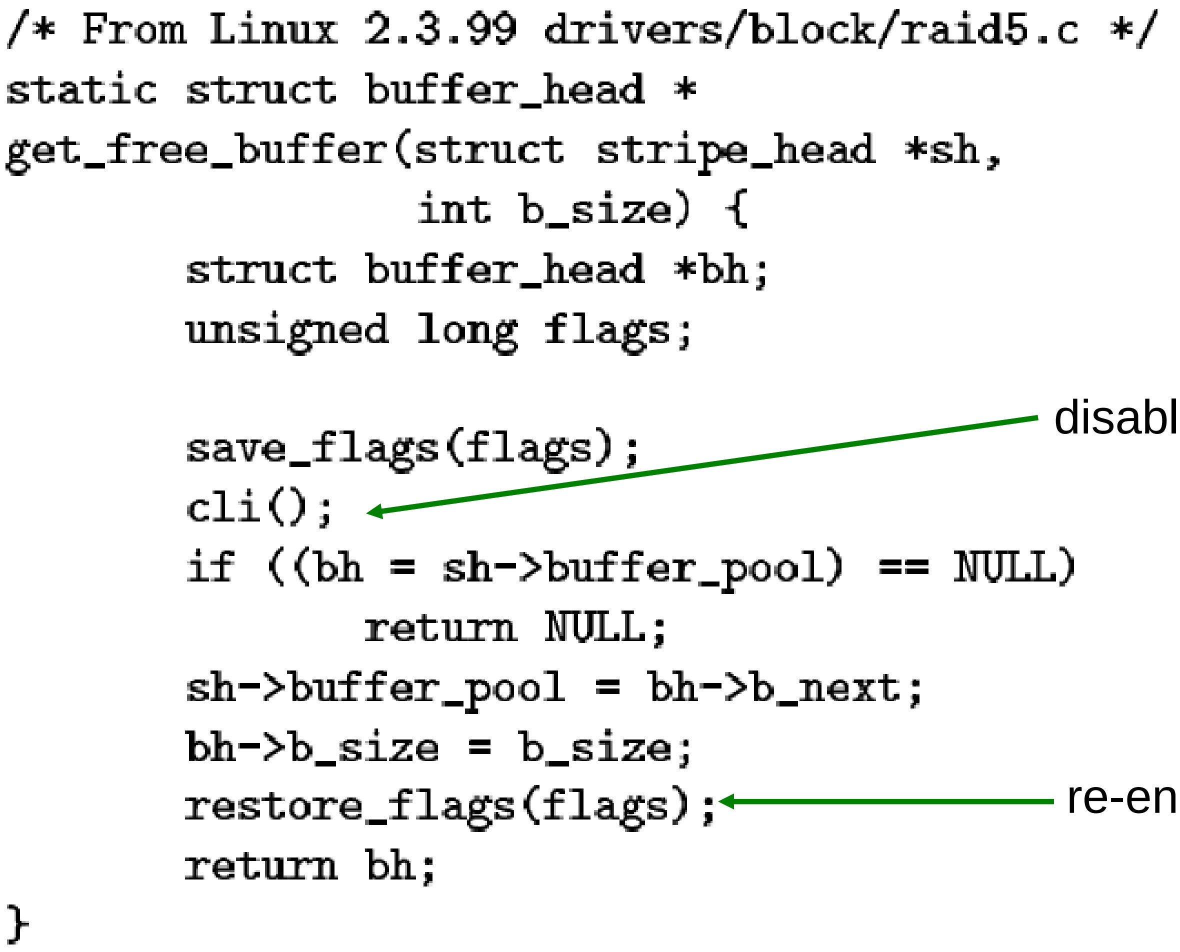
17-355/17-665: Program Analysis

Introduction to Program Analysis

Jonathan Aldrich

Find the Bug!

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



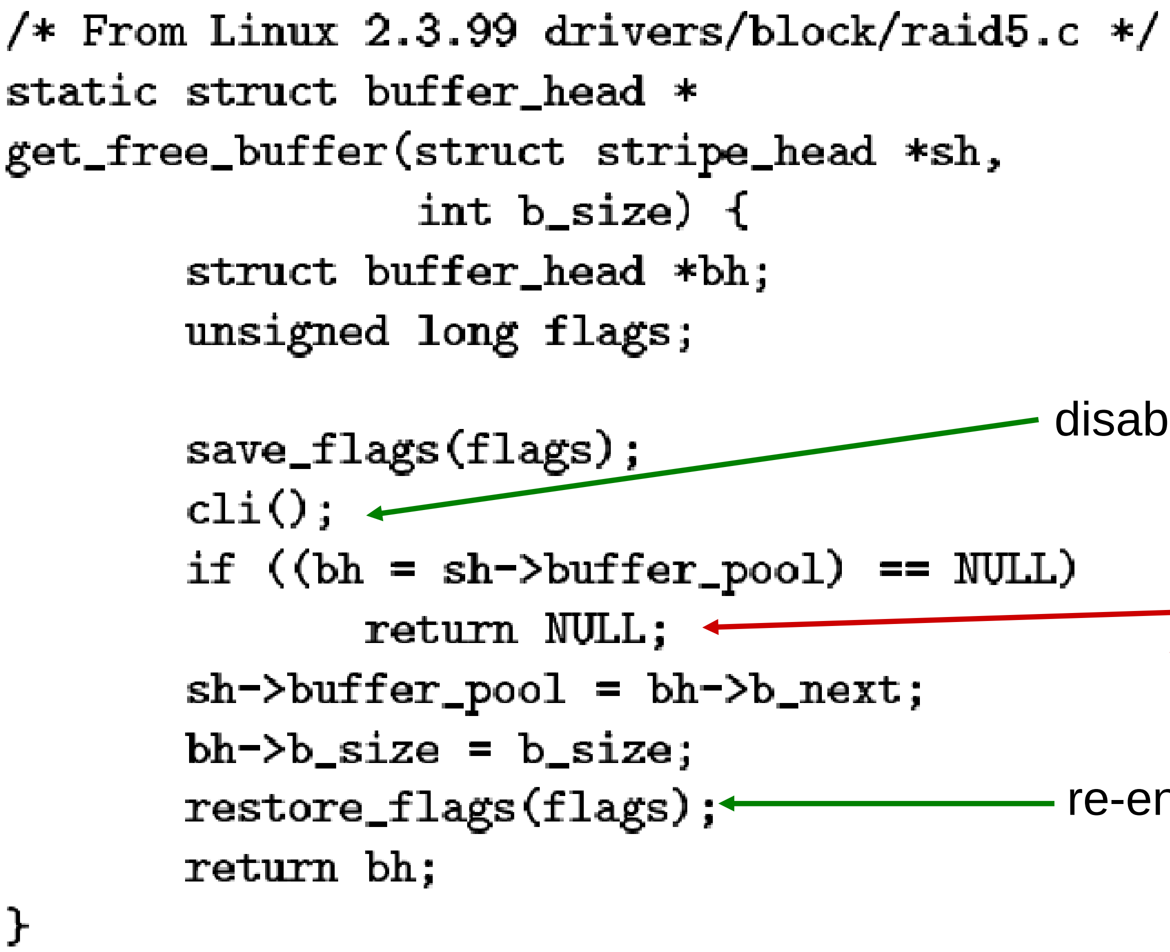
```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        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;
}
```

