Function Extraction for Malicious Code: The FX/MC Project

David A. Mundie
Network System Survivability
Software Engineering Institute
(dmundie@cert.org)

Sponsored by the U.S. Department of Defense

Background

Dave Mundie's View of CERT

- Long-term: survivability research
- Medium-term: process improvement
- Short-term: operations
  - Incidents
  - Vulnerabilities
  - Artifacts

The Problem

- Malware authors write really terrible code
  - And it’s getting worse
- Reverse engineering is hard
- Burneye
  - Almost a year
- Has to be redone over and over
- The IETF RFC - “evil bit”
- Wouldn’t it be nice if they wrote in Haskell!

National Software Dependencies

- Software controls the nation’s infrastructures
  - Embedded security
- Malicious code can have catastrophic effects
- Risks to government, defense, and economy
- Offshore development increases uncertainty

Malicious Code

- Easy to insert
- Easy to hide
- Hard to analyze
  - The guy next door
  - You want to do this?
- Little technology to help

State of Practice in Software Analysis

- Unlike other engineering disciplines, software engineering has no practical means to fully evaluate the expressions it produces

- No programmer can say for sure what a complex program does in all uses

- Programmed but unknown functionality is the Achilles heel of software

- Malicious code is unknown functionality

Malicious Code Analysis

- Automation is essential
  - Only path to scale up and speed up
- Full behavior must be analyzed
  - All functionality must be known
- Continual analysis is required
  - Code inserted a year ago or yesterday

FX/MC System I

- Understanding malicious code today
  - labor intensive
  - error-prone human process
  - human time scale

- Understanding malicious code with FX/MC
  - machine intensive
  - precise computational process
  - CPU time scale

- FX/MC Strategy
  - compute the functional behavior of malicious code
  - basis is function-theoretic foundations of software

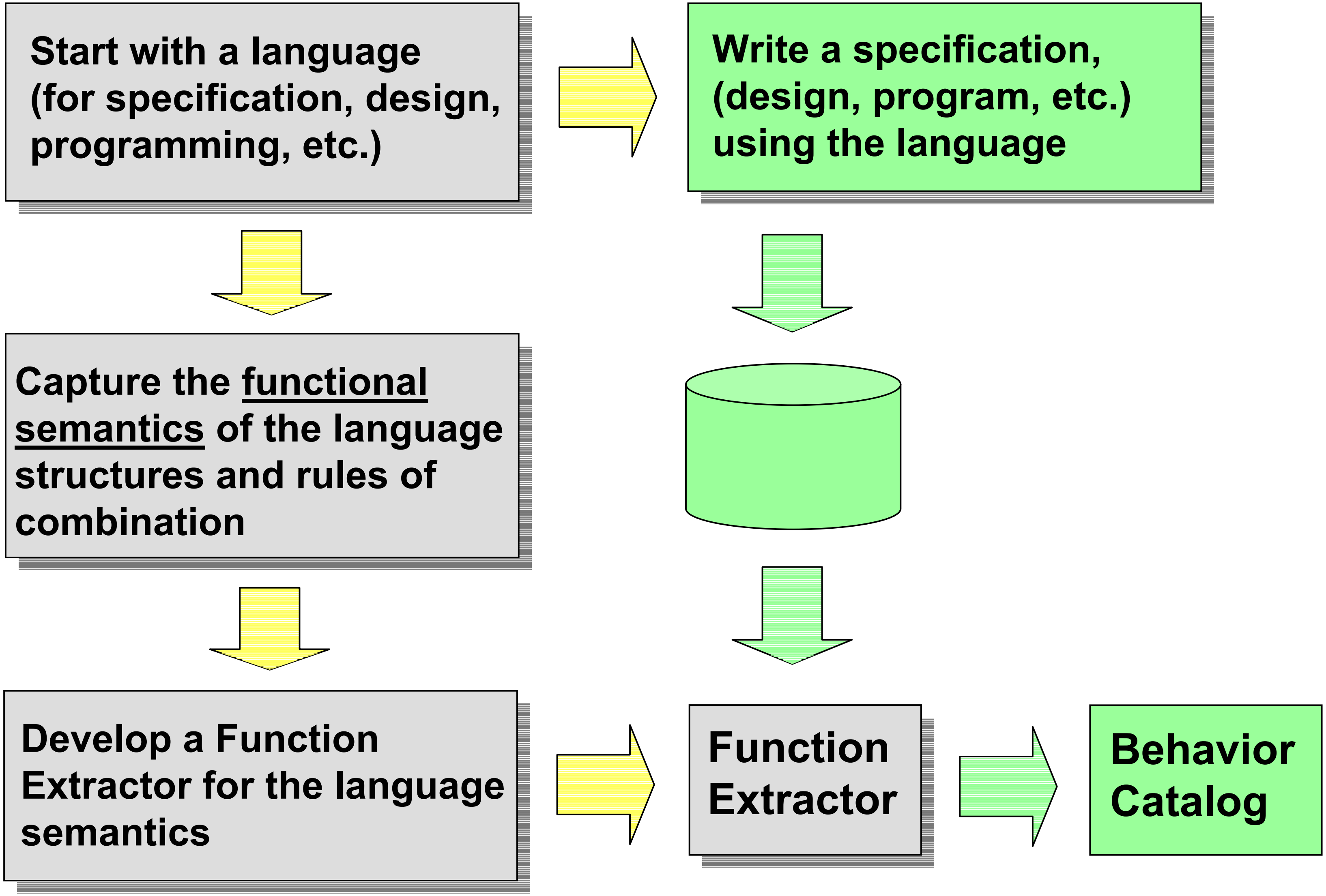
FX/MC System II

FX/MC will provide these capabilities on the desktop for analyzing malicious code in Intel Assembler Language:

