Amorphous Program Slicing

Dave Binkley
Loyola University Maryland

24 March 2014

A Slice

```
sum = 0
prod = 1
i = 1

while (i < 10)
    sum += i
    prod *= i
    i++
endwhile

print sum
print prod
```

```
sum = 0

i = 1

while (i < 10)
    sum += i

    i++
endwhile

print sum
```

Slice Defined

- Given a program P, a statement *s*, and a variable *v, the slice of P taken with respect to* <*s,v*> is those parts of P necessary to compute the same values for v at s.

A Slice - Those parts of P necessary to compute the same values for <11, sum>

```
1     sum = 0                    sum = 0
2     prod = 1
3     i = 1                      i = 1
4
5     while (i < 10)             while (i < 10)
6         sum += i                   sum += i
7         prod *= i
8         i++                        i++
9     endwhile                   endwhile
10
11    print sum                  print sum
12    print prod
```

Two Requirements

- Syntactic
  - subset of the original program's syntax
- Semantic
  - same values for the selected variable

Syntactic Requirement

```
1     sum = 0                    sum = 0
2     prod = 1
3     i = 1                      i = 1
4
5     while (i < 10)             while (i < 10)
6         sum += i                   sum += i
7         prod *= i
8         i++                        i++
9     endwhile                   endwhile
10
11    print sum                  print sum
12    print prod
```

Semantic Requirement

```
sum = 0
prod = 1
i = 1

while (i < 10)
    sum += i
    prod *= i
    i++
endwhile

print sum
print prod
```

```
sum = 0

i = 1

while (i < 10)
    sum += i

    i++
endwhile

print sum
```

A (Syntax Preserving Static Backward) Slice

```
sum = 0
prod = 1
i = 1

while (i < 10)
    sum += i
    prod *= i
    i++
endwhile

print sum
print prod
```

```
sum = 0

i = 1

while (i < 10)
    sum += i

    i++
endwhile

print sum
```

An *Amorphous* Slice

```
sum = 0
prod = 1
i = 1

while (i < 10)
    sum += i
    prod *= i
    i++
endwhile

print sum
print prod
```

```
print 45
```

Two Requirements

- Syntactic
  - “better” (smaller, faster, etc.)
- Semantic
  - same values for the selected variable

An Implementation

- (Syntax Preserving Static Backward) Slice
- Transform
- Repeat while changes

Slice 1

```
sum = 0
prod = 1
i = 1

while (i < 10)
    sum += i
    prod *= i
    i++
endwhile

print sum
print prod
```

```
sum = 0

i = 1

while (i < 10)
    sum += i

    i++
endwhile

print sum
```

Transform 1 (loop unroll)

```
sum = 0

i = 1

while (i < 10)
    sum += i

    i++
endwhile

print sum
```

```
sum = 0
sum += 1
sum += 2
sum += 3
sum += 4
...
sum += 9
print sum
```