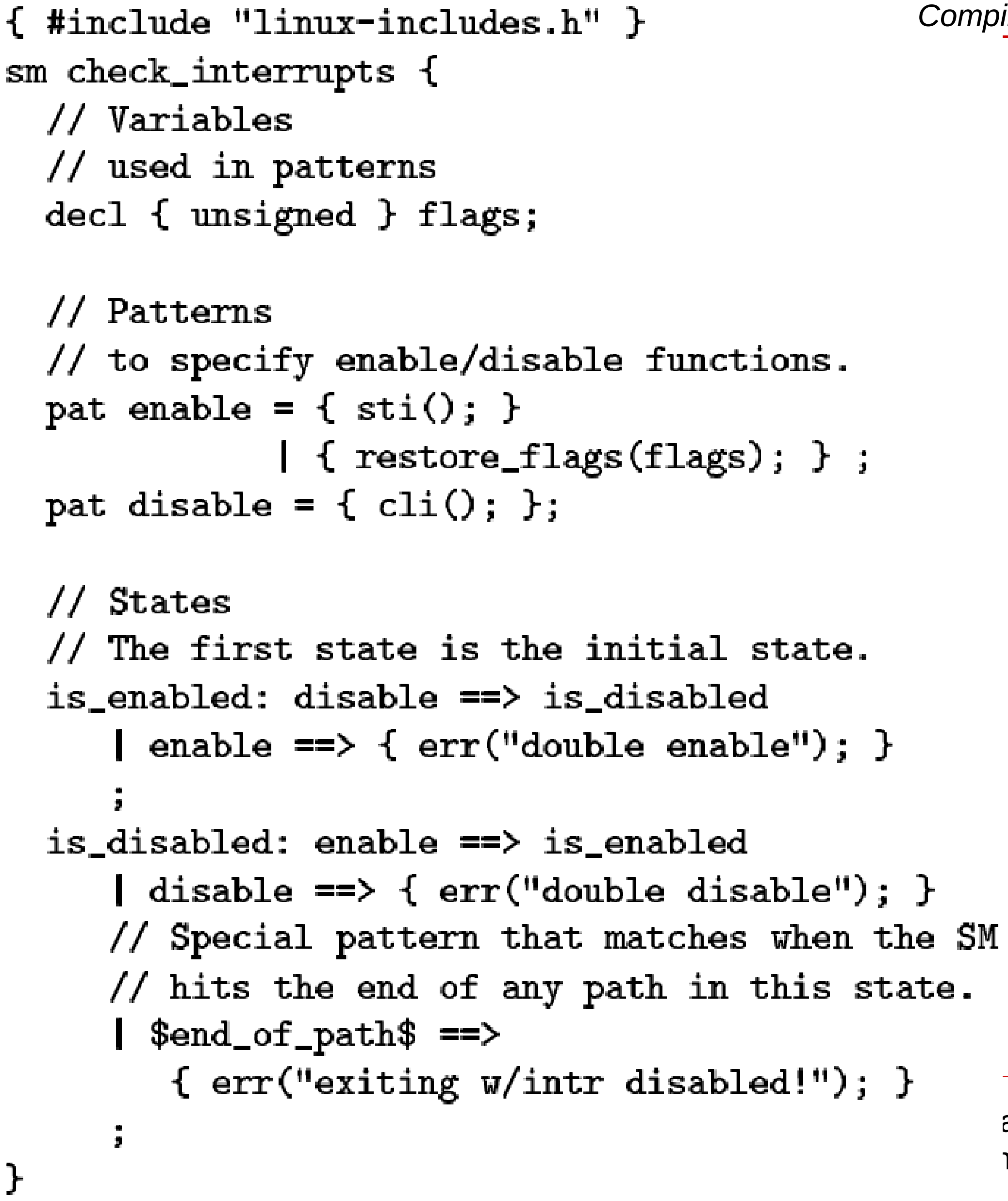
disable interrupts

ERROR: returning with interrupts disabled

re-enable interrupts

Metal Interrupt Analysis

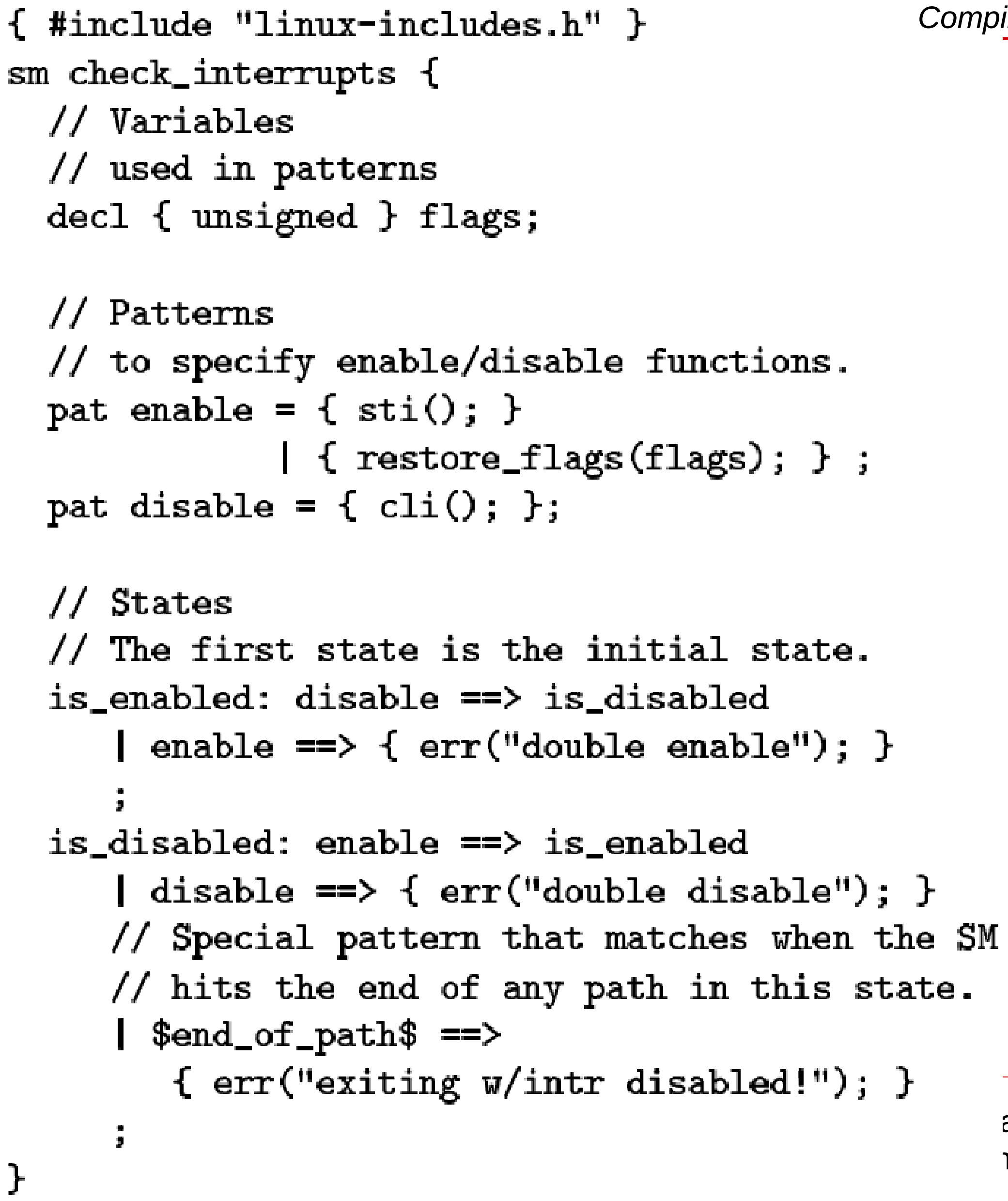
Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI ’00.