- Control Flow Deobfuscation
  - calculate the true structured control flow despite obfuscation
  - eliminate complexities of code structure and sequencing
- Function Extraction
  - calculate the functional behavior of code
  - determine what the code does – its net effect
- Function Comparison
  - compare two code blocks for functional equivalence
  - determine if new code is a disguised version of old code

Foundations of Function Extraction Technology

The FX Idea – Reclaim Semantic Knowledge



Start with a language (for specification, design, programming, etc.)

Write a specification, (design, program, etc.) using the language

Capture the functional semantics of the language structures and rules of combination

Develop a Function Extractor for the language semantics

Function Extractor

Behavior Catalog

Function Extraction Technology

- Programs and their control structures implement mathematical functions or relations
- Control structure functions can be extracted in a stepwise process with mathematical precision
- FX secret weapons
  - Structure Theorem defines structuring process
  - Function Theorem defines extraction process
  - termination assured by finite number of structures
  - behavior language need not be executable

Sequence Program Function

- **Sequence example:**

```
do
    a  := abs(b)
    d := max(c, a)
enddo
```

Program function:

set a to abs(b) and set d to max of c and abs(b), or
a, d := abs(b), max(c, abs(b))

- **Sequence semantics (Function Theorem):**

f = do g; h enddo

$$\boxed{(x, y) \in f \rightarrow y = h(g(x))}$$

(x: inputs, y: outputs)

Alternation Program Function

- **Ifthenelse Example:**

```
if a > b
then
    c := a
else
    c := b
endif
```

Program function:

set b to max of a and b, or

$(a > b \rightarrow c := a \mid a <= b \rightarrow c := b)$

Ifthenelse semantics (Function Theorem):

f = if p then g else h endif

$$(x, y) \in f \quad \rightarrow \quad (p(x) \wedge y = g(x) \quad | \quad \sim p(x) \wedge y = h(x))$$

Iteration Program Function

- **Whiledo example:**

```
while
   b > 1
do
   b := b - 2
enddo
```

Extracted program function:

set odd b to 1 or set even b to 0, or
(b odd → x := 1 | b even → x := 0)

- **Whiledo semantics (Function Theorem):**

f = while p do g enddo

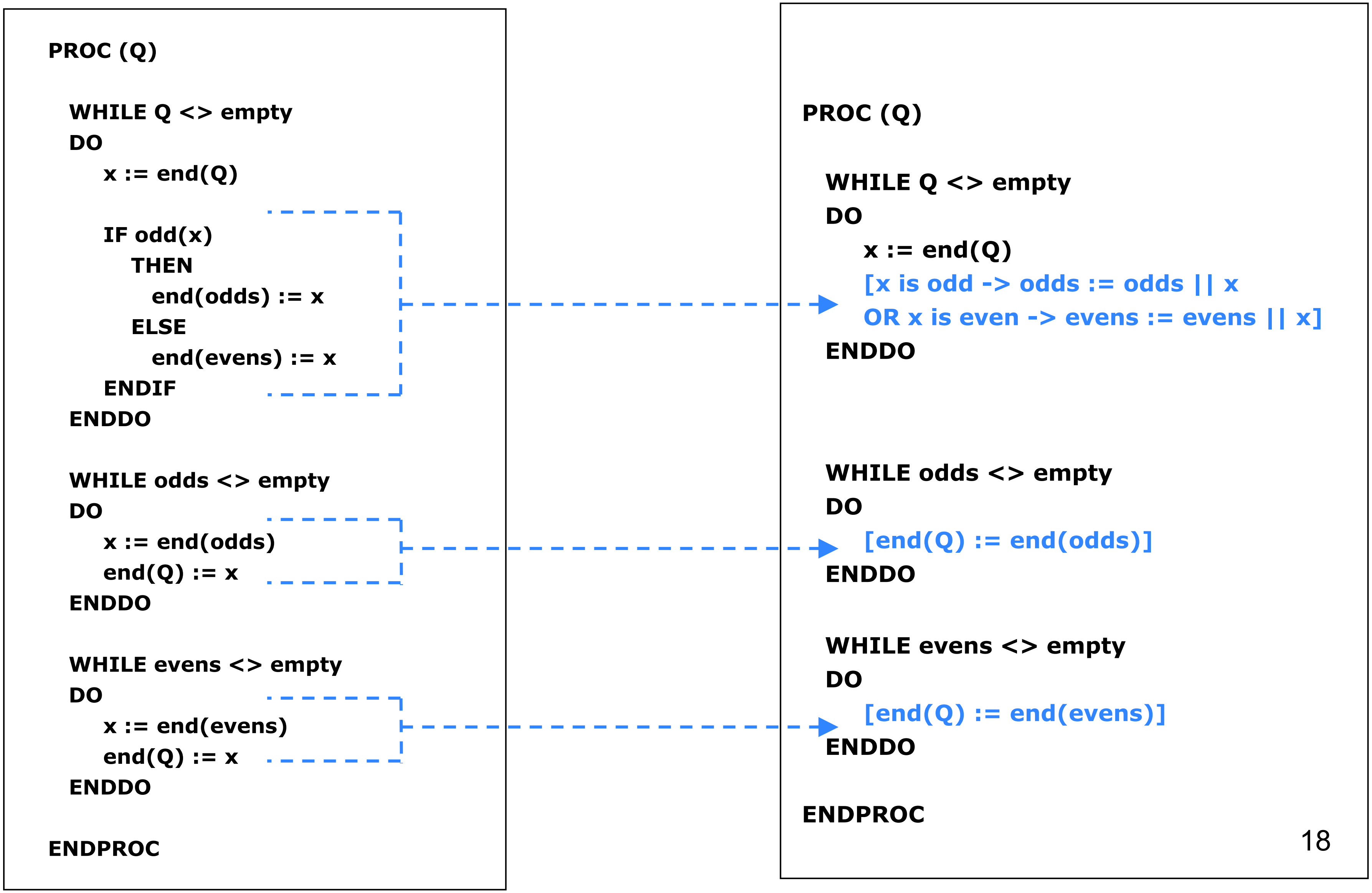
$$(x, y) \in f \rightarrow \text{termination} \wedge (p(x) \wedge y = f(g(x))) \;\; | \;\; (\sim p(x) \wedge y = x))$$

