Analysis of Software Artifacts

Introduction to Static Analysis

Jonathan Aldrich

Find the Bug!

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);
        return bh;
}
```

disable interrupts

re-enable interrupts

Find the Bug!

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);
        return bh;
}
```

disable interrupts

ERROR: returning with interrupts disabled

re-enable interrupts

Metal Interrupt Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // Patterns
  // to specify enable/disable functions.
  pat enable = { sti(); }
             | { restore_flags(flags); } ;
  pat disable = { cli(); };

  // States
  // The first state is the initial state.
  is_enabled: disable ==> is_disabled
     | enable ==> { err("double enable"); }
     ;
  is_disabled: enable ==> is_enabled
     | disable ==> { err("double disable"); }
     // Special pattern that matches when the SM
     // hits the end of any path in this state.
     | $end_of_path$ ==>
        { err("exiting w/intr disabled!"); }
     ;
}
```

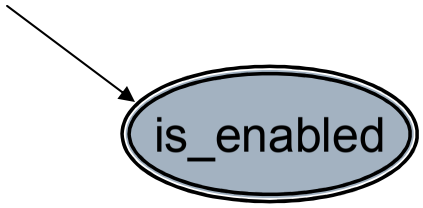
Metal Interrupt Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // Patterns
  // to specify enable/disable functions.
  pat enable = { sti(); }
             | { restore_flags(flags); } ;
  pat disable = { cli(); };

  // States
  // The first state is the initial state.
  is_enabled: disable ==> is_disabled
     | enable ==> { err("double enable"); }
     ;
  is_disabled: enable ==> is_enabled
     | disable ==> { err("double disable"); }
     // Special pattern that matches when the SM
     // hits the end of any path in this state.
     | $end_of_path$ ==>
        { err("exiting w/intr disabled!"); }
     ;
}
```



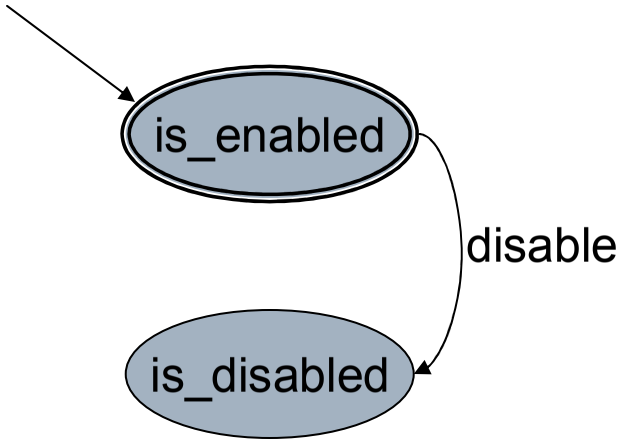
Metal Interrupt Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // Patterns
  // to specify enable/disable functions.
  pat enable = { sti(); }
             | { restore_flags(flags); } ;
  pat disable = { cli(); };

  // States
  // The first state is the initial state.
  is_enabled: disable ==> is_disabled
     | enable ==> { err("double enable"); }
     ;
  is_disabled: enable ==> is_enabled
     | disable ==> { err("double disable"); }
     // Special pattern that matches when the SM
     // hits the end of any path in this state.
     | $end_of_path$ ==>
        { err("exiting w/intr disabled!"); }
     ;
}
```



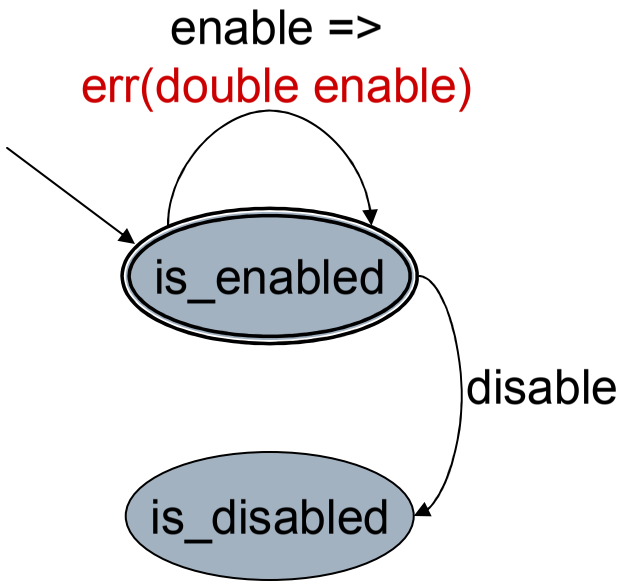
Metal Interrupt Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // Patterns
  // to specify enable/disable functions.
  pat enable = { sti(); }
             | { restore_flags(flags); } ;
  pat disable = { cli(); };

  // States
  // The first state is the initial state.
  is_enabled: disable ==> is_disabled
     | enable ==> { err("double enable"); }
     ;
  is_disabled: enable ==> is_enabled
     | disable ==> { err("double disable"); }
     // Special pattern that matches when the SM
     // hits the end of any path in this state.
     | $end_of_path$ ==>
        { err("exiting w/intr disabled!"); }
     ;
}
```



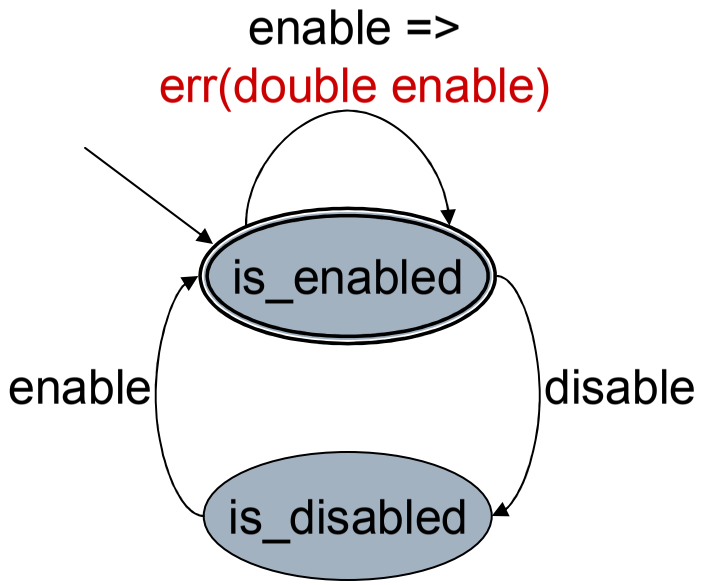
Metal Interrupt Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // Patterns
  // to specify enable/disable functions.
  pat enable = { sti(); }
             | { restore_flags(flags); } ;
  pat disable = { cli(); };

  // States
  // The first state is the initial state.
  is_enabled: disable ==> is_disabled
     | enable ==> { err("double enable"); }
     ;
  is_disabled: enable ==> is_enabled
     | disable ==> { err("double disable"); }
     // Special pattern that matches when the SM
     // hits the end of any path in this state.
     | $end_of_path$ ==>
        { err("exiting w/intr disabled!"); }
     ;
}
```



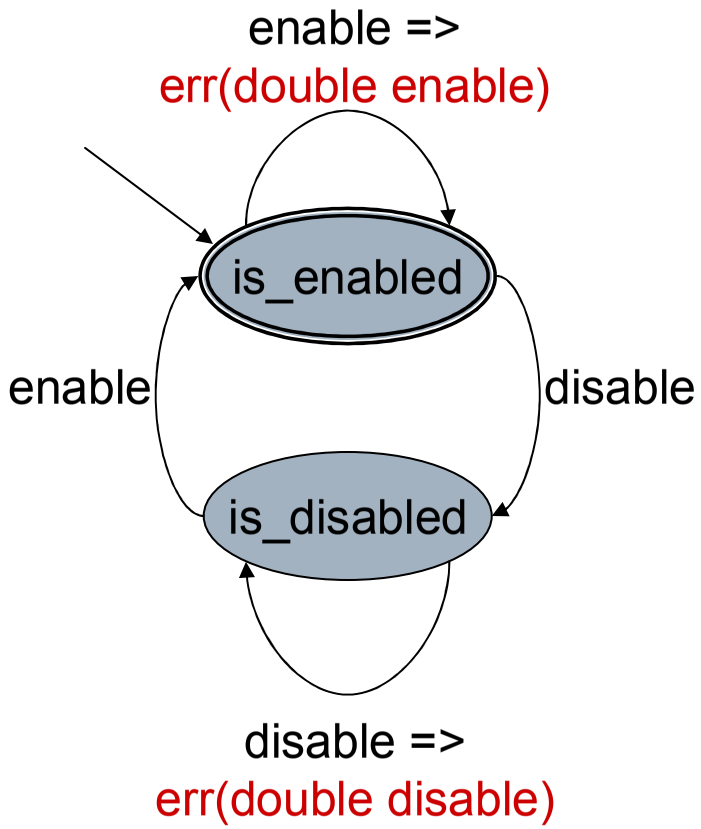
Metal Interrupt Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // Patterns
  // to specify enable/disable functions.
  pat enable = { sti(); }
             | { restore_flags(flags); } ;
  pat disable = { cli(); };

  // States
  // The first state is the initial state.
  is_enabled: disable ==> is_disabled
     | enable ==> { err("double enable"); }
     ;
  is_disabled: enable ==> is_enabled
     | disable ==> { err("double disable"); }
     // Special pattern that matches when the SM
     // hits the end of any path in this state.
     | $end_of_path$ ==>
        { err("exiting w/intr disabled!"); }
     ;
}
```