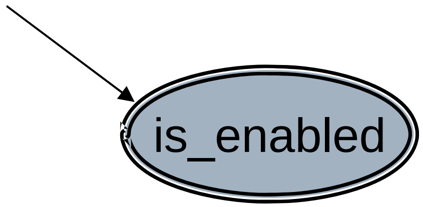
```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // 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.

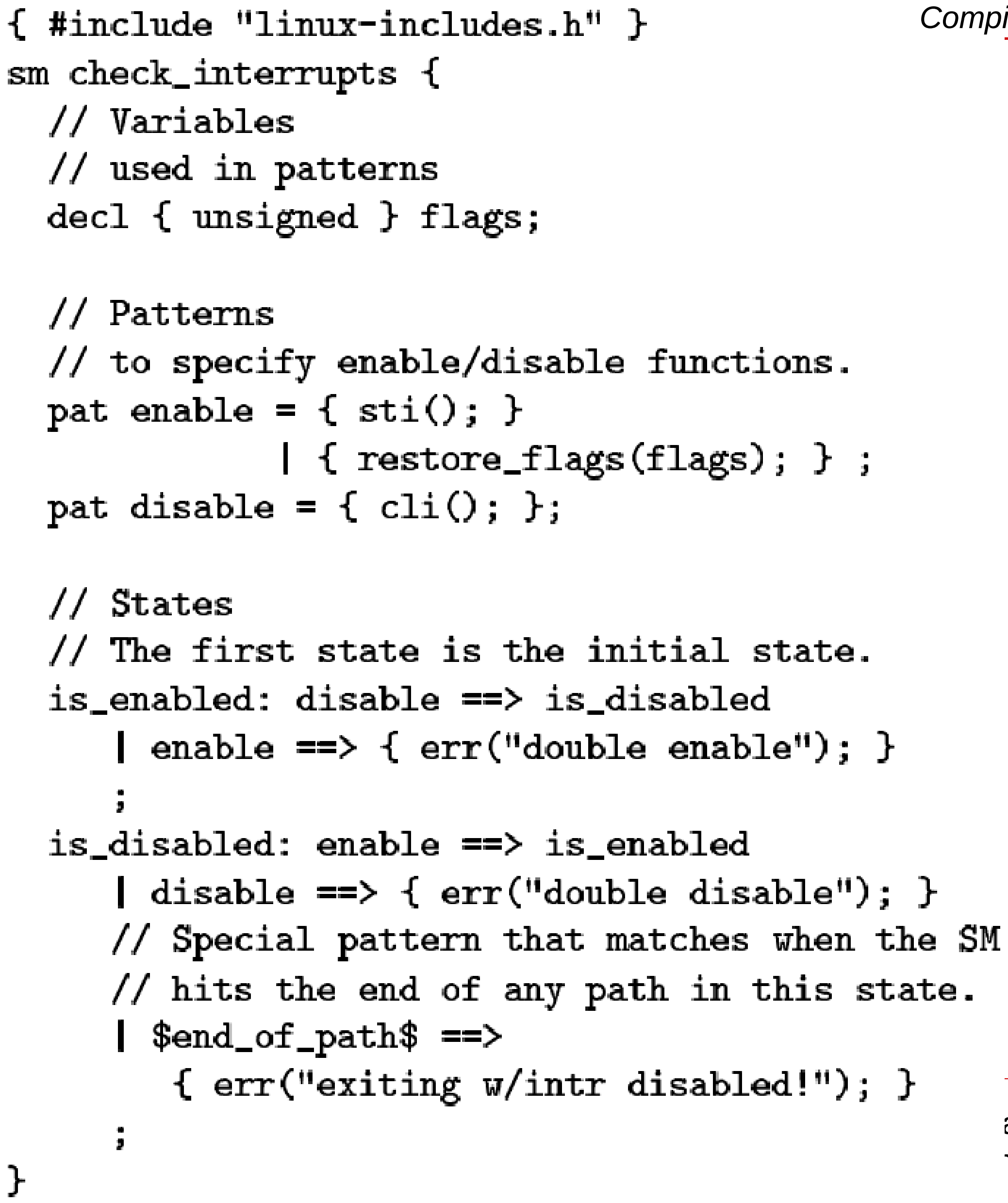