A Function Extractor at Work

```
PROC (Q)
  odds, evens: queue of integer,
                 initial empty
  x: integer
  WHILE Q <> empty
  DO
     x := end(Q)

     IF odd(x)
        THEN
          end(odds) := x
        ELSE
          end(evens) := x
     ENDIF
  ENDDO

  WHILE odds <> empty
  DO
     x := end(odds)
     end(Q) := x
  ENDDO

  WHILE evens <> empty
  DO
     x := end(evens)
     end(Q) := x
  ENDDO
ENDPROC
```

- Move from fallible process in human time scale to precise process in CPU time scale
- Based on algebra of functions
- Extract behavior of each control structure in turn
- Express final values of data in terms of initial values
- Behavior given in procedure-free conditional concurrent assignments
- Extracted behavior reveals any malicious properties

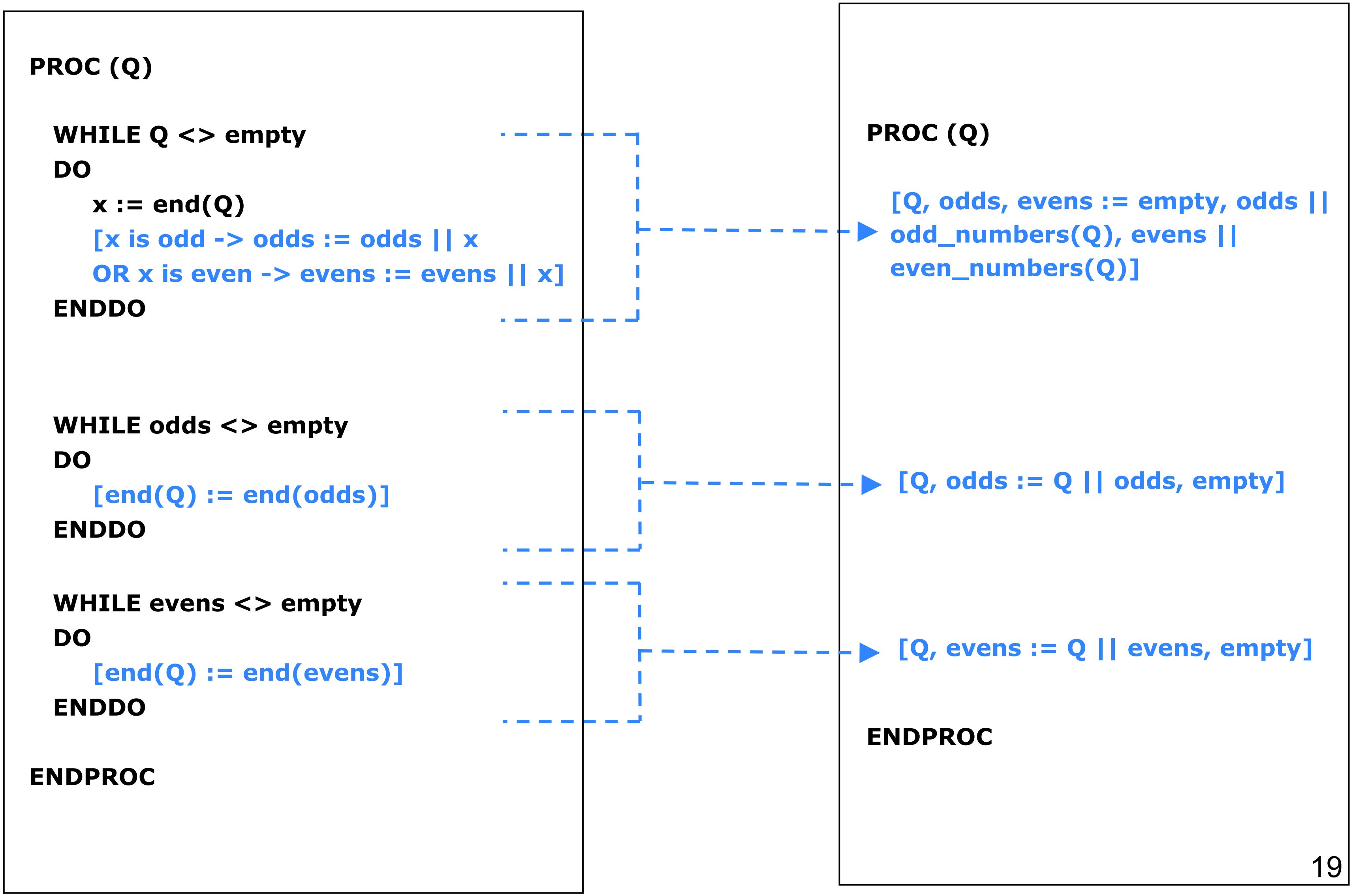
Extracting Behavior of Low-Level Structures



```
PROC (Q)

  WHILE Q <> empty
  DO
     x := end(Q)

     IF odd(x)
        THEN
          end(odds) := x
        ELSE
          end(evens) := x
     ENDIF
  ENDDO

  WHILE odds <> empty
  DO
     x := end(odds)
     end(Q) := x
  ENDDO

  WHILE evens <> empty
  DO
     x := end(evens)
     end(Q) := x
  ENDDO

ENDPROC
```

```
PROC (Q)

  WHILE Q <> empty
  DO
     x := end(Q)
     [x is odd -> odds := odds || x
     OR x is even -> evens := evens || x]
  ENDDO


  WHILE odds <> empty
  DO
     [end(Q) := end(odds)]
  ENDDO

  WHILE evens <> empty
  DO
     [end(Q) := end(evens)]
  ENDDO

ENDPROC
```