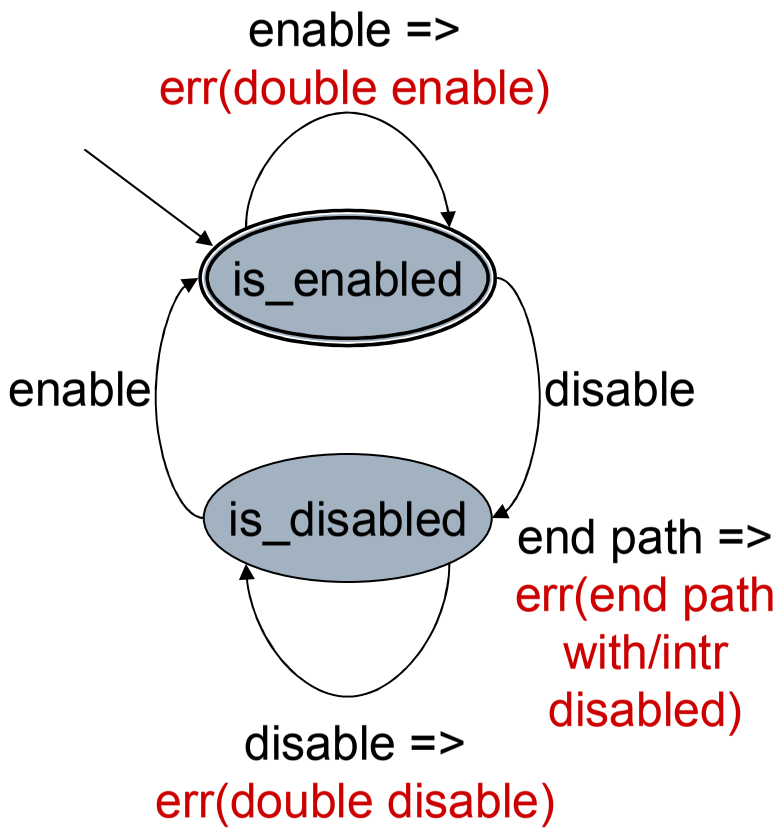
Metal Interrupt Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // Patterns
  // to specify enable/disable functions.
  pat enable = { sti(); }
             | { restore_flags(flags); } ;
  pat disable = { cli(); };

  // States
  // The first state is the initial state.
  is_enabled: disable ==> is_disabled
     | enable ==> { err("double enable"); }
     ;
  is_disabled: enable ==> is_enabled
     | disable ==> { err("double disable"); }
     // Special pattern that matches when the SM
     // hits the end of any path in this state.
     | $end_of_path$ ==>
        { err("exiting w/intr disabled!"); }
     ;
}
```



Applying the Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);
        return bh;
}
```

Applying the Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh, ⟵ initial state is_enabled
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);
        return bh;
}
```

Applying the Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,        <-- initial state is_enabled
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();                                 <-- transition to is_disabled
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);
        return bh;
}
```

Applying the Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);
        return bh;
}
```

initial state is_enabled

transition to is_disabled

final state is_disabled: ERROR!

Applying the Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,        <-- initial state is_enabled
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();                                 <-- transition to is_disabled
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;                   <-- final state is_disabled: ERROR!
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);                  <-- transition to is_enabled
        return bh;
}
```

Applying the Analysis

Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '00.

```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,        <-- initial state is_enabled
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        save_flags(flags);
        cli();                                 <-- transition to is_disabled
        if ((bh = sh->buffer_pool) == NULL)
                return NULL;                   <-- final state is_disabled: ERROR!
        sh->buffer_pool = bh->b_next;
        bh->b_size = b_size;
        restore_flags(flags);                  <-- transition to is_enabled
        return bh;                             <-- final state is_enabled is OK
}
```

Outline

- Why static analysis?
  - The limits of testing and inspection
- What is static analysis?
- Representing programs
- AST-walking analyses
- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm

A problem has been detected and Windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Static Analysis Finds "Mechanical" Errors

- Defects that result from inconsistently following simple, mechanical design rules

- Security vulnerabilities
  - Buffer overruns, unvalidated input…
- Memory errors
  - Null dereference, uninitialized data…
- Resource leaks
  - Memory, OS resources…
- Violations of API or framework rules
  - e.g. Windows device drivers; real time libraries; GUI frameworks
- Exceptions
  - Arithmetic/library/user-defined
- Encapsulation violations
  - Accessing internal data, calling private functions…
- Race conditions
  - Two threads access the same data without synchronization

Difficult to Find with Testing, Inspection

- Non-local, uncommon paths
  - Security vulnerabilities
  - Memory errors
  - Resource leaks
  - Violations of API or framework rules
  - Exceptions
  - Encapsulation violations
- Non-deterministic
  - Race conditions

Quality Assurance at Microsoft (Part 1)

- Original process: manual code inspection
  - Effective when system and team are small
  - Too many paths to consider as system grew

Quality Assurance at Microsoft (Part 1)

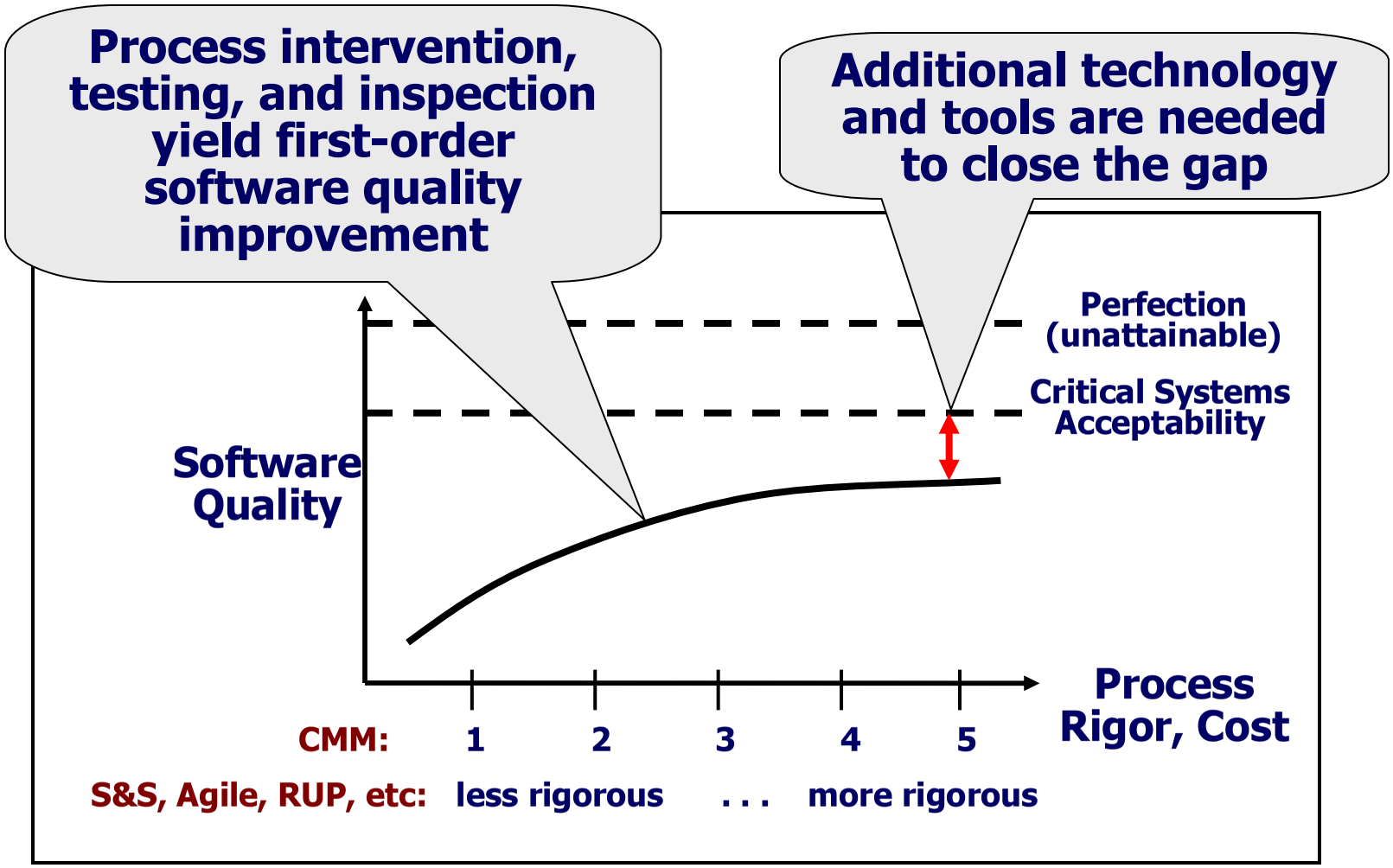
- Original process: manual code inspection
  - Effective when system and team are small
  - Too many paths to consider as system grew
- Early 1990s: add massive system and unit testing
  - Tests took weeks to run
    - Diversity of platforms and configurations
    - Sheer volume of tests
  - Inefficient detection of common patterns, security holes
    - Non-local, intermittent, uncommon path bugs
  - Was treading water in Windows Vista development