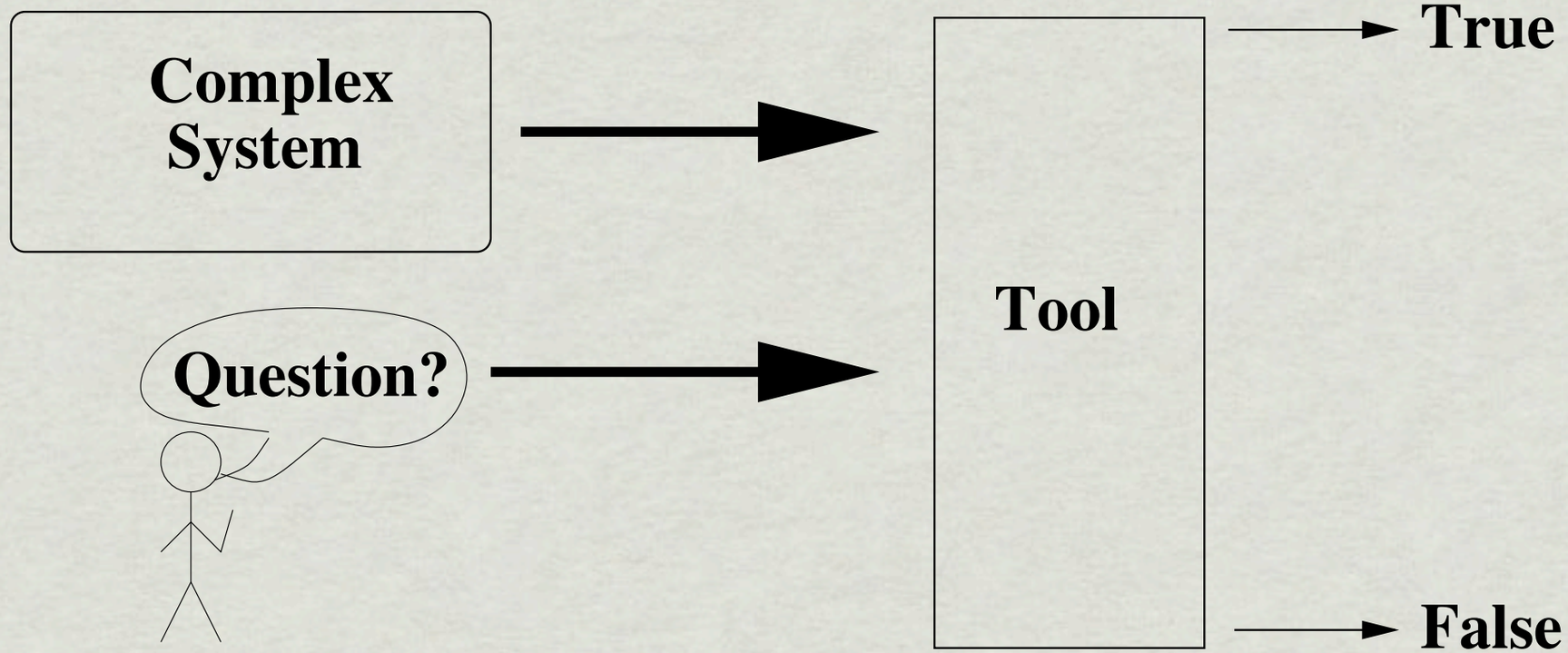
Transform 2 (Constant Propagation)

```
sum = 0
sum += 1
sum += 2
sum += 3
sum += 4
...
sum += 9
print sum                    print 45
```