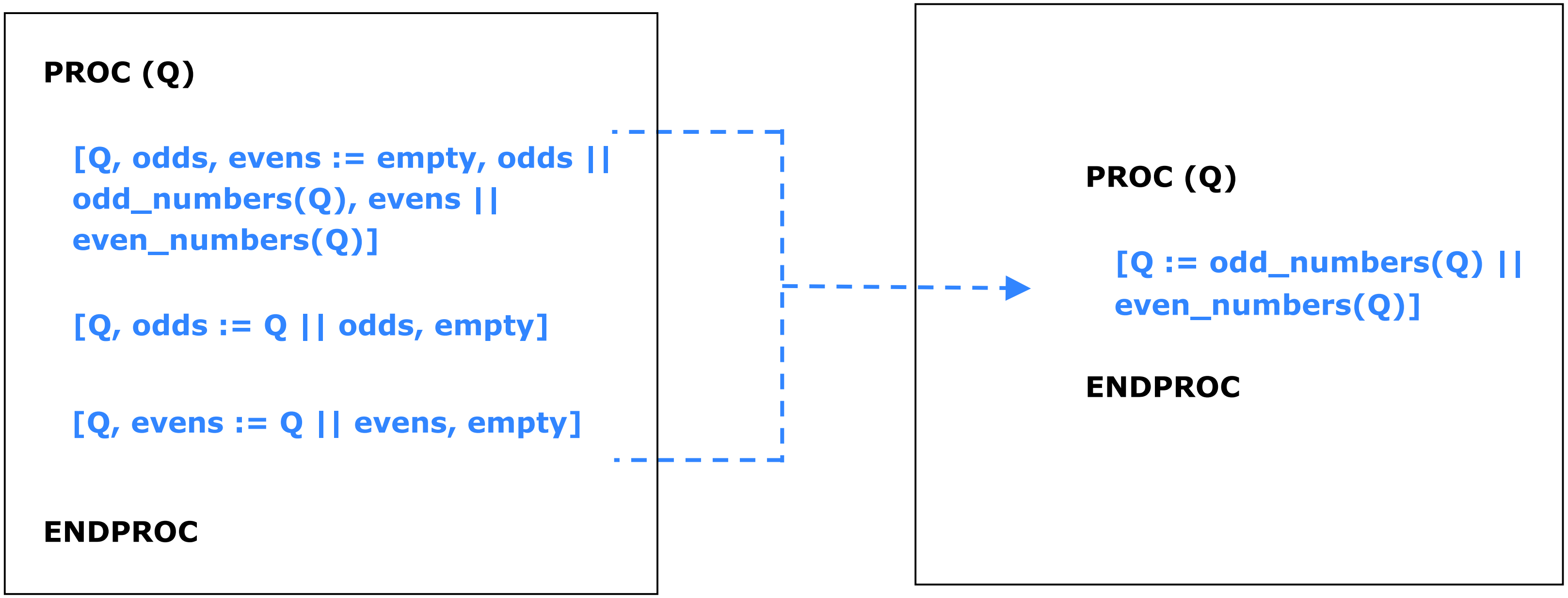
Extracting Behavior of Next-Level Structures



```
PROC (Q)

   WHILE Q <> empty
   DO
        x := end(Q)
        [x is odd -> odds := odds || x
        OR x is even -> evens := evens || x]
   ENDDO


   WHILE odds <> empty
   DO
        [end(Q) := end(odds)]
   ENDDO

   WHILE evens <> empty
   DO
        [end(Q) := end(evens)]
   ENDDO

ENDPROC
```

```
PROC (Q)

   [Q, odds, evens := empty, odds ||
   odd_numbers(Q), evens ||
   even_numbers(Q)]


   [Q, odds := Q || odds, empty]


   [Q, evens := Q || evens, empty]


ENDPROC
```