Quality Assurance at Microsoft (Part 1)

- Original process: manual code inspection
  - Effective when system and team are small
  - Too many paths to consider as system grew
- Early 1990s: add massive system and unit testing
  - Tests took weeks to run
    - Diversity of platforms and configurations
    - Sheer volume of tests
  - Inefficient detection of common patterns, security holes
    - Non-local, intermittent, uncommon path bugs
  - Was treading water in Windows Vista development
- Early 2000s: add static analysis
  - More on this later

Process, Cost, and Quality

Slide: William Scherlis



Outline

- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Representing programs
- AST-walking analyses
- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm

Static Analysis Definition

- Static program analysis is the systematic examination of an abstraction of a program's state space

Static Analysis Definition

- Static program analysis is the systematic examination of an abstraction of a program's state space

- Metal interrupt analysis
  - Abstraction
    - 2 states: enabled and disabled
      - All program information—variable values, heap contents—is abstracted by these two states, plus the program counter

Static Analysis Definition

- Static program analysis is the systematic examination of an abstraction of a program's state space

- Metal interrupt analysis
  - Abstraction
    - 2 states: enabled and disabled
      - All program information—variable values, heap contents—is abstracted by these two states, plus the program counter
  - Systematic
    - Examines all paths through a function
      - What about loops? More later…
    - Each path explored for each reachable state
      - Assume interrupts initially enabled (Linux practice)
      - Since the two states abstract all program information, the exploration is exhaustive

Outline

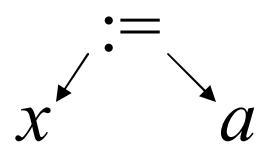
- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Representing programs
- AST-walking analyses
- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm

Representing Programs

- To analyze software automatically, we must be able to represent it precisely
- Some representations
  - Source code
  - Abstract syntax trees
  - Control flow graph
  - Bytecode
  - Assembly code
  - Binary code

Abstract Syntax Trees

- A tree representation of source code
- Based on the language grammar
  - One type of node for each production
  - $S ::= x := a$ ➔



  - $S ::= \text{while } b \text{ do } S$ ➔

Parsing: Source to AST

- Parsing process (top down)
  1. Determine the top-level production to use
  2. Create an AST element for that production
  3. Determine what text corresponds to each child of the AST element
  4. Recursively parse each child
- Algorithms have been studied in detail
  - For this course you only need the intuition
  - Details covered in compiler courses

Parsing Example

```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

- Top-level production?
- What are the parts?

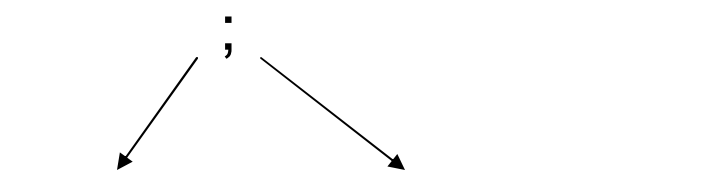
Parsing Example

```
y := x;                                   ;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

- Top-level production?
  - $S_1; S_2$
- What are the parts?

Parsing Example

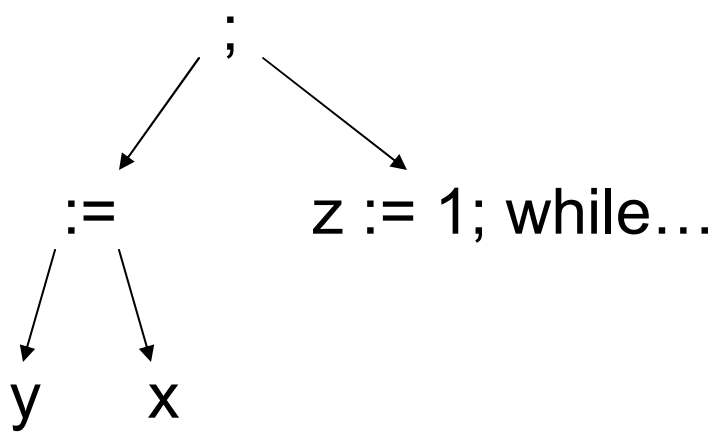
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

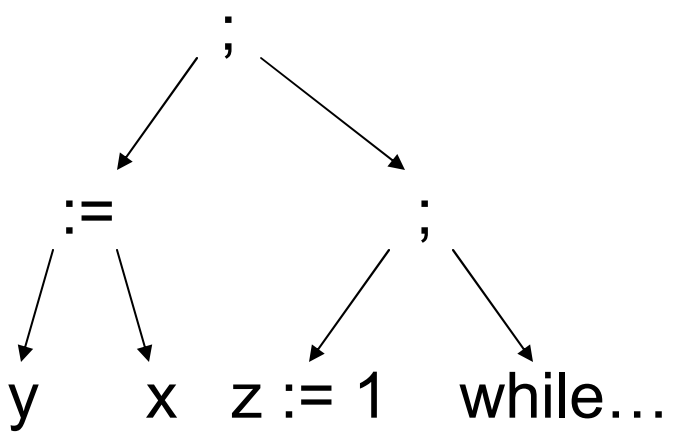
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

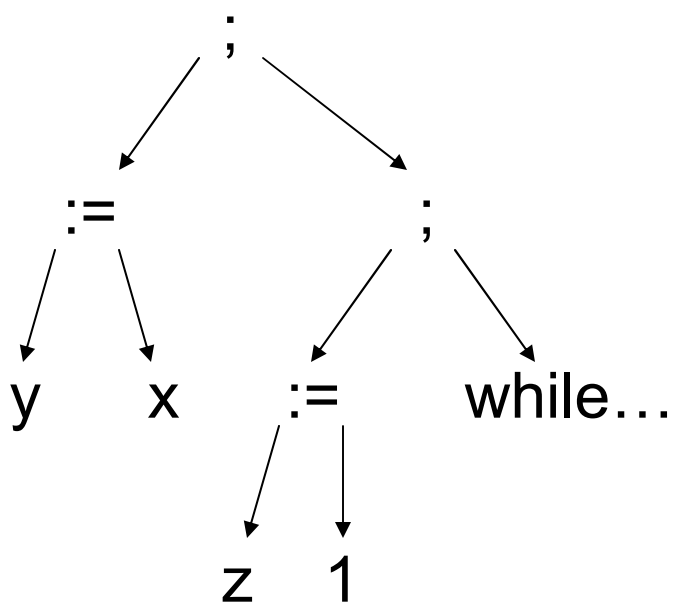
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

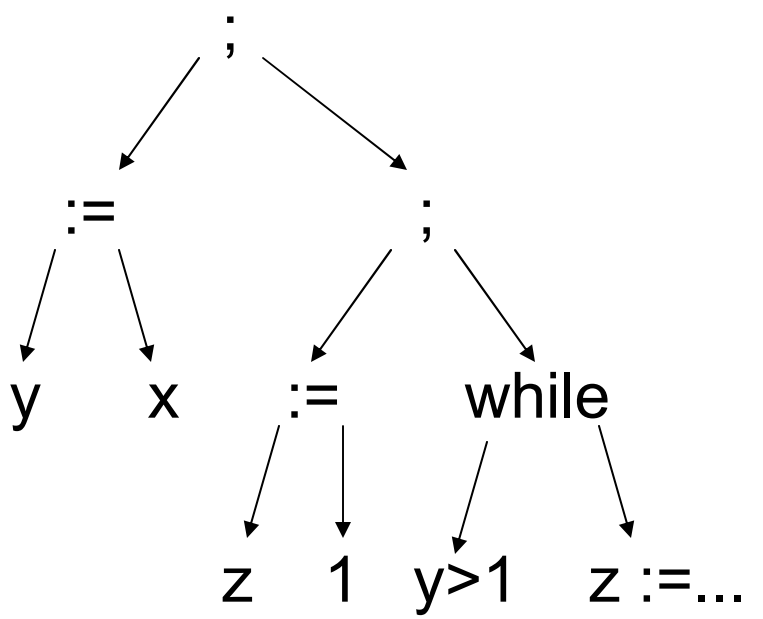
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

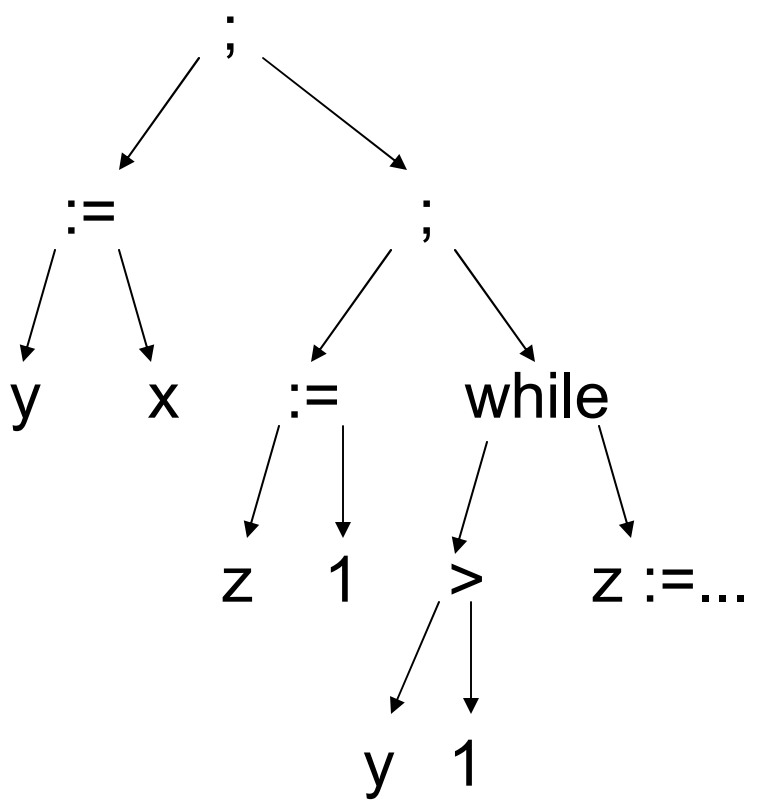
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