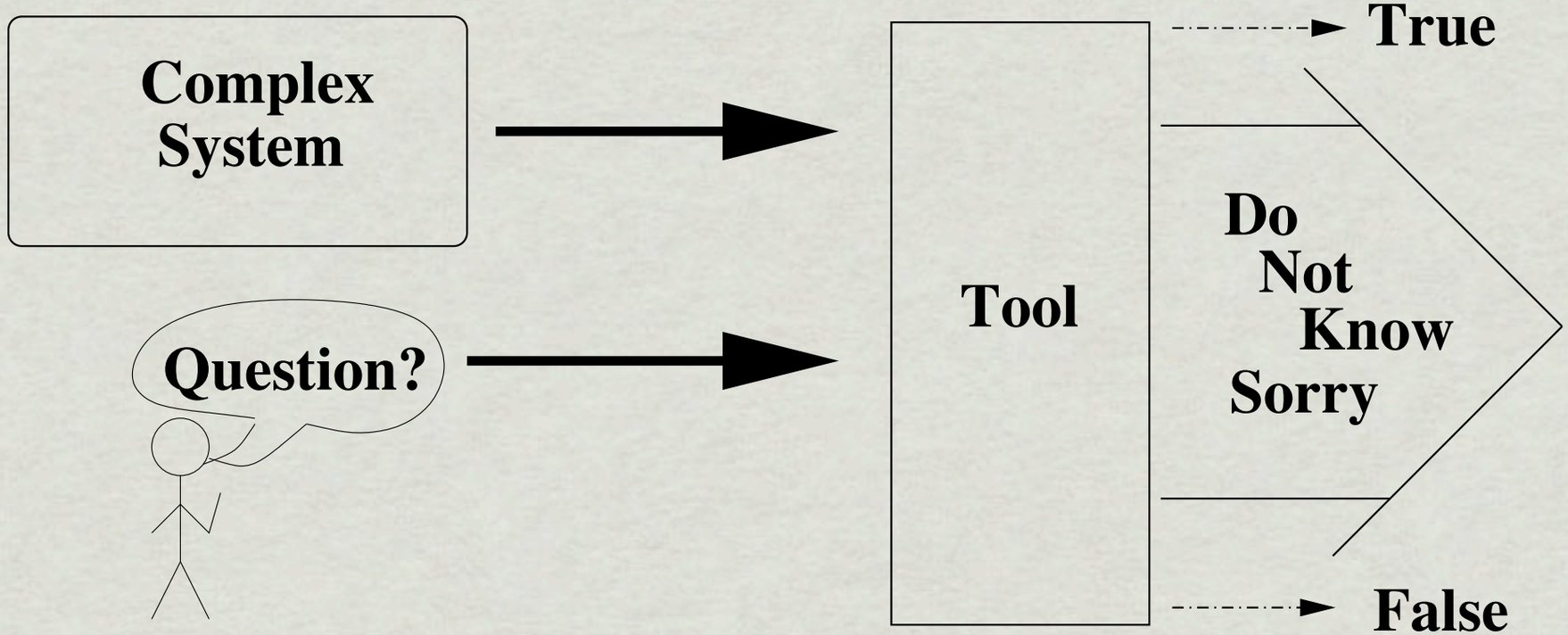
A Use

- Goal: Answer the unsolvable

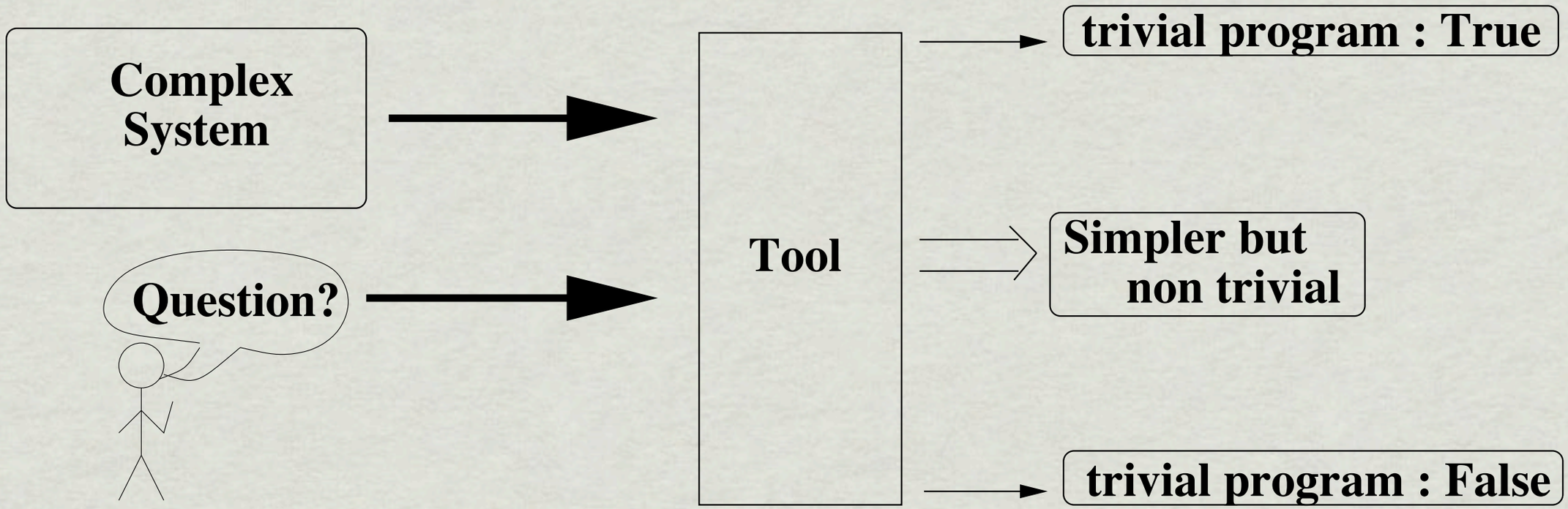
An Ideal Tool



Reality



Amorphous Slicing Way



Approach

- Make question explicit in a program variable
- Compute an amorphous slice w.r.t that variable

Example Question

- Is program **P** free from array bounds violations?

Here's a **P**

```
int result[128];
char source[MAX];
printf("Enter a  string");
scanf("%s", source);
i = 0;
while (i < 128)
{
    result[i] = 0;
    i = i + 1;
}

i = 0;
while (source[i] != '\0')
{
    x = source[i];
    i = i + 1;
    result[x] = result[x] + 1;
}
```

```
b = 0;
bc = 0;
i = 'a';
while (i <= 'z')
{
    if (result[i] > bc)
    {
        b = i;
        bc = result[i];
    }
    i = i + 1;
}
printf("%c occurs
            %d times\n",
             b, bc);
```

Approach

- Make question explicit in a program variable
- Compute an amorphous slice w.r.t that variable