Extracting Behavior of Highest-Level Structure



```
PROC (Q)

  [Q, odds, evens := empty, odds ||
  odd_numbers(Q), evens ||
  even_numbers(Q)]

  [Q, odds := Q || odds, empty]

  [Q, evens := Q || evens, empty]

ENDPROC
```

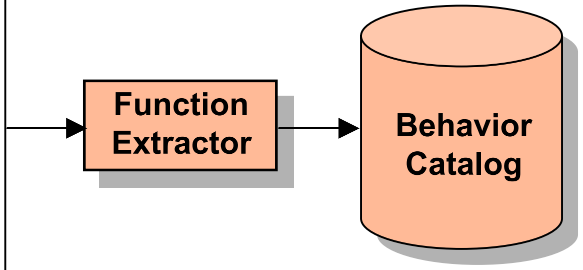
```
PROC (Q)

  [Q := odd_numbers(Q) ||
  even_numbers(Q)]

ENDPROC
```

- Extracted behavior is the precise as-built specification
- Number of control structures is finite for stepwise abstraction
- Intermediate structures and data drop out to simplify scale-up
- Behavior is recorded at all levels of abstraction

Examples

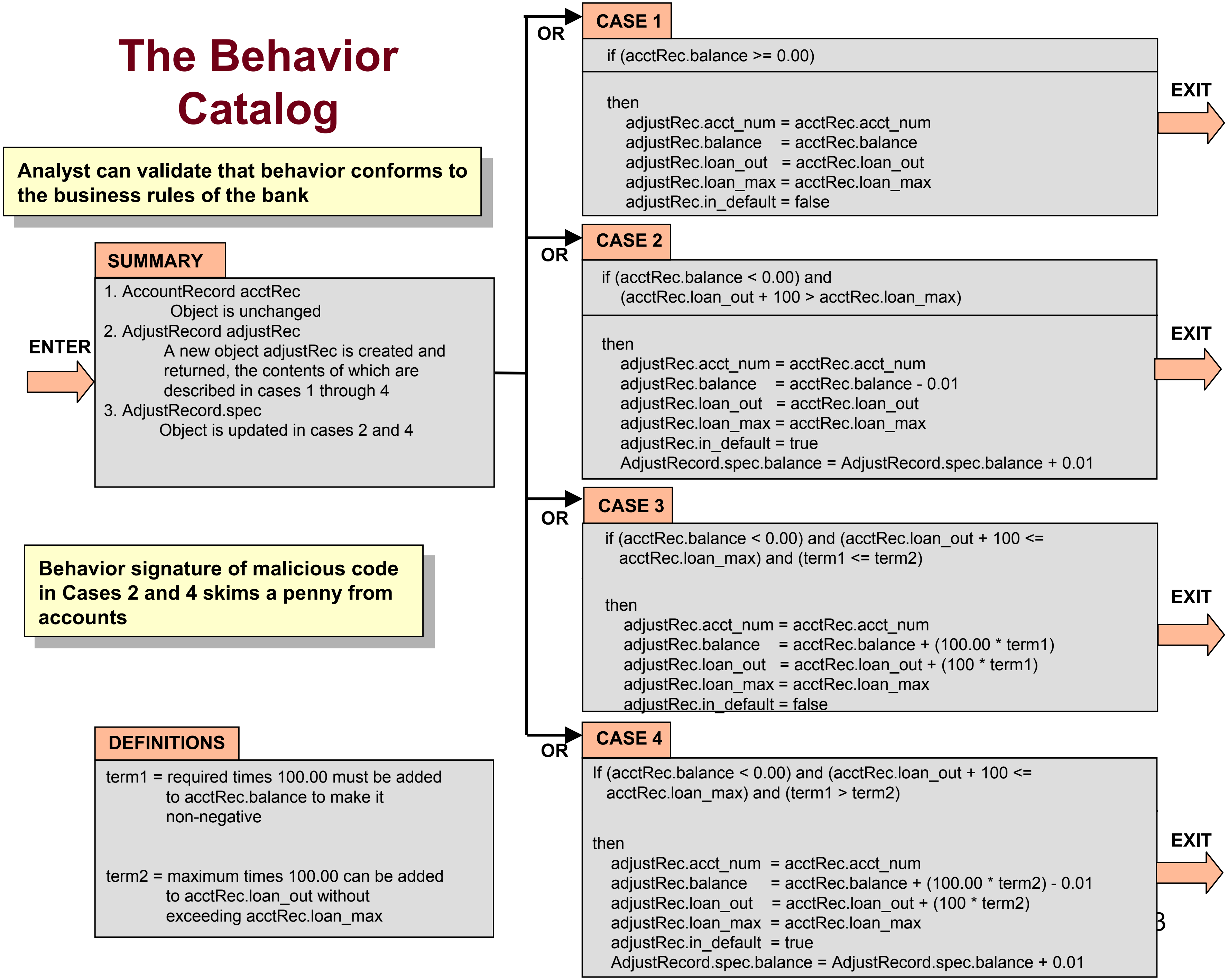
Does This Program Contain Malicious Code?

```
public class AccountRecord {
  public int acct_num;
  public double balance;
  public int loan_out;
  public int loan_max;
} // end of AccountRecord

public class AdjustRecord
extends AccountRecord {
  public boolean in_default;
  public static AdjustRecord spec;
} // end of AdjustRecord

public static AdjustRecord classify_account
(AccountRecord acctRec) {
  AdjustRecord adjustRec = new AdjustRecord();
  adjustRec.acct_num = acctRec.acct_num;
  adjustRec.balance = acctRec.balance;
  adjustRec.loan_out = acctRec.loan_out;
  adjustRec.loan_max = acctRec.loan_max;
  while ((adjustRec.balance < 0.00) &&
      ((adjustRec.loan_out + 100) <= adjustRec.loan_max)) {
    adjustRec.loan_out += 100;
    adjustRec.balance += 100.00;
  }
  adjustRec.in_default = (adjustRec.balance < 0.00);
  if (adjustRec.balance < 0.00) {
    adjustRec.balance -= 0.01;
    AdjustRecord.spec.balance += 0.01;
  }
  return adjustRec;
}
```