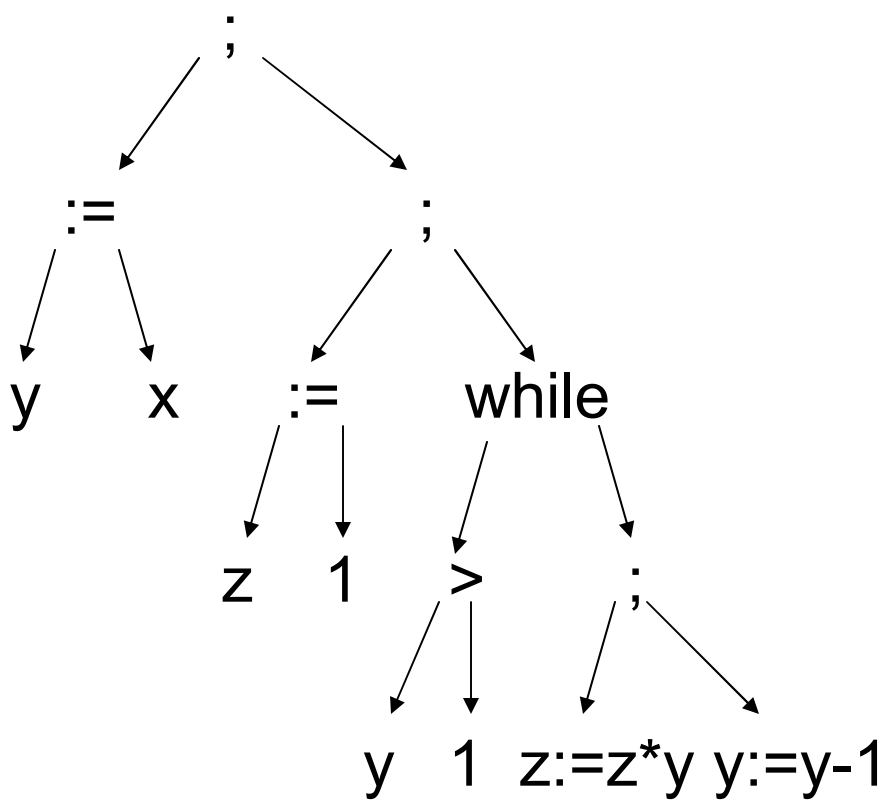
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

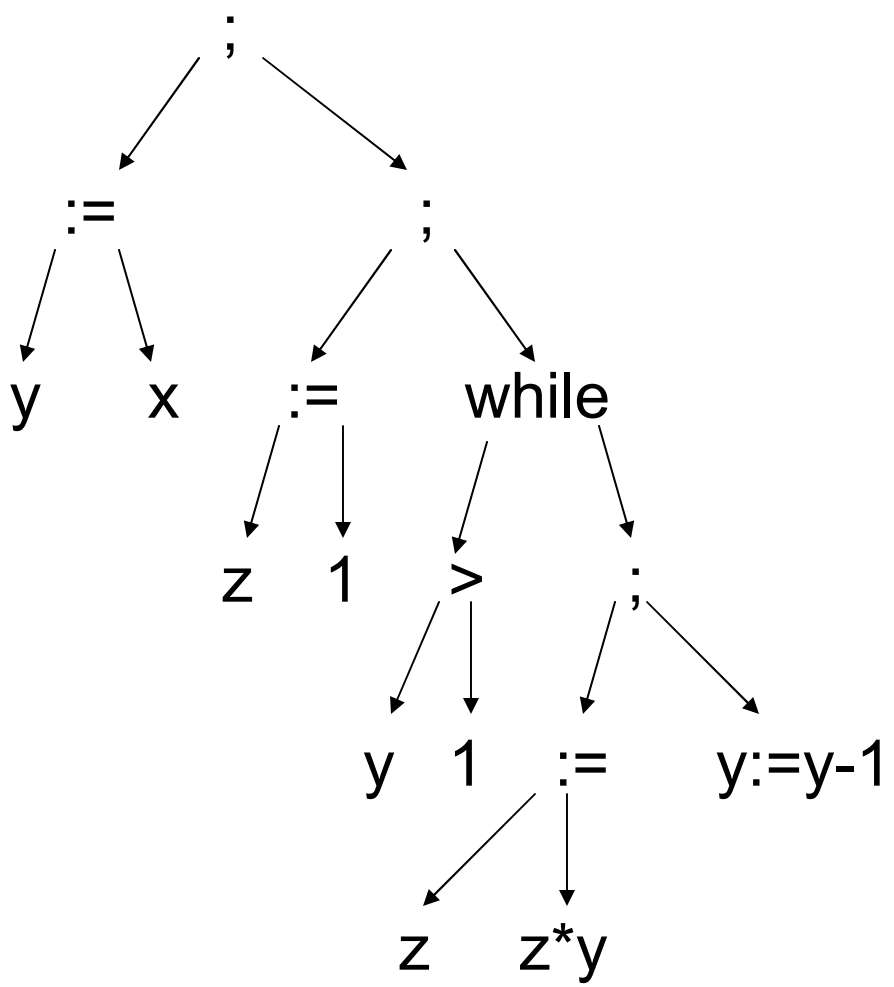
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