```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // 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!"); }
     ;
}
```

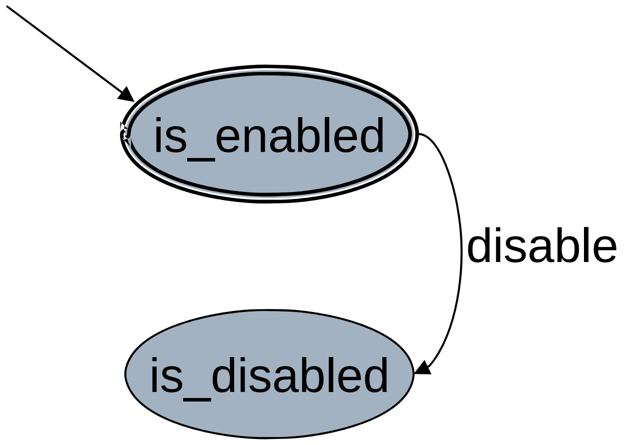


Metal Interrupt Analysis

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

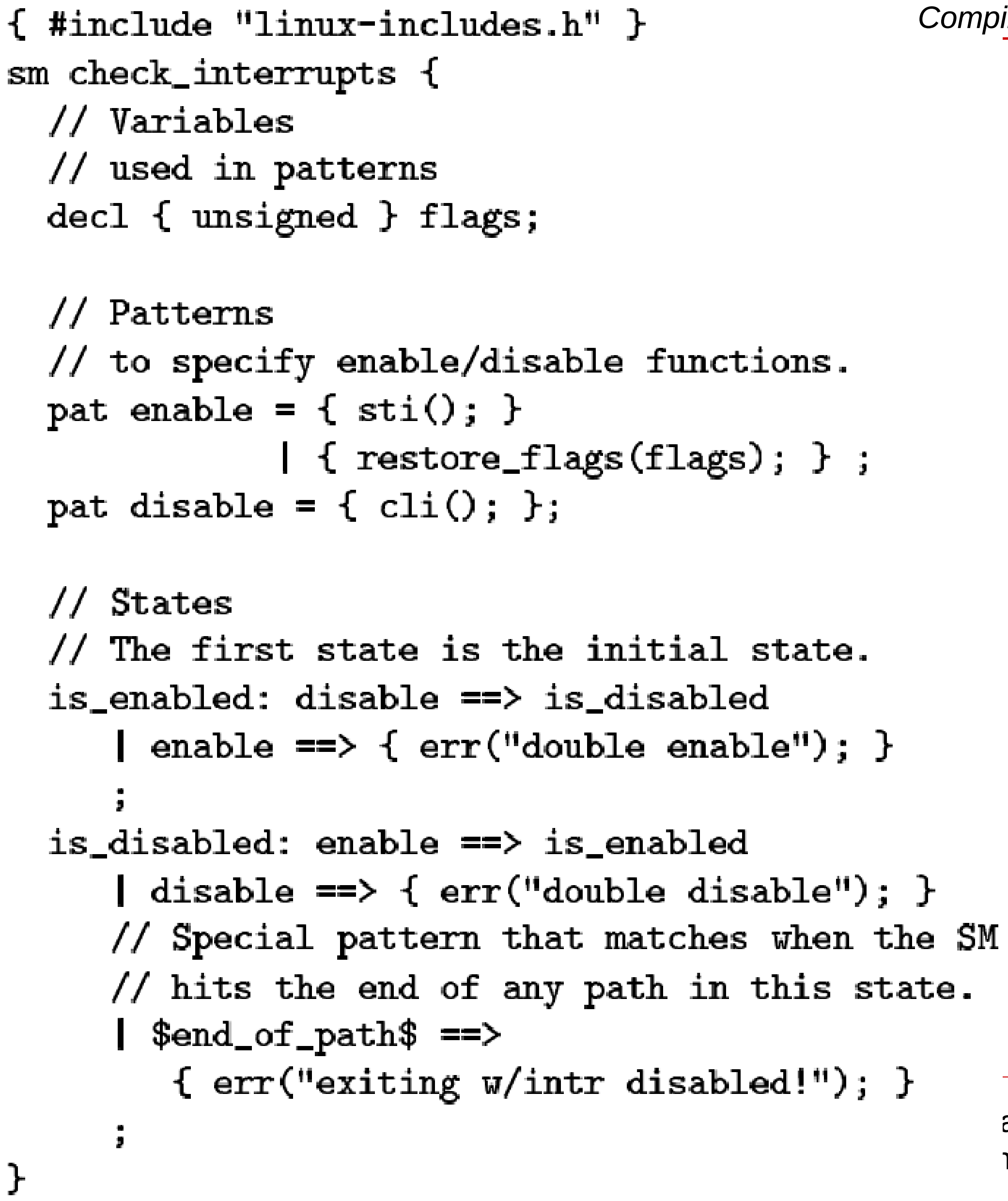


```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // 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!"); }
     ;
}
```

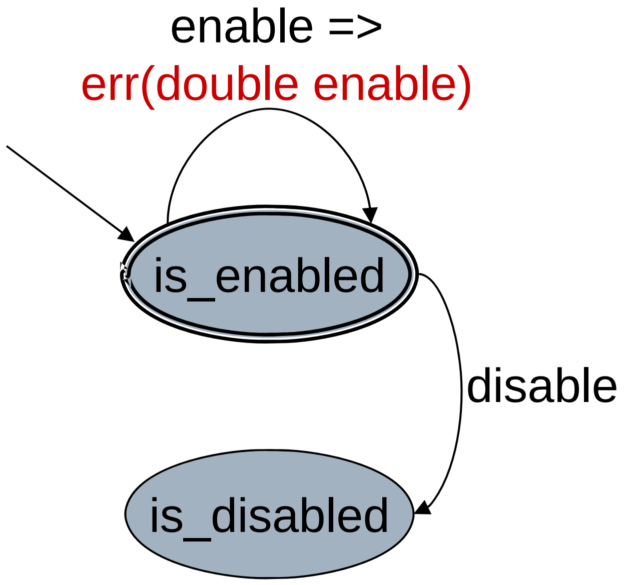


Metal Interrupt Analysis

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!"); }
     ;
}
```



enable =>

err(double enable)