Step 1 - Make Explicit

```
int result[128];
char source[MAX];
safe = true;       /* starts safe */
printf("Enter a string");
scanf("%s", source);
i = 0;
while (i<128)
{
    safe = safe && i>=0 && i<128;
    result[i] = 0;
    i = i + 1;
}
```

```
i = 0;
safe = safe && i>=0 && i<MAX;
while (source[i] != '\0')
{
    safe = safe && i>=0 && i<MAX;
    x = source[i];
    i = i + 1;
    safe = safe && x>=0 && x<128;
    safe = safe && x>=0 && x<128;
    result[x] = result[x] + 1;
    safe = safe && i>=0 && i<MAX;
}
...
printf("safe = %d\n", safe);
```

Step 2 - Slice

```
int result[128];
char source[MAX];
safe = true;       /* starts safe */
printf("Enter a string");
scanf("%s", source);
i = 0;
while (i<128)
{
    safe = safe && i>=0 && i<128;
    result[i] = 0;
    i = i + 1;
}
```

Amorphous Slicing iterates
(Syntax Preserving) Slicing
and
Transformation

Initial slice has little effect!

Initial Transformation

```
int result[128];
char source[MAX];
safe = true;       /* starts safe */
scanf("%s", source);
i = 0;
while (i<128)
{
    safe = safe && i>=0 && i<128;
    result[i] = 0;
    i = i + 1;
}
```

```
...


i = 0
while (i<128)
{
    safe = safe && i>=0 && i<128;
    i = i + 1;
}

i = 0
while (i<128)
{
    result[i] = 0;
    i = i + 1;
}
```

Second Slice

~~int result[128];~~
char source[MAX];
safe = true; /* starts safe */
scanf("%s", source);

...

i = 0
while (i<128)
{
  safe = safe && i>=0 && i<128;
  i = i + 1;
}

~~i = 0~~
~~while (i<128)~~
~~{~~
~~ result[i] = 0;~~
~~ i = i + 1;~~
~~}~~

Second Transformation

```
char source[MAX];
safe = true;       /* starts safe */
scanf("%s", source);

i = 0
while (i<128)
{
    safe = safe && i>=0 && i<128;
    i = i + 1;
}
```

```
char source[MAX];
safe = true;       /* starts safe */
scanf("%s", source);

safe = safe && 0>=0 && 0<128;
safe = safe && 127>=0 && 127<128;
```

Round Three

```
char source[MAX];
safe = true;       /* starts safe */
scanf("%s", source);

safe = safe && 0>=0 && 0<128;
safe = safe && 127>=0 && 127<128;
```

```
char source[MAX];
scanf("%s", source);

safe = true;
```

Middle of Code

```
i = 0;
safe = safe && i>=0 && i<MAX;
while (source[i] != '\0')
{
    safe = safe && i>=0 && i<MAX;
     x = source[i];
     i = i + 1;
    safe = safe && x>=0 && x<128;
    safe = safe && x>=0 && x<128;
     result[x] = result[x] + 1;
    safe = safe && i>=0 && i<MAX;
}
```

```
safe = safe && 0>=0 && 0<MAX;
i = 0;
while (source[i] != '\0')
{
    safe = safe && i>=0 && i<MAX;
    safe = safe && source[i]>=0
                && source[i]>=0
                && source[i]<128;
                && source[i]<128;
                && i+1>=0
                && i+1<MAX;
     x = source[i];
     i = i + 1;
     result[x] = result[x] + 1;
}
```

Middle - Round 2

```
safe = safe && 0>=0 && 0<MAX;
i = 0;
while (source[i] != '\0')
{
    safe = safe && i>=0 && i<MAX;
    safe = safe && source[i]>=0
                && source[i]<128;
                && source[i]>=0
                && source[i]<128;
                && i+1>=0
                && i+1<MAX;
    x = source[i];
    i = i + 1;
    result[x] = result[x] + 1;
}
```

```
safe = safe;
i = 0;
while (source[i] != '\0')
{
    safe = safe && i>=0
                && i<MAX
                && source[i]>=0
                && source[i]<128;
               && i+1>=0
                && i+1<MAX;
    x = source[i];
    i = i + 1;
    result[x] = result[x] + 1;
}
```

Final Amorphous Slice

```
char source[MAX];
scanf("%s", source);
safe = true;

i = 0;
while (source[i] != '\0')
{
    safe = safe && i>=0
                && source[i]>=0
                && source[i]<128;
                && i+1<MAX;
    i = i + 1;
}

printf("%d\n”, safe);
```