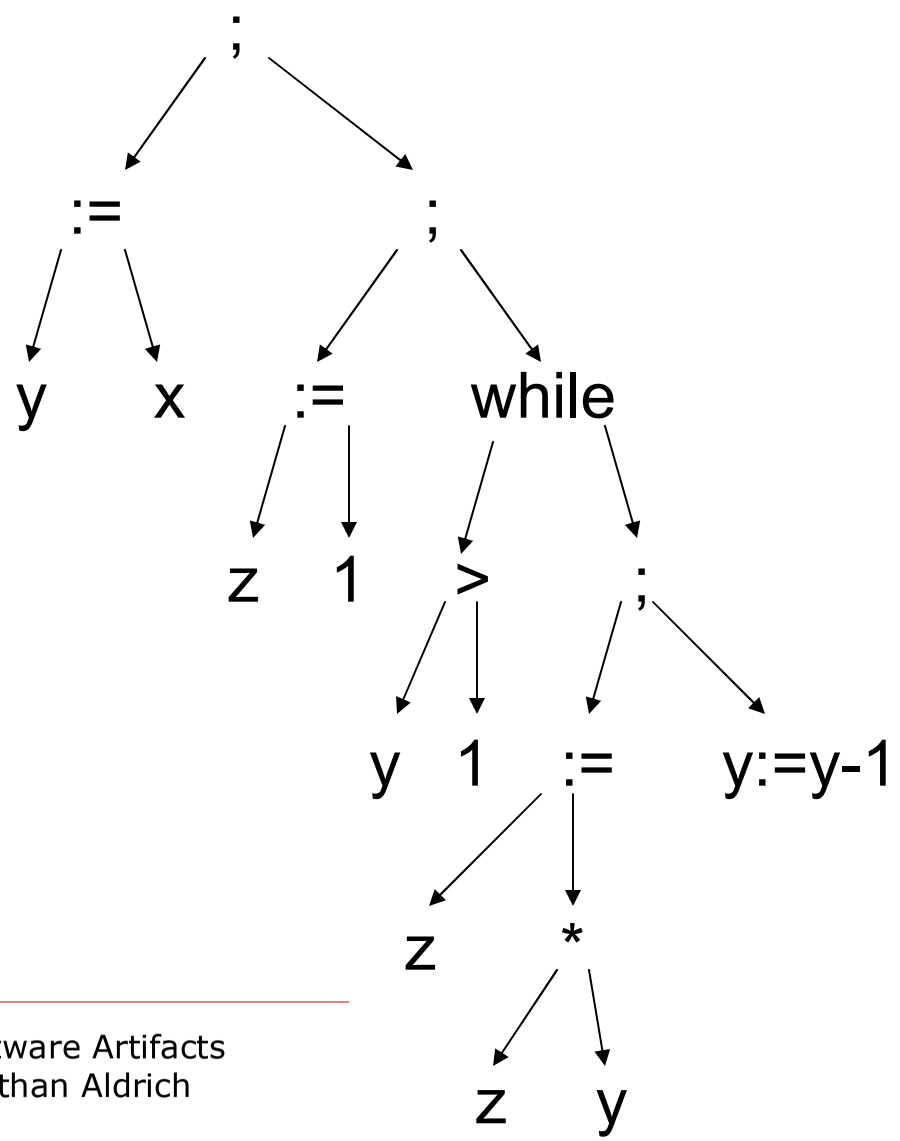


- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Parsing Example

```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

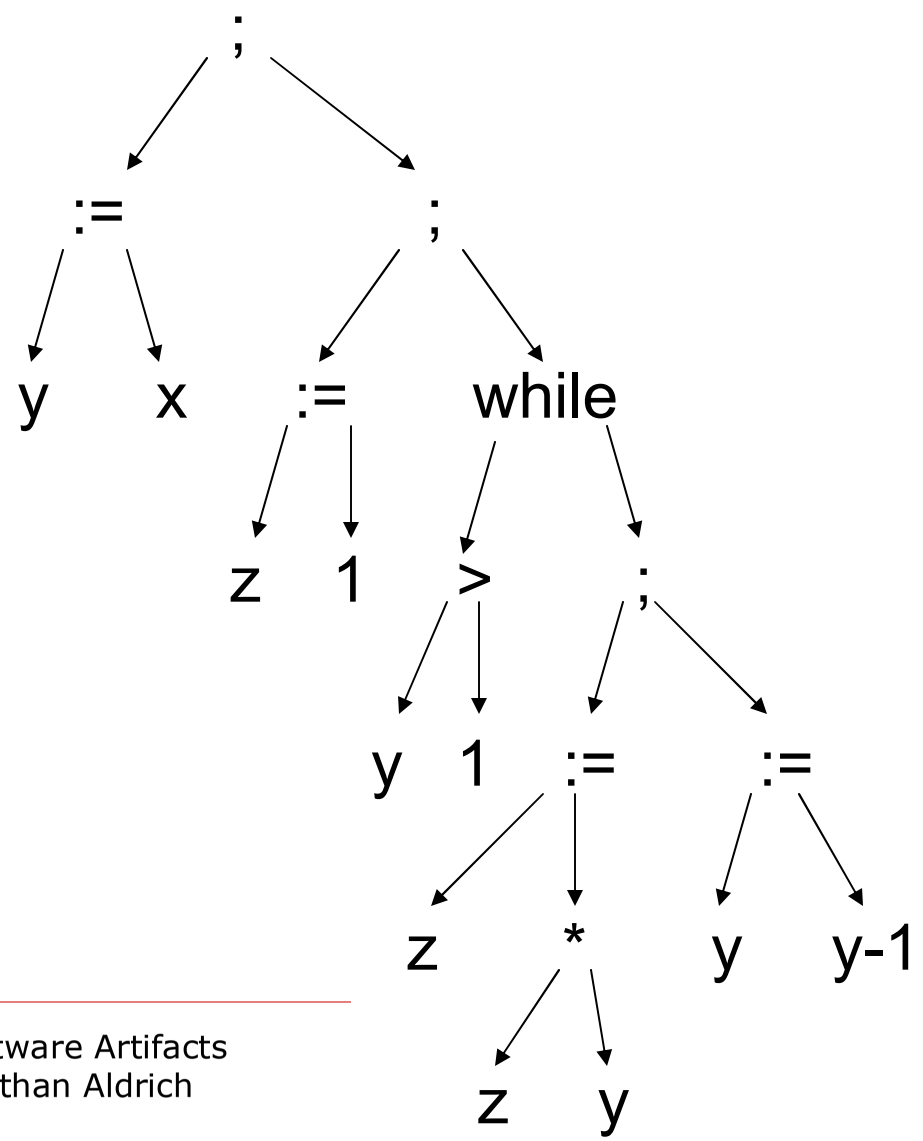
- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …



Parsing Example

```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

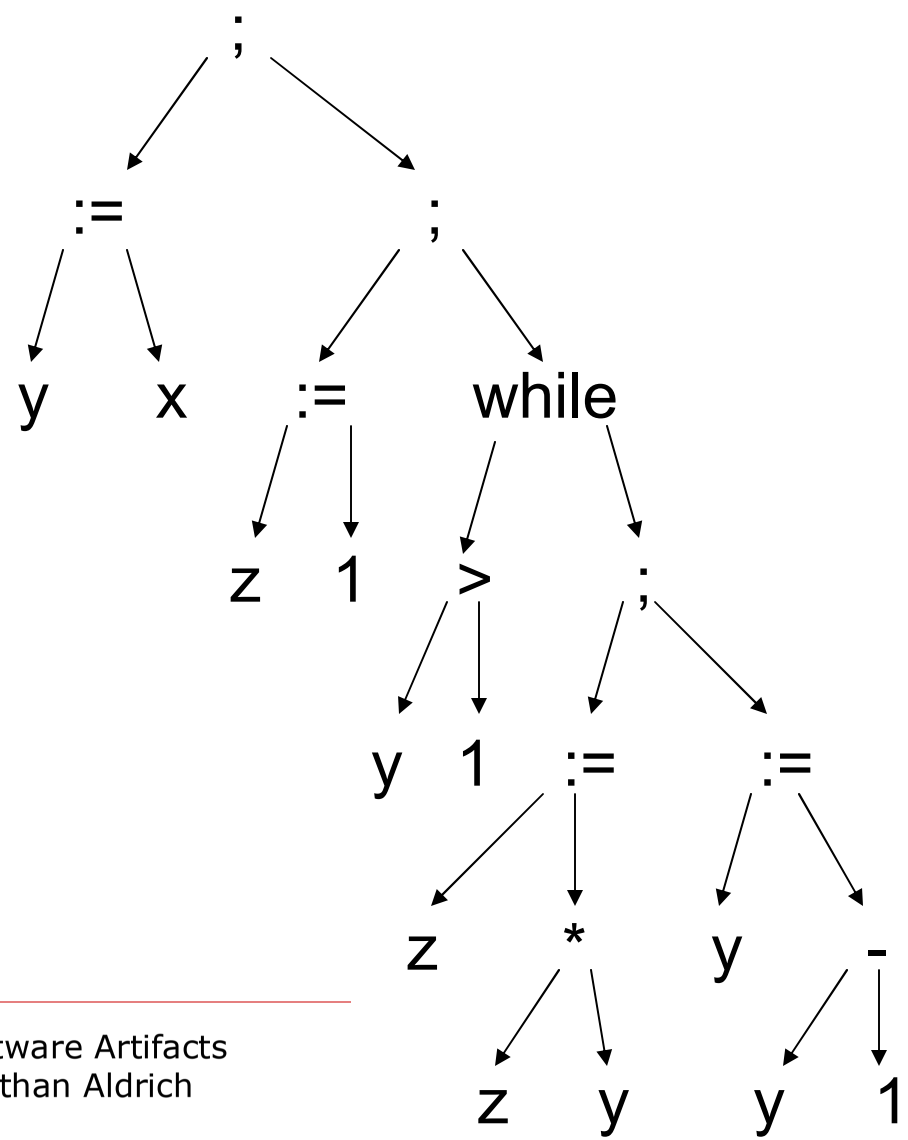
- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …



Parsing Example

```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …



Quick Quiz

Draw a parse tree for the function below. You can assume that the "for" statement is at the top of the parse tree.

```
void copy_bytes(char dest[], char source[], int n) {
    for (int i = 0; i < n; ++i)
        dest[i] = source[i];
}
```

WHILE ASTs in Java

- Java data structures mirror grammar
- $S ::= x := a$

```
class AST { … }
class Stmt extends AST { … }
class Assign extends Stmt {
        Var var;
        AExpr expr;
}
```

WHILE ASTs in Java

- Java data structures mirror grammar
- $S ::= x := a$
  $\mid \text{skip}$

```
class AST { … }
class Stmt extends AST { … }
class Assign extends Stmt {
      Var var;
      AExpr expr;
}
class Skip extends Stmt { }
```

WHILE ASTs in Java

- Java data structures mirror grammar
- $S ::= x := a$
  $\mid \text{skip}$
  $\mid S_1; S_2$

```
class AST { … }
class Stmt extends AST { … }
class Assign extends Stmt {
        Var var;
        AExpr expr;
}
class Skip extends Stmt { }
class Seq extends Stmt {
        Stmt left;
        Stmt right;
}
```

WHILE ASTs in Java

- Java data structures mirror grammar
- $S ::= x := a$
  $\mid \text{skip}$
  $\mid S_1; S_2$
  $\mid \text{if } b \text{ then } S_1 \text{ else } S_2$

```
class AST { … }
class Stmt extends AST { … }
class Assign extends Stmt {
        Var var;
        AExpr expr;
}
class Skip extends Stmt { }
class Seq extends Stmt {
        Stmt left;
        Stmt right;
}
class If extends Stmt {
        BExpr cond;
        Stmt thenStmt;
        Stmt elseStmt;
}
```

WHILE ASTs in Java

- Java data structures mirror grammar
- $S ::= x := a$
  $\mid$ skip
  $\mid S_1; S_2$
  $\mid$ if b then S_1 else S_2
  $\mid$ while b do S