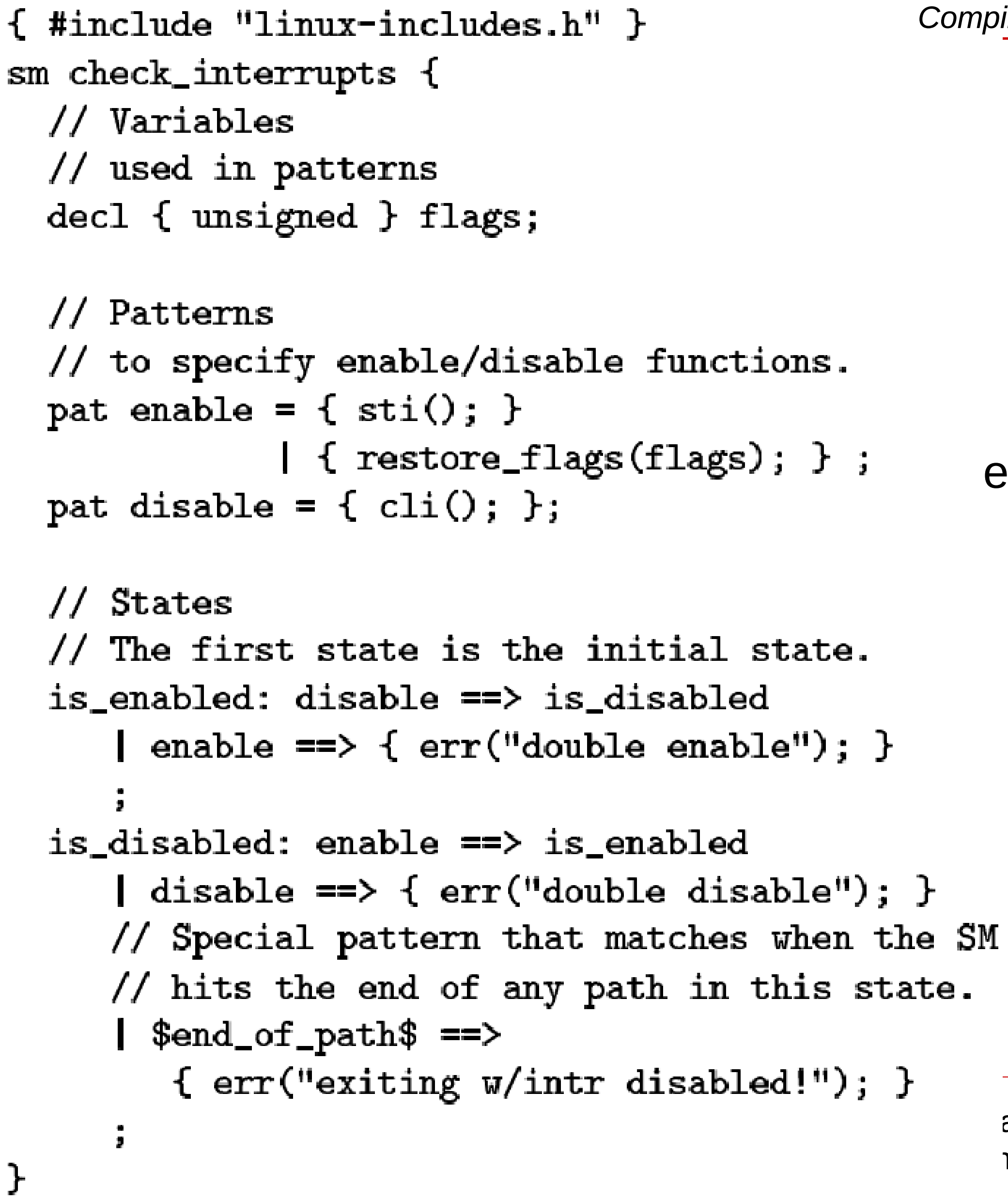
is_enabled

disable

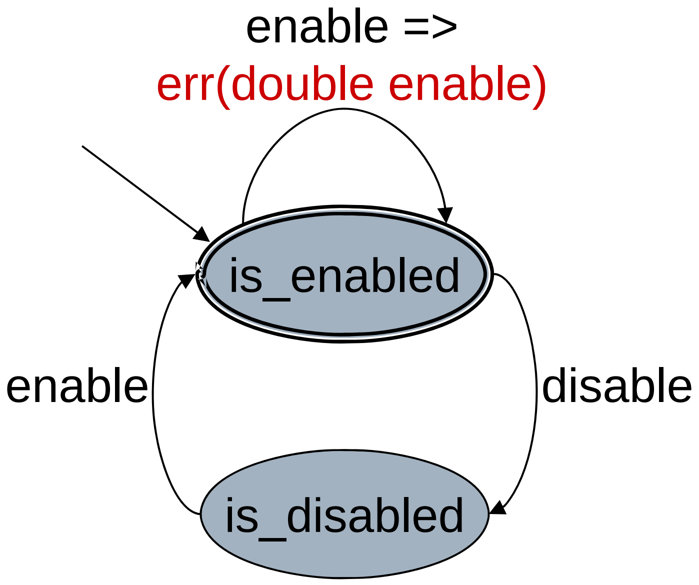
is_disabled

Metal Interrupt Analysis

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

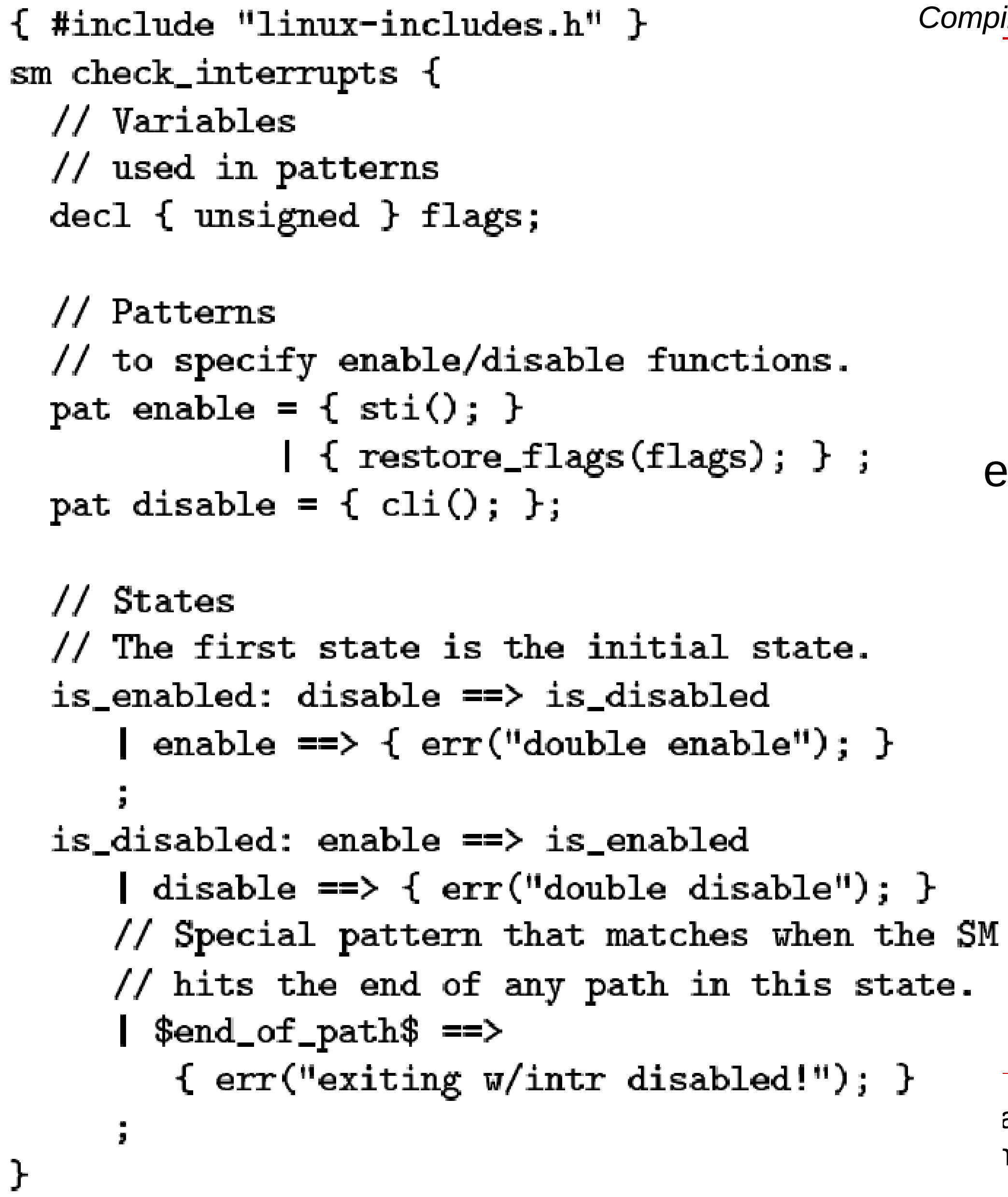


```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // 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!"); }
     ;
}
```