So, Is **P** Safe?

```
int result[128];
char source[MAX];
printf("Enter a string");
scanf("%s", source);
i = 0;
while (i < 128)
{
    result[i] = 0;
    i = i + 1;
}
```

```
i = 0;
while (source[i] != '\0')
{
    x = source[i];
    i = i + 1;
    result[x] = result[x] + 1;
}
```

```
b = 0;
bc = 0;
i = 'a';
while (i <= 'z')
{
    if (result[i] > bc)
    {
        b = i;
        bc = result[i];
    }
    i = i + 1;
}
printf("%c occurs
        %d times\n",
         b, bc);
```

Middle Loop of **P**

```
i = 0;
while (source[i] != '\0')
{
    x = source[i];
    i = i + 1;
    result[x] = result[x] + 1;
}
```

```
i = 0;
while (source[i] != '\0')
{
    safe = safe && i>=0
                && source[i]>=0
                && source[i]<128;
                && i+1<MAX;
    i = i + 1;
}
```

Is **P** safe? ... Kind of

```
i = 0;
while (source[i] != '\0')
{
    safe = safe && i >= 0
```

Tool limitation

```
                && source[i]>=0
                && source[i]<128;
```

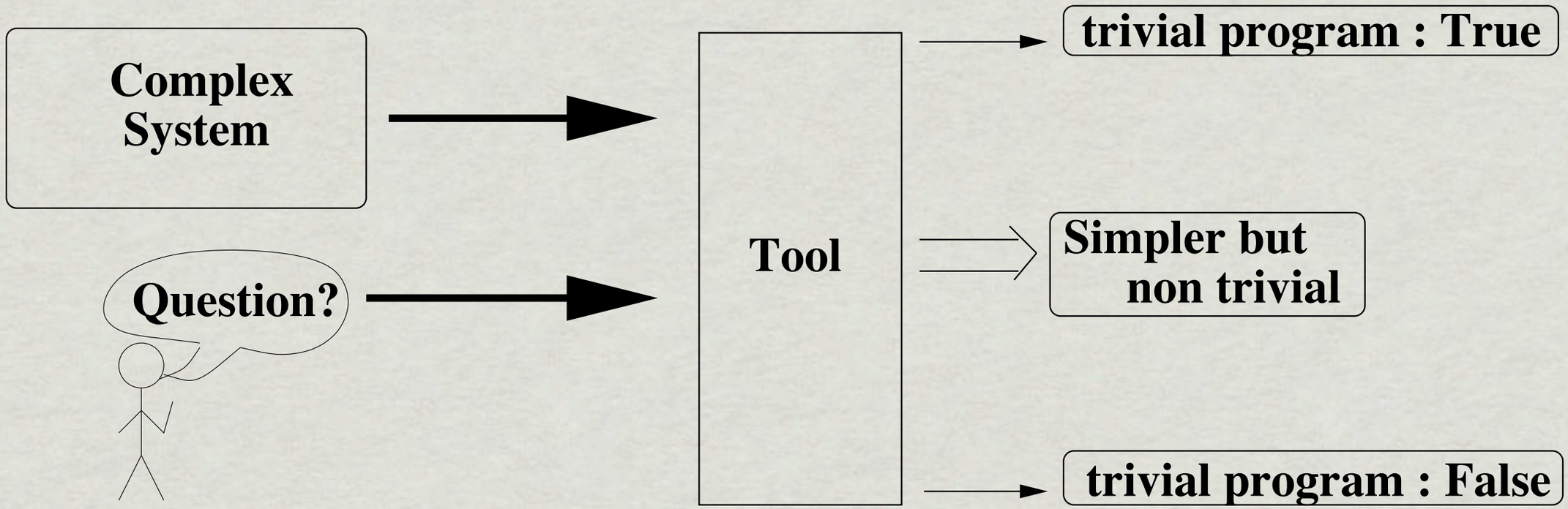
Input character outside 0 ...127

```
                && i+1<MAX;
```