```
class AST { … }
class Stmt extends AST { … }
class Assign extends Stmt {
      Var var;
      AExpr expr;
}
class Skip extends Stmt { }
class Seq extends Stmt {
      Stmt left;
      Stmt right;
}
class If extends Stmt {
      BExpr cond;
      Stmt thenStmt;
      Stmt elseStmt;
}
class While extends Stmt {
      BExpr cond;
      Stmt body;
}
```

Outline

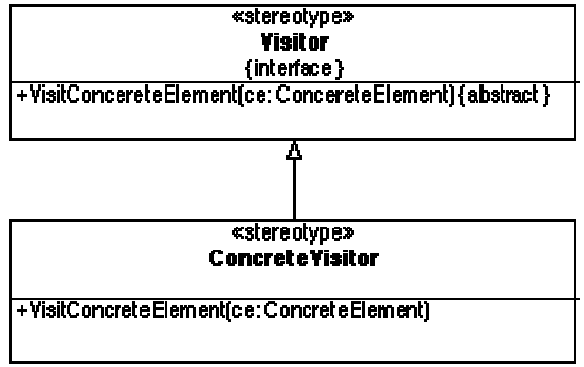
- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Representing programs
- AST-walking analyses
- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm

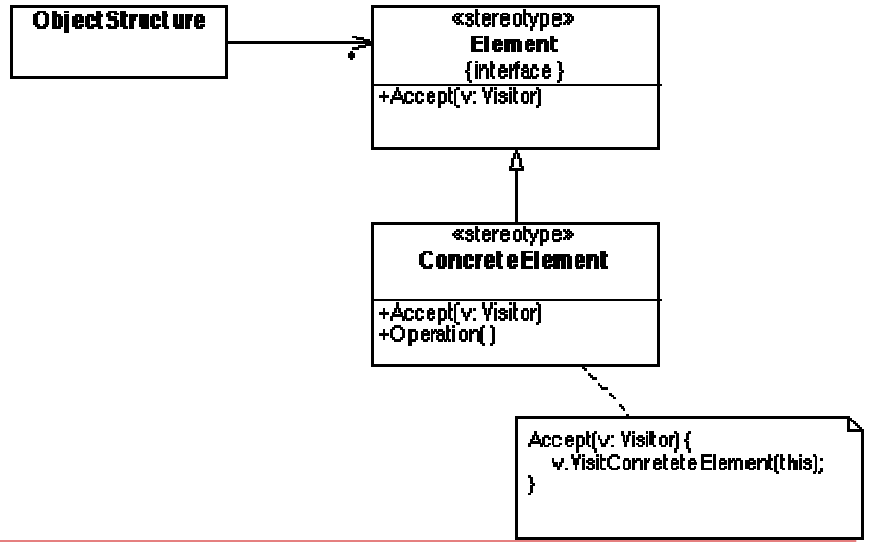
Matching AST against Bug Patterns

- AST Walker Analysis
  - Walk the AST, looking for nodes of a particular type
  - Check the immediate neighborhood of the node for a bug pattern
  - Warn if the node matches the pattern
- Semantic grep
  - Like grep, looking for simple patterns
  - Unlike grep, consider not just names, but semantic structure of AST
    - Makes the analysis more precise
- Common architecture based on Visitors
  - class Visitor has a visitX method for each type of AST node X
  - Default Visitor code just descends the AST, visiting each node
  - To find a bug in AST element of type X, override visitX

Behavioral Patterns: Visitor

- Applicability
  - Structure with many classes
  - Want to perform operations that depend on classes
  - Set of classes is stable
  - Want to define new operations
- Consequences
  - Easy to add new operations
  - Groups related behavior in Visitor
  - Adding new elements is hard
  - Visitor can store state
  - Elements must expose interface





Example: Shifting by more than 31 bits

```
class BadShiftAnalysis extends Visitor
    visitShiftExpression(ShiftExpression e) {
        if (type of e's left operand is int)
            if (e's right operand is a constant)
                if (value of constant < 0 or > 31)
                        warn("Shifting by less than 0 or more
    than 31 is meaningless")

        super.visitShiftExpression(e);
    }
```

Example: String concatenation in a loop

```
class StringConcatLoopAnalysis extends Visitor
    private int loopLevel = 0;

    visitStringConcat(StringConcat e) {
        if (loopLevel > 0)
            warn("Performance issue: String concatenation in loop (use
    StringBuffer instead)")
        super.visitStringConcat(e);       // visits AST children
    }

    visitWhile(While e) {
        loopLevel++;
        super.visitWhile(e);              // visits AST children
        loopLevel--;
    }
    // similar for other looping constructs
```

Example Tool: FindBugs

- Origin: research project at U. Maryland
  - Now freely available as open source
  - Standalone tool, plugins for Eclipse, etc.
- Checks over 250 “bug patterns”
  - Over 100 correctness bugs
  - Many style issues as well
  - Includes the two examples just shown
- Focus on simple, local checks
  - Similar to the patterns we’ve seen
  - But checks bytecode, not AST
    - Harder to write, but more efficient and doesn’t require source
- http://findbugs.sourceforge.net/

Example FindBugs Bug Patterns

- Correct equals()
- Use of ==
- Closing streams
- Illegal casts
- Null pointer dereference
- Infinite loops
- Encapsulation problems
- Inconsistent synchronization
- Inefficient String use
- Dead store to variable

FindBugs Experiences

- Useful for learning idioms of Java
  - Rules about libraries and interfaces
    - e.g. equals()
- Customization is important
  - Many warnings may be irrelevant, others may be important – depends on domain
    - e.g. embedded system vs. web application
- Useful for pointing out things to examine
  - Not all are real defects
  - Turn off false positive warnings for future analyses on codebase

Outline

- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Representing programs
- AST-walking analyses
- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm

An Analysis We've Seen

- Hoare logic
  - Useful for proving correctness
  - Requires a lot of work (even for ESC/Java)
  - Automated tool is unsound
    - So is manual proof, without a proof checker

Motivation: Dataflow Analysis

- Catch interesting errors
  - Non-local: x is null, x is written to y, y is dereferenced
- Optimize code
  - Reduce run time, memory usage…
- Soundness required
  - Safety-critical domain
    - Assure lack of certain errors
  - Cannot optimize unless it is proven safe
    - Correctness comes before performance
- Automation required
  - Dramatically decreases cost
  - Makes cost/benefit worthwhile for far more purposes

Dataflow analysis

- Tracks value flow through program
  - Can distinguish order of operations
    - Did you read the file after you closed it?
    - Does this null value flow to that dereference?
  - Differs from AST walker
    - Walker simply collects information or checks patterns
    - Tracking flow allows more interesting properties
- Abstracts values
  - Chooses abstraction particular to property
    - Is a variable null?
    - Is a file open or closed?
    - Could a variable be 0?
    - Where did this value come from?
  - More *specialized* than Hoare logic
    - Hoare logic allows any property to be expressed
    - Specialization allows automation and soundness

Zero Analysis

- Could variable x be 0?
  - Useful to know if you have an expression y/x
  - In C, useful for null pointer analysis
- Program semantics
  - η maps every variable to an integer
- Semantic abstraction
  - σ maps every variable to non zero (NZ), zero(Z), or maybe zero (MZ)
  - Abstraction function for integers α_{ZI} :
    - $\alpha_{ZI}(0)$ = Z
    - $\alpha_{ZI}(n)$ = NZ for all $n \neq 0$
  - We may not know if a value is zero or not
    - Analysis is always an approximation
    - Need MZ option, too

Zero Analysis Example

σ =[]

```
x := 10;
y := x;
z := 0;
while y > -1 do
    x := x / y;
    y := y-1;
    z := 5;
```

Zero Analysis Example

σ =[]

x := 10;

σ =[x↦α_{ZI}(10)]

y := x;

z := 0;

while y > -1 do

    x := x / y;

    y := y-1;

    z := 5;

Zero Analysis Example

σ =[]

x := 10;

σ =[x↦NZ]

y := x;

z := 0;

while y > -1 do

    x := x / y;

    y := y-1;

    z := 5;

Zero Analysis Example

```
x := 10;
y := x;
z := 0;
while y > -1 do
    x := x / y;
    y := y-1;
    z := 5;
```

σ =[]

σ =[x↦NZ]

σ =[x↦NZ,y↦σ(x)]

Zero Analysis Example

```
x := 10;
y := x;
z := 0;
while y > -1 do
    x := x / y;
    y := y-1;
    z := 5;
```

σ =[]

σ =[x↦NZ]

σ =[x↦NZ,y↦NZ]

Zero Analysis Example

```
x := 10;
y := x;
z := 0;
while y > -1 do
    x := x / y;
    y := y-1;
    z := 5;
```

σ =[]

σ =[x↦NZ]

σ =[x↦NZ,y↦NZ]

σ =[x↦NZ,y↦NZ,z↦α_{ZI}(0)]

Zero Analysis Example

```
x := 10;
y := x;
z := 0;
while y > -1 do
    x := x / y;
    y := y-1;
    z := 5;
```

σ =[]

σ =[x↦NZ]

σ =[x↦NZ,y↦NZ]

σ =[x↦NZ,y↦NZ,z↦Z]

Zero Analysis Example

```
x := 10;
y := x;
z := 0;
while y > -1 do
    x := x / y;
    y := y-1;
    z := 5;
```

σ =[]

σ =[x↦NZ]

σ =[x↦NZ,y↦NZ]

σ =[x↦NZ,y↦NZ,z↦Z]

σ =[x↦NZ,y↦NZ,z↦Z]

Zero Analysis Example