The Behavior Catalog

Analyst can validate that behavior conforms to the business rules of the bank



SUMMARY

1. AccountRecord acctRec
   Object is unchanged
2. AdjustRecord adjustRec
   A new object adjustRec is created and returned, the contents of which are described in cases 1 through 4
3. AdjustRecord.spec
   Object is updated in cases 2 and 4

Behavior signature of malicious code in Cases 2 and 4 skims a penny from accounts

DEFINITIONS

term1 = required times 100.00 must be added to acctRec.balance to make it non-negative

term2 = maximum times 100.00 can be added to acctRec.loan_out without exceeding acctRec.loan_max

CASE 1

```
if (acctRec.balance >= 0.00)

then
   adjustRec.acct_num = acctRec.acct_num
   adjustRec.balance   = acctRec.balance
   adjustRec.loan_out   = acctRec.loan_out
   adjustRec.loan_max = acctRec.loan_max
   adjustRec.in_default = false
```

CASE 2

```
if (acctRec.balance < 0.00) and
   (acctRec.loan_out + 100 > acctRec.loan_max)

then
   adjustRec.acct_num = acctRec.acct_num
   adjustRec.balance    = acctRec.balance - 0.01
   adjustRec.loan_out   = acctRec.loan_out
   adjustRec.loan_max = acctRec.loan_max
   adjustRec.in_default = true
   AdjustRecord.spec.balance = AdjustRecord.spec.balance + 0.01
```

CASE 3

```
if (acctRec.balance < 0.00) and (acctRec.loan_out + 100 <=
   acctRec.loan_max) and (term1 <= term2)

then
   adjustRec.acct_num = acctRec.acct_num
   adjustRec.balance    = acctRec.balance + (100.00 * term1)
   adjustRec.loan_out   = acctRec.loan_out + (100 * term1)
   adjustRec.loan_max = acctRec.loan_max
   adjustRec.in_default = false
```

CASE 4

```
If (acctRec.balance < 0.00) and (acctRec.loan_out + 100 <=
   acctRec.loan_max) and (term1 > term2)

then
   adjustRec.acct_num  = acctRec.acct_num
   adjustRec.balance     = acctRec.balance + (100.00 * term2) - 0.01
   adjustRec.loan_out    = acctRec.loan_out + (100 * term2)
   adjustRec.loan_max  = acctRec.loan_max
   adjustRec.in_default  = true
   AdjustRecord.spec.balance = AdjustRecord.spec.balance + 0.01
```

Other Uses of Function Extractors

- Development
  Software engineer submits a program being written to an extractor to determine if its behavior is the function intended.