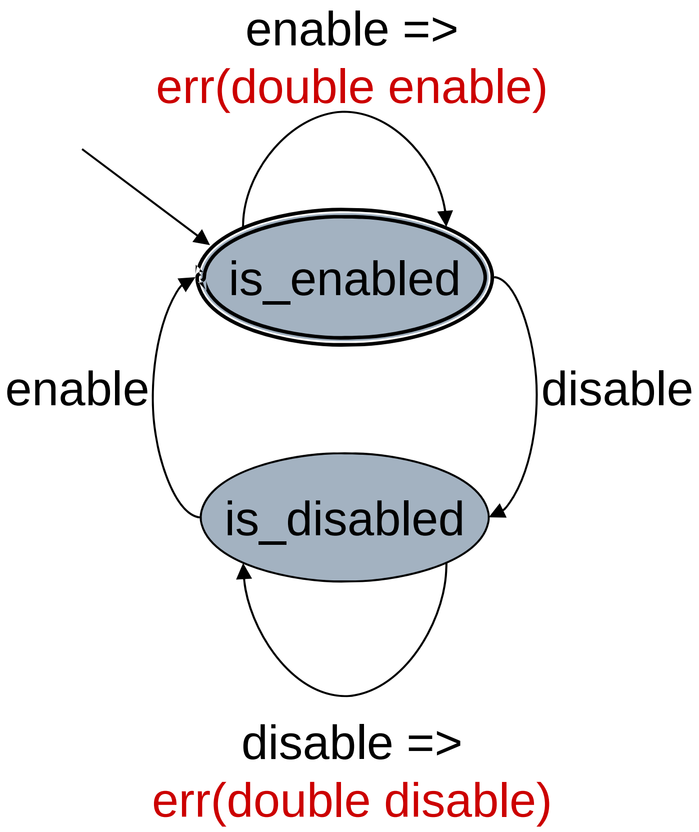


Metal Interrupt Analysis

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

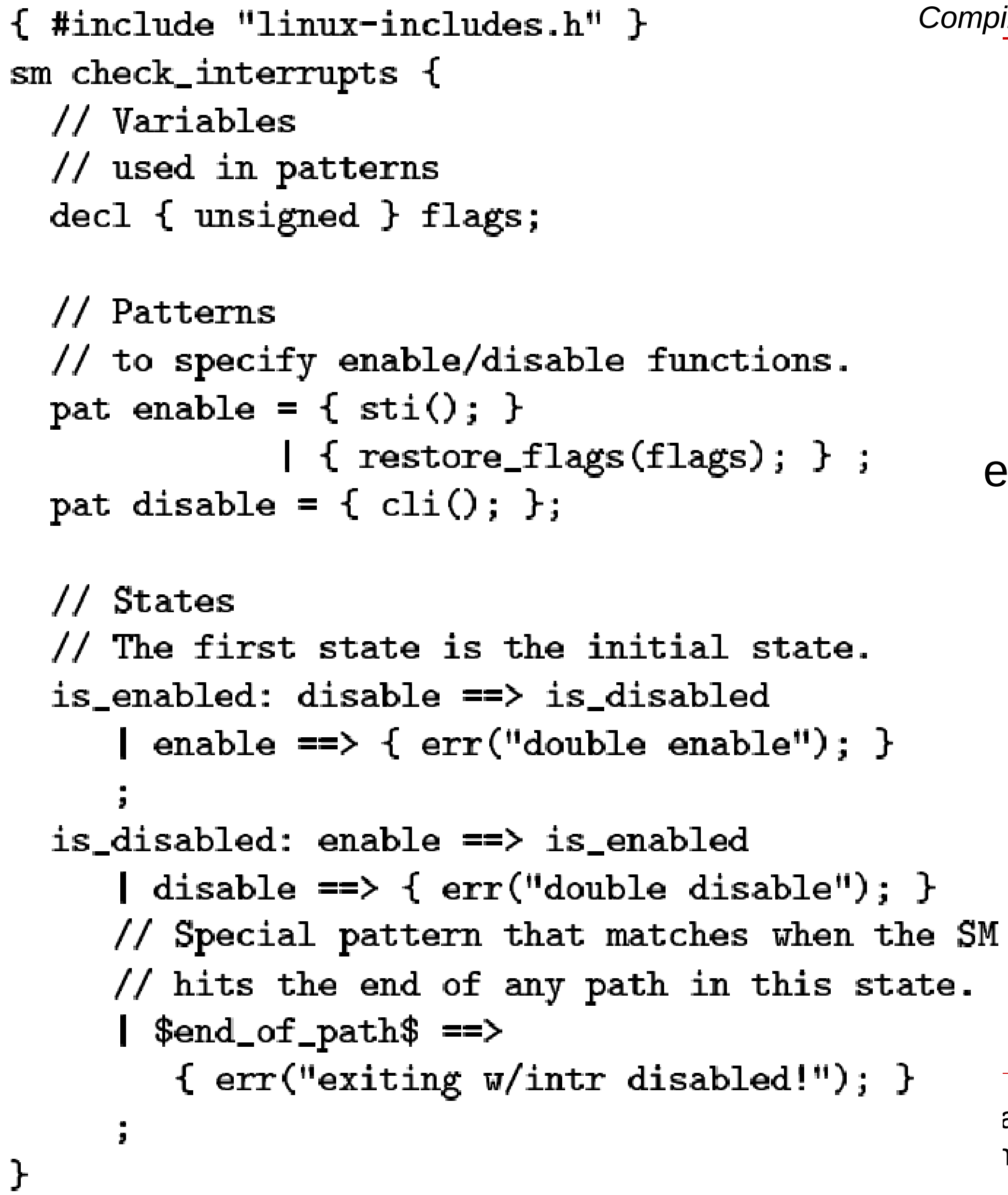


```
{ #include "linux-includes.h" }
sm check_interrupts {
  // Variables
  // used in patterns
  decl { unsigned } flags;

  // 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!"); }
     ;
}
```