```
x := 10;
y := x;
z := 0;
while y > -1 do
    x := x / y;
    y := y-1;
    z := 5;
```

σ =[]

σ =[x↦NZ]

σ =[x↦NZ,y↦NZ]

σ =[x↦NZ,y↦NZ,z↦Z]

σ =[x↦NZ,y↦NZ,z↦Z]

σ =[x↦NZ,y↦NZ,z↦Z]

Zero Analysis Example

	σ =[]
x := 10;	σ =[x↦NZ]
y := x;	σ =[x↦NZ,y↦NZ]
z := 0;	σ =[x↦NZ,y↦NZ,z↦Z]
while y > -1 do	σ =[x↦NZ,y↦NZ,z↦Z]
x := x / y;	σ =[x↦NZ,y↦NZ,z↦Z]
y := y-1;	σ =[x↦NZ,y↦MZ,z↦Z]
z := 5;	

Zero Analysis Example

	σ =[]
x := 10;	σ =[x↦NZ]
y := x;	σ =[x↦NZ,y↦NZ]
z := 0;	σ =[x↦NZ,y↦NZ,z↦Z]
while y > -1 do	σ =[x↦NZ,y↦NZ,z↦Z]
x := x / y;	σ =[x↦NZ,y↦NZ,z↦Z]
y := y-1;	σ =[x↦NZ,y↦MZ,z↦Z]
z := 5;	σ =[x↦NZ,y↦MZ,z↦NZ]

Zero Analysis Example

	σ =[]
x := 10;	σ =[x↦NZ]
y := x;	σ =[x↦NZ,y↦NZ]
z := 0;	σ =[x↦NZ,y↦NZ,z↦Z]
while y > -1 do	σ =[x↦NZ,y↦MZ,z↦MZ]
x := x / y;	σ =[x↦NZ,y↦NZ,z↦Z]
y := y-1;	σ =[x↦NZ,y↦MZ,z↦Z]
z := 5;	σ =[x↦NZ,y↦MZ,z↦NZ]

Zero Analysis Example

	σ =[]
x := 10;	σ =[x↦NZ]
y := x;	σ =[x↦NZ,y↦NZ]
z := 0;	σ =[x↦NZ,y↦NZ,z↦Z]
while y > -1 do	σ =[x↦NZ,y↦MZ,z↦MZ]
x := x / y;	σ =[x↦NZ,y↦MZ,z↦MZ]
y := y-1;	σ =[x↦NZ,y↦MZ,z↦Z]
z := 5;	σ =[x↦NZ,y↦MZ,z↦NZ]

Zero Analysis Example

	σ =[]
x := 10;	σ =[x↦NZ]
y := x;	σ =[x↦NZ,y↦NZ]
z := 0;	σ =[x↦NZ,y↦NZ,z↦Z]
while y > -1 do	σ =[x↦NZ,y↦MZ,z↦MZ]
x := x / y;	σ =[x↦NZ,y↦MZ,z↦MZ]
y := y-1;	σ =[x↦NZ,y↦MZ,z↦MZ]
z := 5;	σ =[x↦NZ,y↦MZ,z↦NZ]

Zero Analysis Example

	σ =[]
x := 10;	σ =[x↦NZ]
y := x;	σ =[x↦NZ,y↦NZ]
z := 0;	σ =[x↦NZ,y↦NZ,z↦Z]
while y > -1 do	σ =[x↦NZ,y↦MZ,z↦MZ]
x := x / y;	σ =[x↦NZ,y↦MZ,z↦MZ]
y := y-1;	σ =[x↦NZ,y↦MZ,z↦MZ]
z := 5;	σ =[x↦NZ,y↦MZ,z↦NZ]
	Nothing more happens!

Zero Analysis Termination

- The analysis values will not change, no matter how many times we execute the loop
  - Proof: our analysis is deterministic
  - We run through the loop with the current analysis values, none of them change
  - Therefore, no matter how many times we run the loop, the results will remain the same
  - Therefore, we have computed the dataflow analysis results for any number of loop iterations

Zero Analysis Termination

- The analysis values will not change, no matter how many times we execute the loop
  - Proof: our analysis is deterministic
  - We run through the loop with the current analysis values, none of them change
  - Therefore, no matter how many times we run the loop, the results will remain the same
  - Therefore, we have computed the dataflow analysis results for any number of loop iterations
- Why does this work
  - If we simulate the loop, the data values could (in principle) keep changing indefinitely
    - There are an infinite number of data values possible
    - Not true for 32-bit integers, but might as well be true
      - Counting to 2^{32} is slow, even on today's processors
  - Dataflow analysis only tracks 2 possibilities!
    - So once we've explored them all, nothing more will change
    - This is the secret of abstraction
- We will make this argument more precise later

Using Zero Analysis

- Visit each division in the program
- Get the results of zero analysis for the divisor
- If the results are definitely zero, report an error
- If the results are possibly zero, report a warning

Quick Quiz

- Fill in the table to show how what information zero analysis will compute for the function given.

Program Statement	Analysis Info after that statement
0: <*beginning of program*>	
1: x := 0	
2: y := 1	
3: if (z == 0)	
4: x := x + y	
5: else y := y – 1	
6: w := y	

Outline

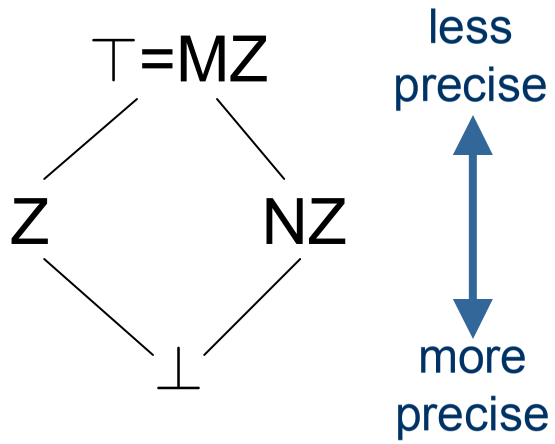
- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Representing programs
- AST-walking analyses
- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm