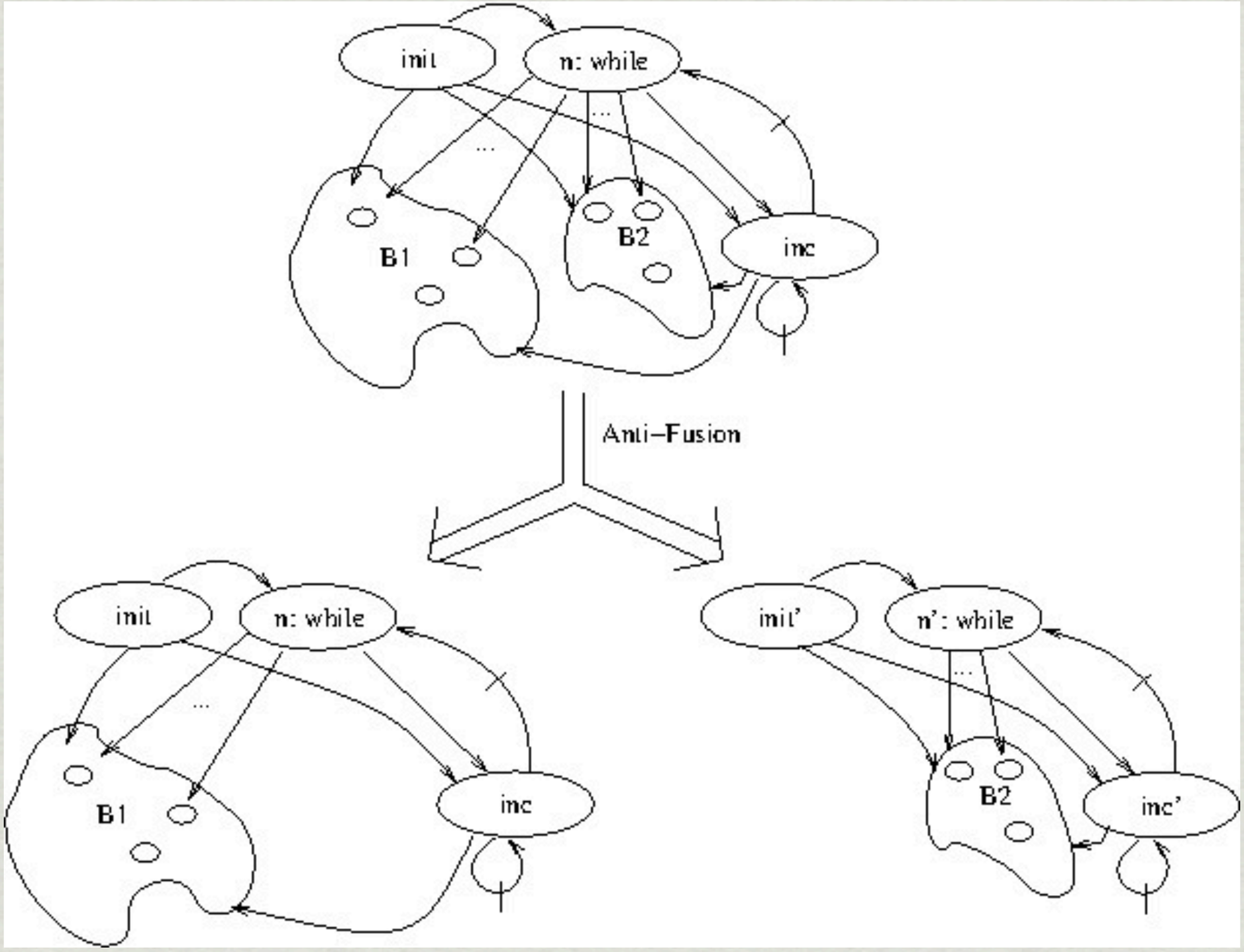
String too long

```
    i = i + 1;
}
```

Amorphous Slicing Way



Challenges - Anti Fusion



Other Challenges

- side effects
  - x = i++
  - y = x * x
- which naively transforms into
  - y = (i++) * (i++)

Amorphous Slicing of Functional Programs

- Potential
  - Exploit existing transformations
  - Develop Amorphous Slicing specific transformations
- Challenges
  - Dependence tracking in higher order functions

Avoiding “Something Complex”

- let
  - fun append a b = fold a b cons tail
  - fun length l = fold l 0 (λx. λy. x+1) tail
  - val element = “something complex”
  - val list1 = cons element ...
  - val list2 = append list1 ...
- in
  - length list2

Question Is

- Do functional programing languages make better targets for amorphous slicing?

M. Harman, D. Binkley, S. Danic. *Amorphous Program Slicing*. Journal of Systems and Software, Volume 68, Issue 1, pages 45-64.