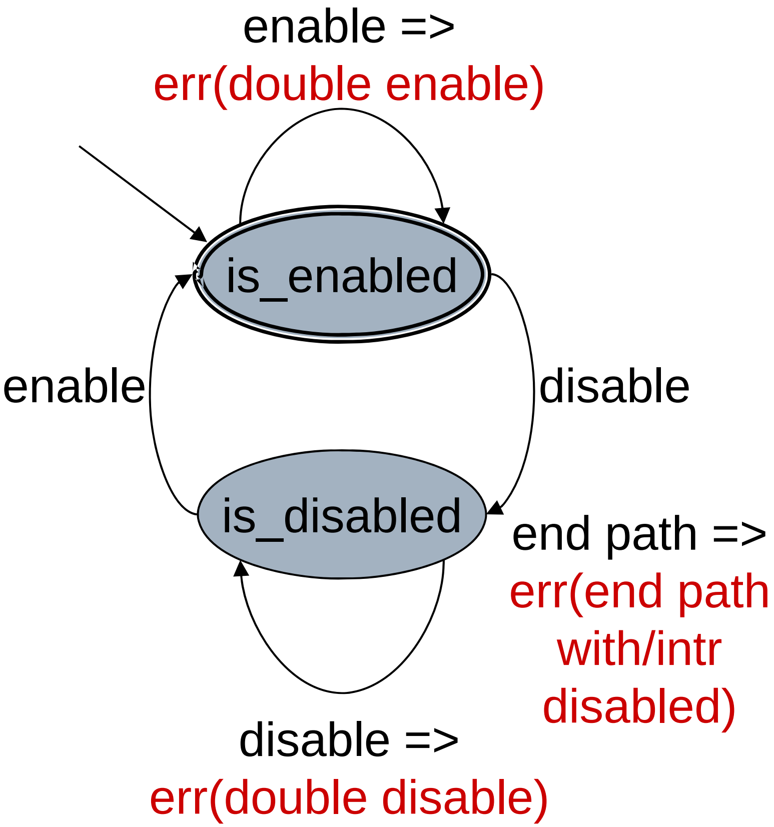


Metal Interrupt Analysis

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!"); }
     ;
}
```



enable =>
err(double enable)

is_enabled

enable

disable

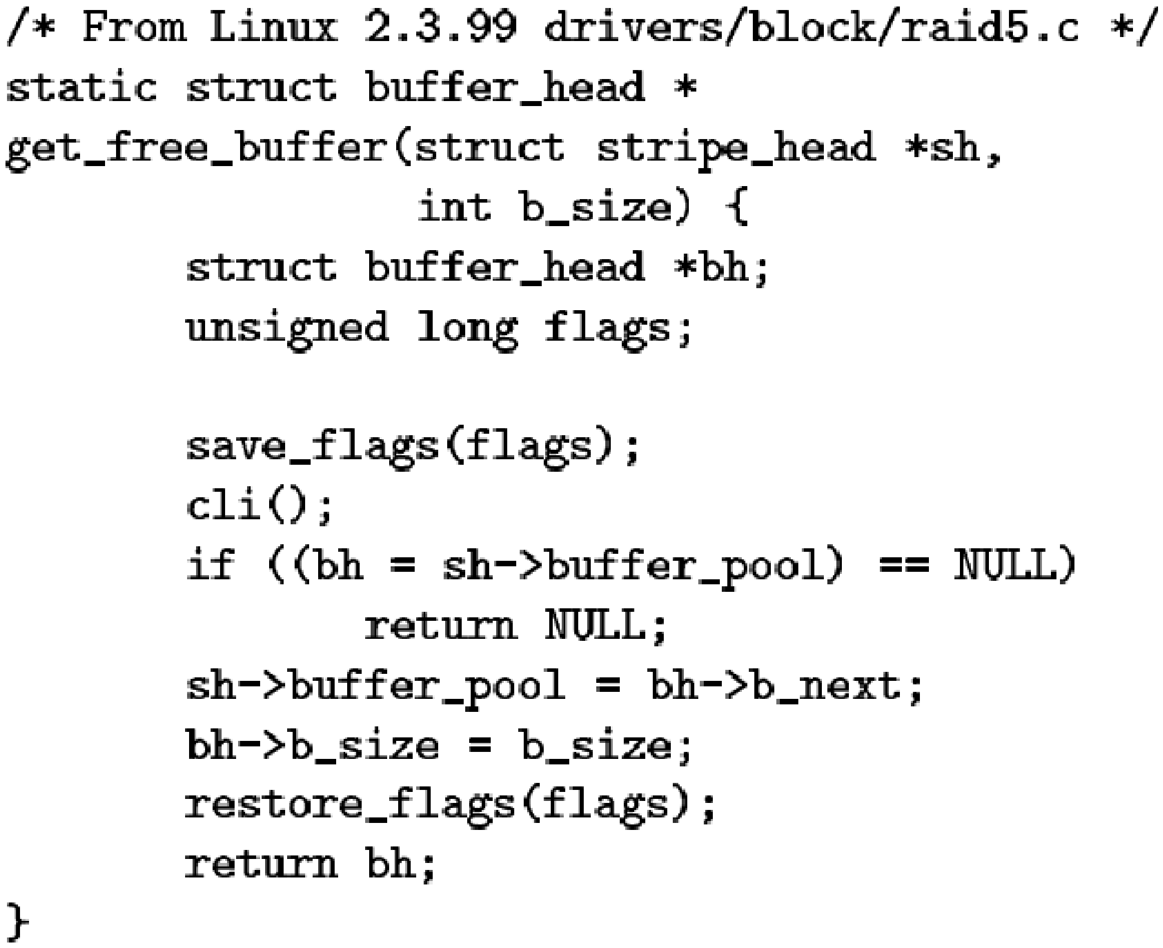
is_disabled

end path =>
err(end path
with/intr
disabled)

disable =>
err(double disable)

Applying the Analysis

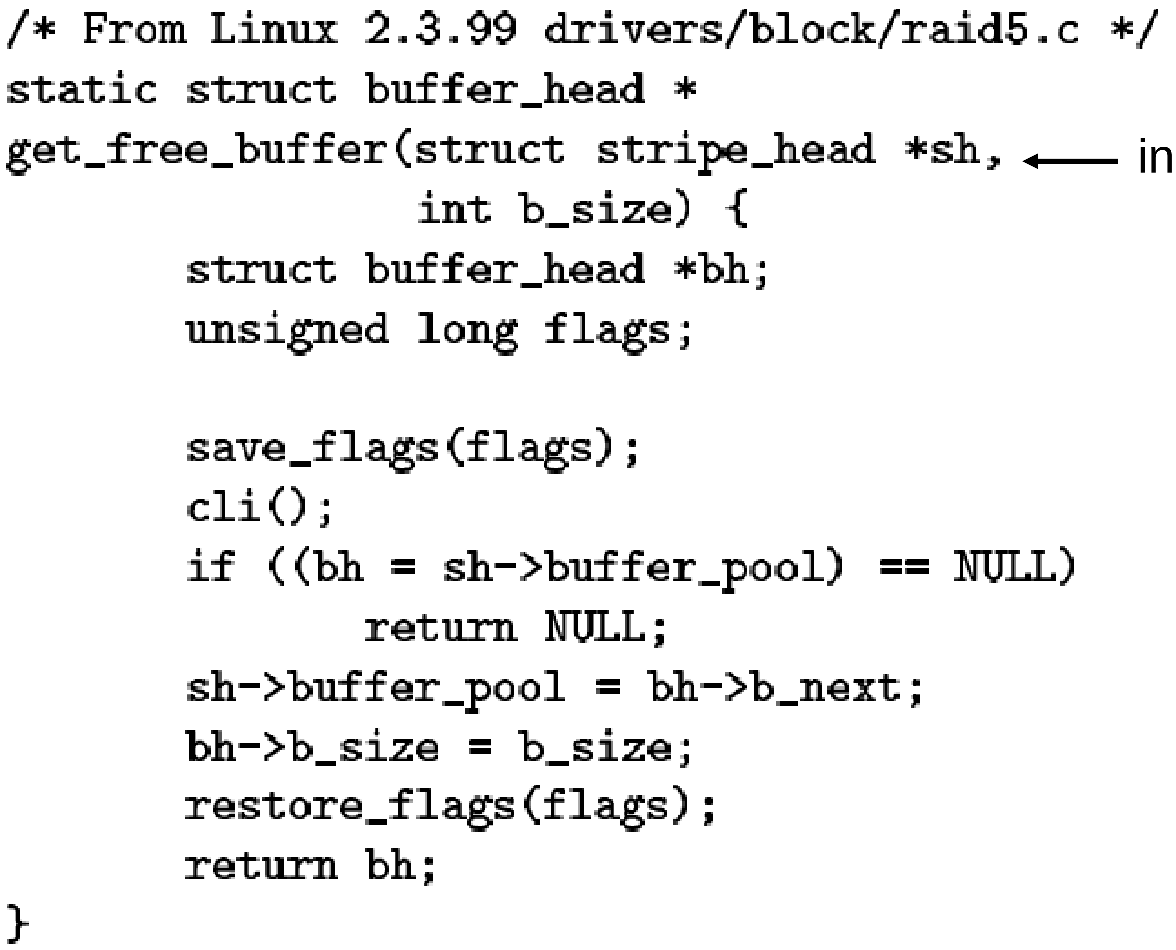
Source: Engler et al., *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI ’00.



```
/* 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.