Defining Dataflow Analyses

- Lattice
  - Describes program data abstractly
  - Abstract equivalent of environment
- Abstraction function
  - Maps concrete environment to lattice element
- Flow functions
  - Describes how abstract data changes
  - Abstract equivalent of expression semantics
- Control flow graph
  - Determines how abstract data propagates from statement to statement
  - Abstract equivalent of statement semantics

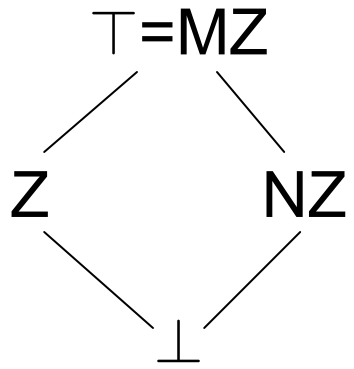
Lattice

- A lattice is a tuple (L, $\sqsubseteq$, $\sqcup$, $\bot$, $\top$)
  - L is a set of abstract elements
  - $\sqsubseteq$ is a partial order on L
    - Means *at least as precise as*
  - $\sqcup$ is the least upper bound of two elements
    - Must exist for every two elements in L
    - Used to merge two abstract values
  - $\bot$ (bottom) is the least element of L
    - Means we haven't yet analyzed this yet
    - Will become clear later
  - $\top$ (top) is the greatest element of L
    - Means we don't know anything
- L may be infinite
  - Typically should have finite height
    - All paths from $\bot$ to $\top$ should be finite
    - We'll see why later



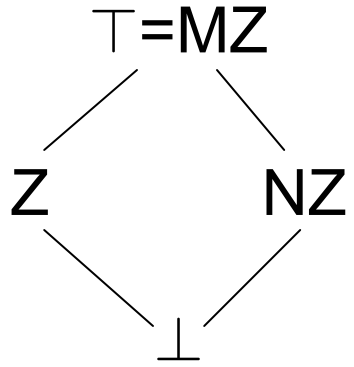
Zero Analysis Lattice

- Integer zero lattice
  - L_{ZI} = { ⊥, Z, NZ, MZ }
  - ⊥ ⊑ Z, ⊥ ⊑ NZ, NZ ⊑ MZ, Z ⊑ MZ
    - ⊥ ⊑ MZ holds by transitivity
  - ⊔ defined as join for ⊑
    - x ⊔ y = z iff
      - z is an upper bound of x and y
      - z is the least such bound
    - Obeys laws: $\bot \sqcup \mathcal{X} = \mathcal{X}$, $\top \sqcup \mathcal{X} = \top$, $\mathcal{X} \sqcup \mathcal{X} = \mathcal{X}$
    - Also Z ⊔ NZ = MZ
  - ⊥ = ⊥
    - $\forall \mathcal{X} . \bot \sqsubseteq \mathcal{X}$
  - ⊤ = MZ
    - $\forall \mathcal{X} . \mathcal{X} \sqsubseteq \top$



Zero Analysis Lattice

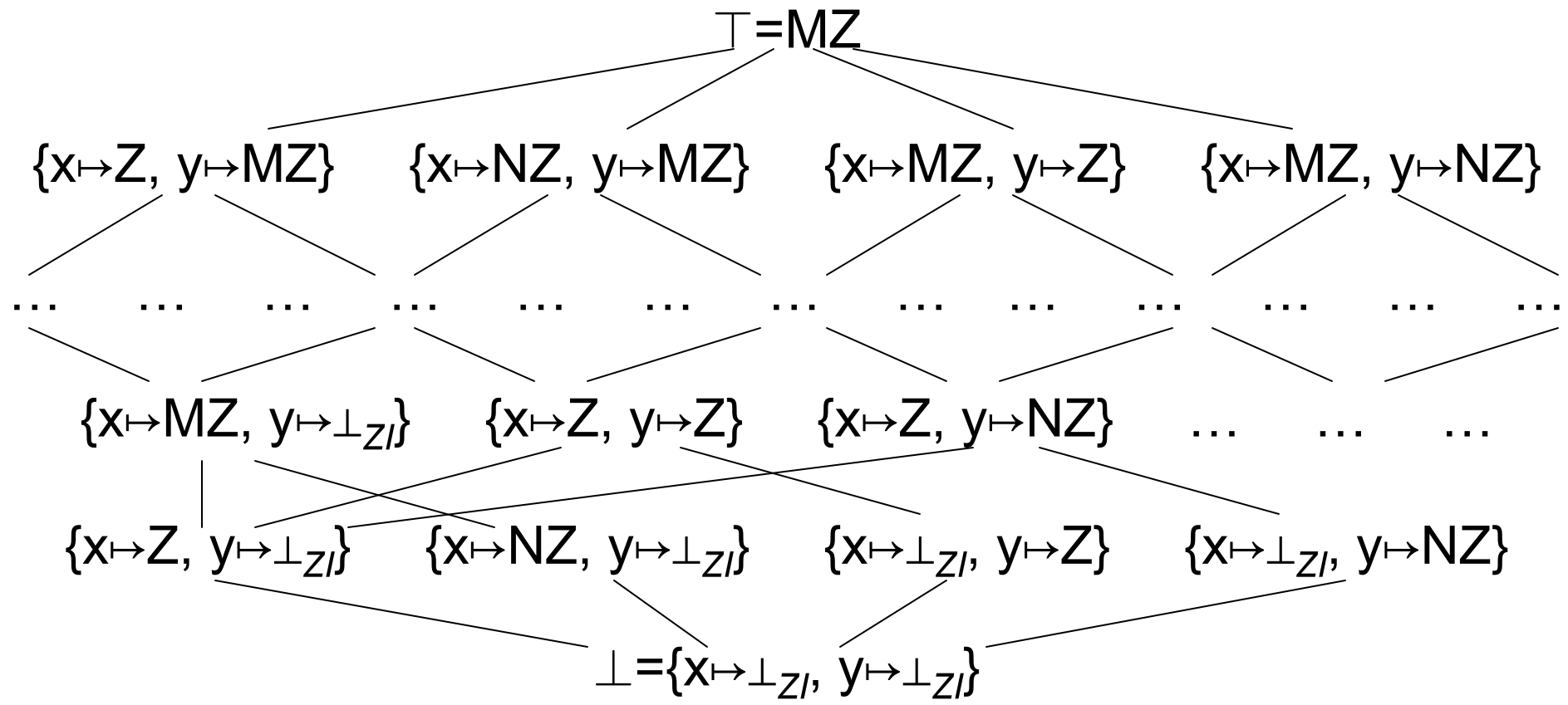
- Integer zero lattice
  - L_{ZI} = { ⊥, Z, NZ, MZ }
  - ⊥ ⊑ Z, ⊥ ⊑ NZ, NZ ⊑ MZ, Z ⊑ MZ
  - ⊔ defined as join for ⊑
  - ⊥ = ⊥
  - ⊤ = MZ



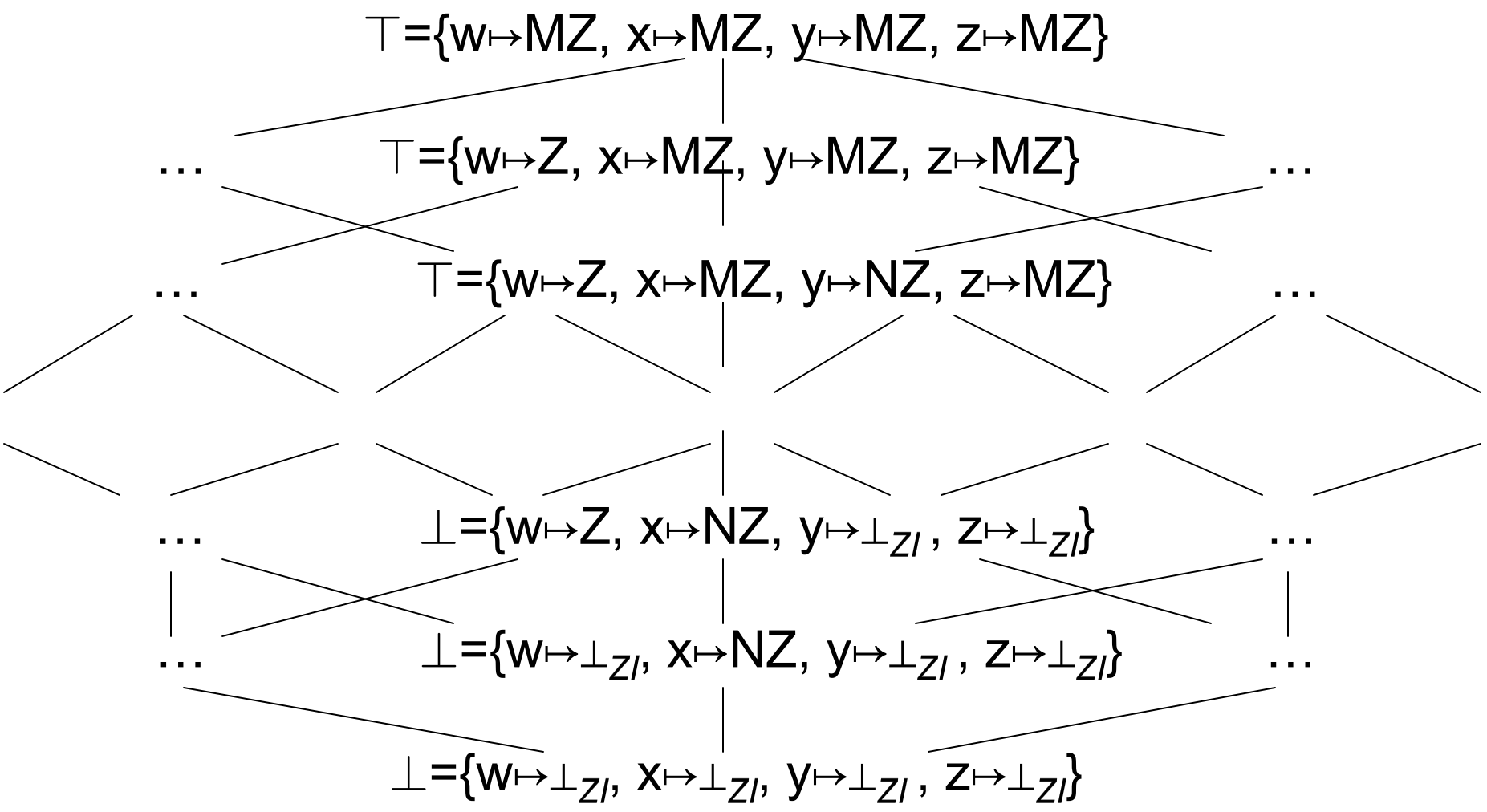
- Program lattice is a *tuple lattice*
  - L_Z is the set of all maps from **Var** to L_{ZI}
  - $\sigma_1 \sqsubseteq_Z \sigma_2$ iff $\forall x \in \mathbf{Var} \,.\, \sigma_1(x) \sqsubseteq_{ZI} \sigma_2(x)$
  - $\sigma_1 \sqcup_Z \sigma_2 = \{ x \mapsto \sigma_1(x) \sqcup_{ZI} \sigma_2(x) \mid x \in \mathbf{Var} \}$
  - $\bot = \{ x \mapsto \bot_{ZI} \mid x \in \mathbf{Var} \}$
  - $\top = \{ x \mapsto \top_{ZI} \mid x \in \mathbf{Var} \} = \{ x \mapsto MZ \mid x \in \mathbf{Var} \}$
  - Can produce a tuple lattice from *any* base lattice
    - Just define as above

Tuple Lattices Visually

- For **Var** = { x,y }



One Path in a Tuple Lattice



Outline

- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Representing programs
- AST-walking analyses
- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm

Abstraction Function

- Maps each concrete program state to a lattice element
  - For tuple lattices, the function can be defined for values and lifted to tuples
- Integer Zero abstraction function α_{ZI}:
  - $\alpha_{ZI}(0) = Z$
  - $\alpha_{ZI}(n) = NZ$ for all $n \neq 0$
- Zero Analysis abstraction function α_{ZA}:
  - $\alpha_{ZA}(\eta) = \{x \mapsto \alpha_{ZI}(\eta(x)) \mid x \in \textbf{Var}\}$
  - This is just the tuple form of $\alpha_{ZI}(n)$
    - Can be done for any tuple lattice

Quick Quiz

- Consider the following two tuple lattice values: [x↦Z, y↦MZ] and [x↦MZ, y↦NZ]
  - How do the two compare in the lattice ordering for zero analysis?
  - What is the join of these two tuple lattice values?