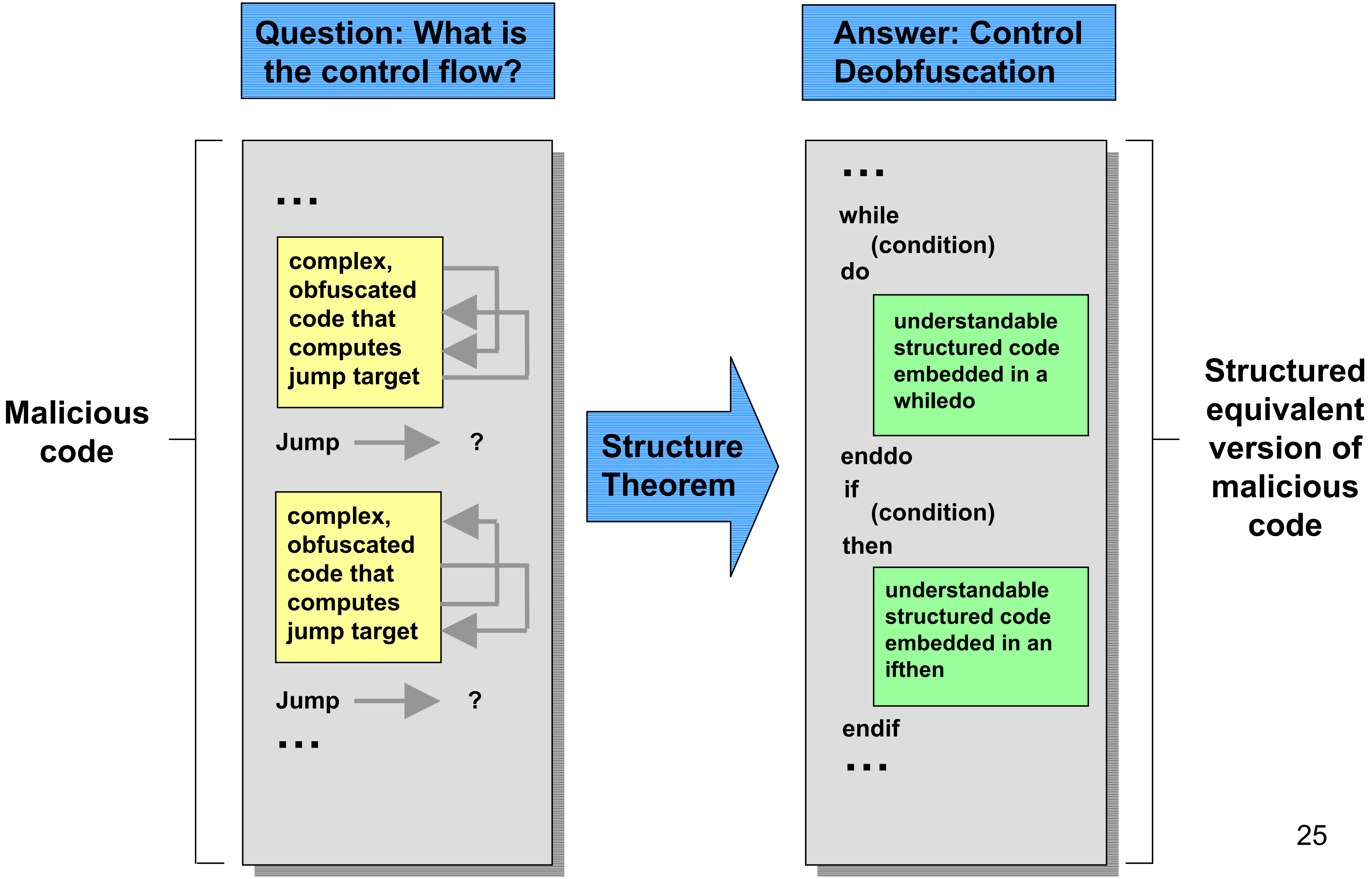
- Errors
  Software engineer submits a program to an extractor to see if it exhibits behavior that is correct with respect to requirements.

- Vulnerabilities
  Software engineer submits a program to an extractor to determine if it exhibits behavior that can be exploited by an intruder.

- Legacy and vendor code
  Systems engineer submits a program to an extractor to generate its behavior catalog for use in new system integration.

- Composition
  Systems engineer submits a composition of components to an extractor to determine if their combined behavior is correct.

Control Deobfuscation Example



Question: What is the control flow?

Answer: Control Deobfuscation

Malicious code

. . .

complex, obfuscated code that computes jump target

Jump ?

complex, obfuscated code that computes jump target

Jump ?

. . .

Structure Theorem

```
. . .
while
    (condition)
do
    understandable structured code embedded in a whiledo
enddo
if
    (condition)
then
    understandable structured code embedded in an ifthen
endif
. . .
```

Structured equivalent version of malicious code

Control Flow Deobfuscation

- Dead-point analysis
- Star-point analysis
- Instruction misdirection analysis

Function Extraction Example

- **Program:**

```
do
    x := x + y
    y := x - y
    x : =x - y
 enddo
```

Question: What does this program do?

Function Theorem

Answer: Function Extraction

- **Function extraction:**

assignment	x	y
1 x := x + y	x1 = x0 + y0	y1 = y0
2 y := x - y	x2 = x1	y2 = x1 - y1
3 x := x - y	x3 = x2 - y2	y3 = y2

derivations:

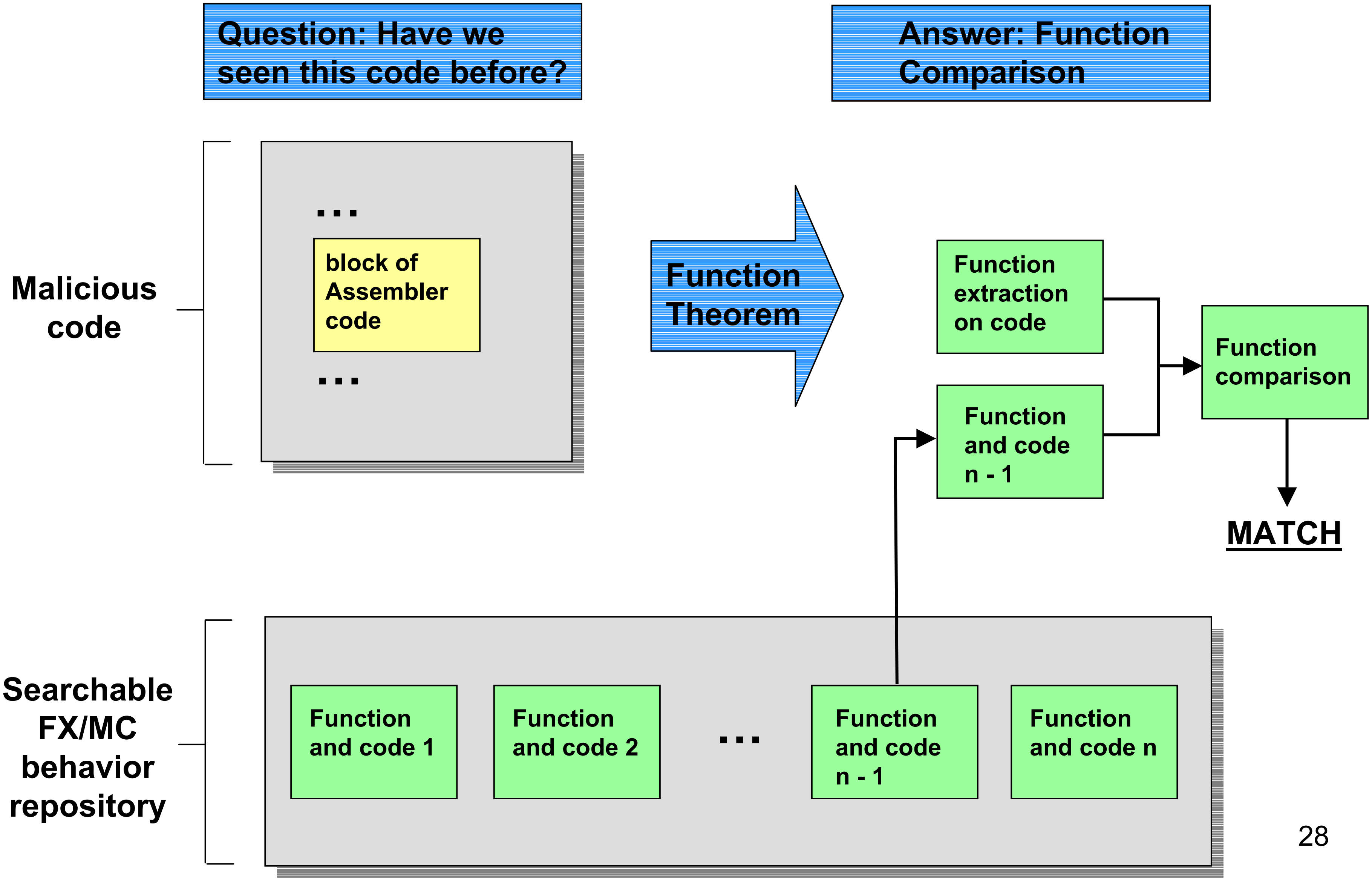
$$\begin{aligned} x3 &= x2 - y2 \\ &= x1 - (x1 - y1) \\ &= y1 \\ &= y0 \end{aligned}$$

$$\begin{aligned} y3 &= y2 \\ &= x1 - y1 \\ &= x0 + y0 - y0 \\ &= x0 \end{aligned}$$

Program behavior:

x, y := y, x

Function Comparison Example



Conclusion

FX Life Cycle Impacts – Where to Next?

	Life Cycle Activity	Specification Automation	Architecture Automation	Assembler Automation	C Automation	C++ Automation	Java automation	Other Lang. Automation
1	**Specification Development**	Specification Behavior Extractor Behavior Catalog Analyzer						
2	**Architecture Development**		Architecture Behavior Extractor Behavior Catalog Analyzer					
3	**Component Development (Design & Implementation) and Evaluation (of vendor software)**			**Structure Transformer Function Extractor Behavior Catalog Analyzer**	**Structure Transformer Function Extractor Behavior Catalog Analyzer**	**Structure Transformer Function Extractor Behavior Catalog Analyzer**	**Structure Transformer Function Extractor Behavior Catalog Analyzer**	**Structure Transformer Function Extractor Behavior Catalog Analyzer**
4	**Correctness Verification**			Correctness Verifier	Correctness Verifier	Correctness Verifier	Correctness Verifier	Correctness Verifier
5	**System Testing**			Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer
6	**System Integration**			Component Composition Generator Behavior Catalog Analyzer	Component Composition Generator Behavior Catalog Analyzer	Component Composition Generator Behavior Catalog Analyzer	Component Composition Generator Behavior Catalog Analyzer	Component Composition Generator Behavior Catalog Analyzer
7	**System Maintenance**			Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer
8	**Component Reuse**			Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer	Behavior Catalog Analyzer

Status

- Hard problem!
- FX/MC project is underway, additional participation is welcome
- SEI is conducting a study to determine the best course of evolution for FX technology
- Future FX targets can include C, C++, Java, or other languages
- FX extensions can include correctness verification and composition of components, as well as function extraction from specifications, architectures, and designs
- Contact Rick Linger (rlinger@sei.cmu.edu)