on performance as possible
- Output should be free of bugs
- Least manual effort required, require little or no changes for the rewriter to operate
- Compatibility between transformed and not transformed code
- Require little or no annotations

## Ccured - Memory Safety

- Safe, seq, dynamic

## Pointer operations in C (\*Gets you in trouble)

- ~~Can point anywhere~~
- Ptrs-to-functions
- Arithmetic\*
- Dereference
- Cast between pointer types\*
- Ptr - to - int casts (implicit conversions)
- Int - to - ptr (implicit conversion)\*
- Assignment/parameter parsing
- &
- Allocation/free
- NULL
- Void\*

## Temporal Memory Bugs

1. p=malloc(10;)  
Free(p);  
\*p=0;
2. Int \*foo(void)  
{  
    Int x;  
    Return &x;  
}  
Void bar(void)  
{  
    Int \*p=foo();  
    \*p=0;  
}

## SAFE

- Ptrs-to-functions
- Dereference
- Ptr-to-int casts
- Assignment/parameter parsing
- &, alloc
- NULL

## Runtime Checks

- None
- Maybe check for NULL

## SEQ (SAFE+arith+int-to-ptr casts)

- Needs bounds checking
- Needs to keep track of bounds
- No type- inconstant aliasing  
i.e.. Always know statically, the type of data pointer to by every pointer

Need: undereferable pointer

- Casts int-to-ptr
- Yield an undereferable ptr.

## Pointer representation

<l,h,p>  
<0,0,p>

Void foo(int \*p) //SAFE

```
{
    *p=0;
    *p=1;
    *p=2;
}
```

Void bar (void); //SEQ

```
{
    Int *p;
    p=a;
    p=p+3;
    Foo(p);
}
```

```
Void * memcpy (void * dest, void * src, int sz)
Int *p;
Int *q;
.
.
Memcp((Void*)q,(void*)p,sizeof (*q));
.
.
Int foo(int & x)
{
    x=5;
}
```

Struct s {	Struct P{
Int x;	Int *p;
};	};

```
Struct S;
Struct P *p;
s->x=<blah>;
p=(struct P *Rs;
*(p->p)=0;
```

```
*p=0
↓ Rewrite
If(p<l | |p>h)
Abort();
*p=0;
```

## Conversion from SEQ to SAFE

```
p~<&A,&A[4],p>
Boundscheck(<&A,&A[4],p>)
Foo(p);
```