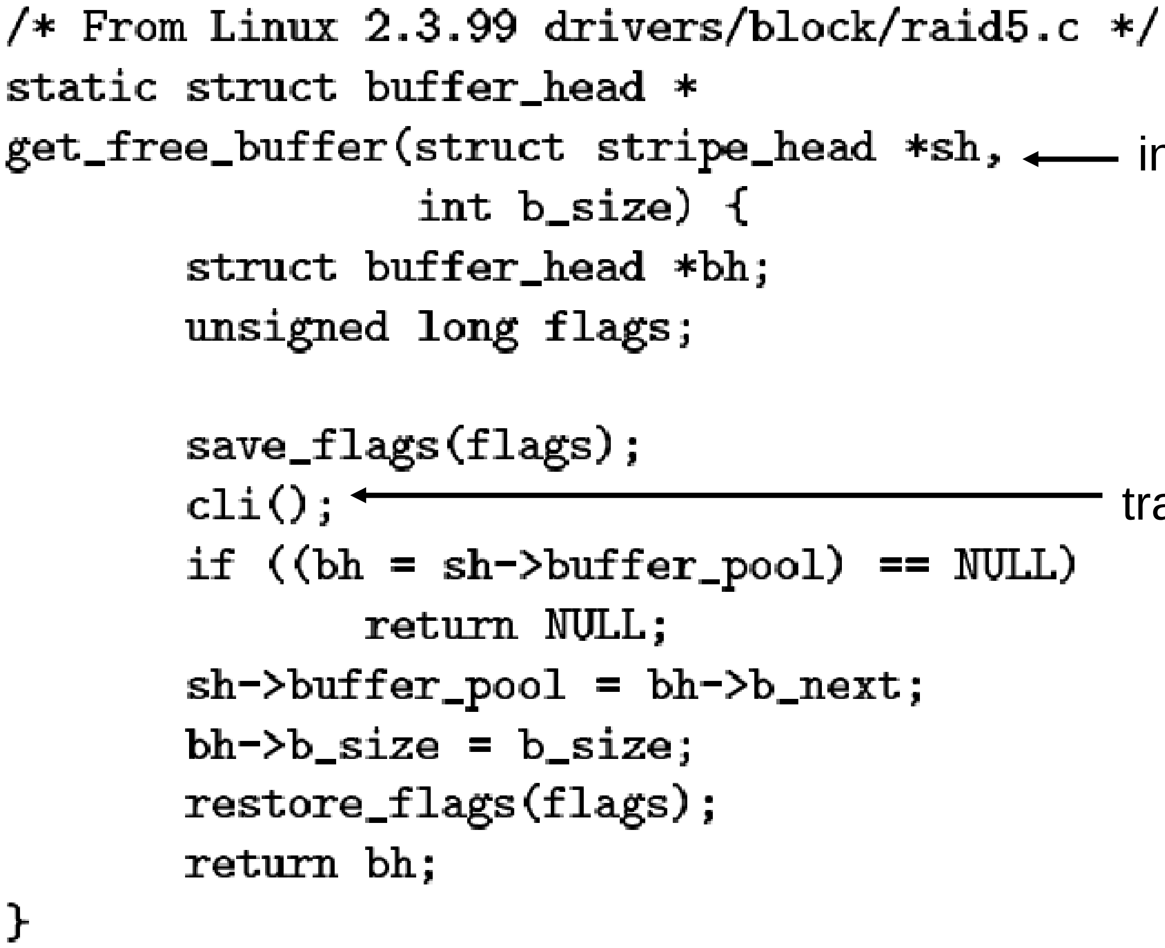


```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        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

Applying the Analysis

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



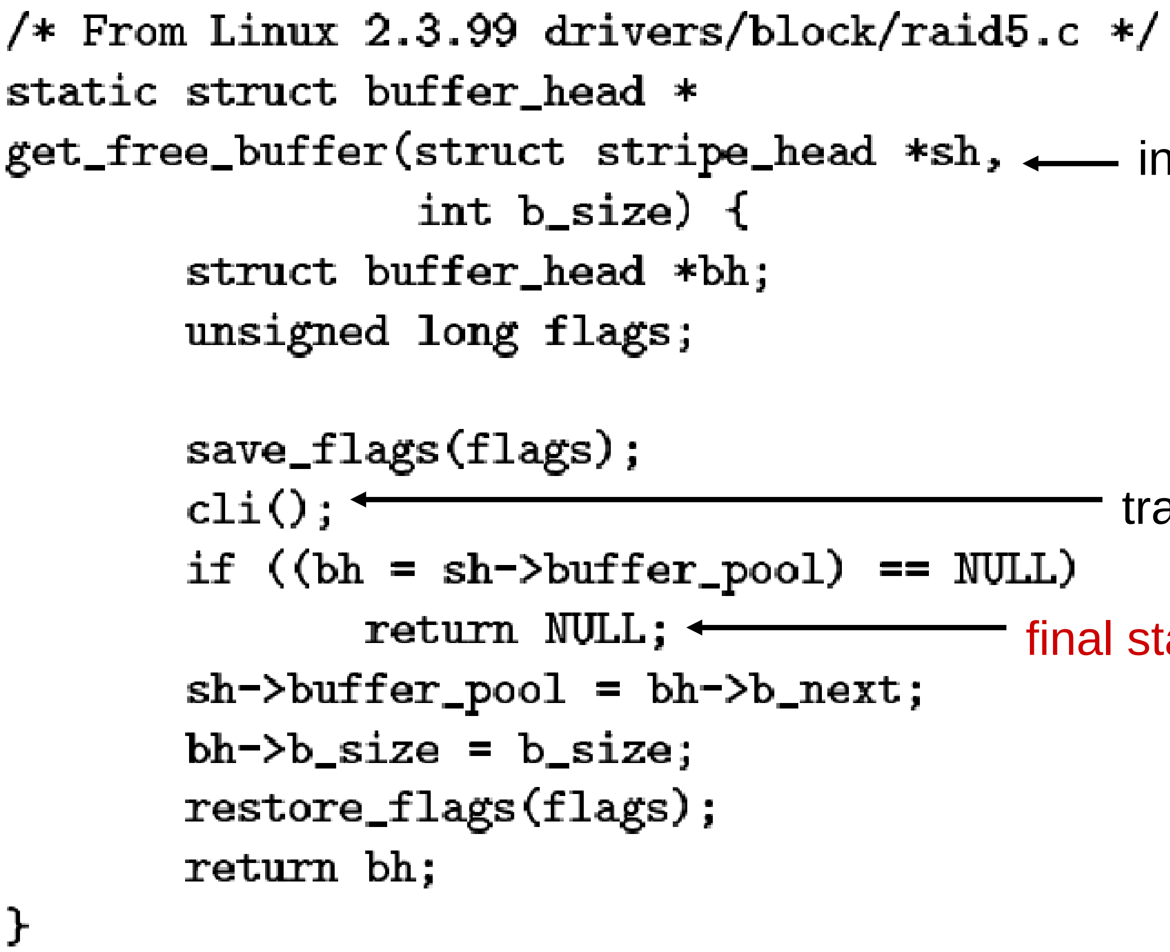
```
/* From Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head *sh,
                int b_size) {
        struct buffer_head *bh;
        unsigned long flags;

        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

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;
}
```

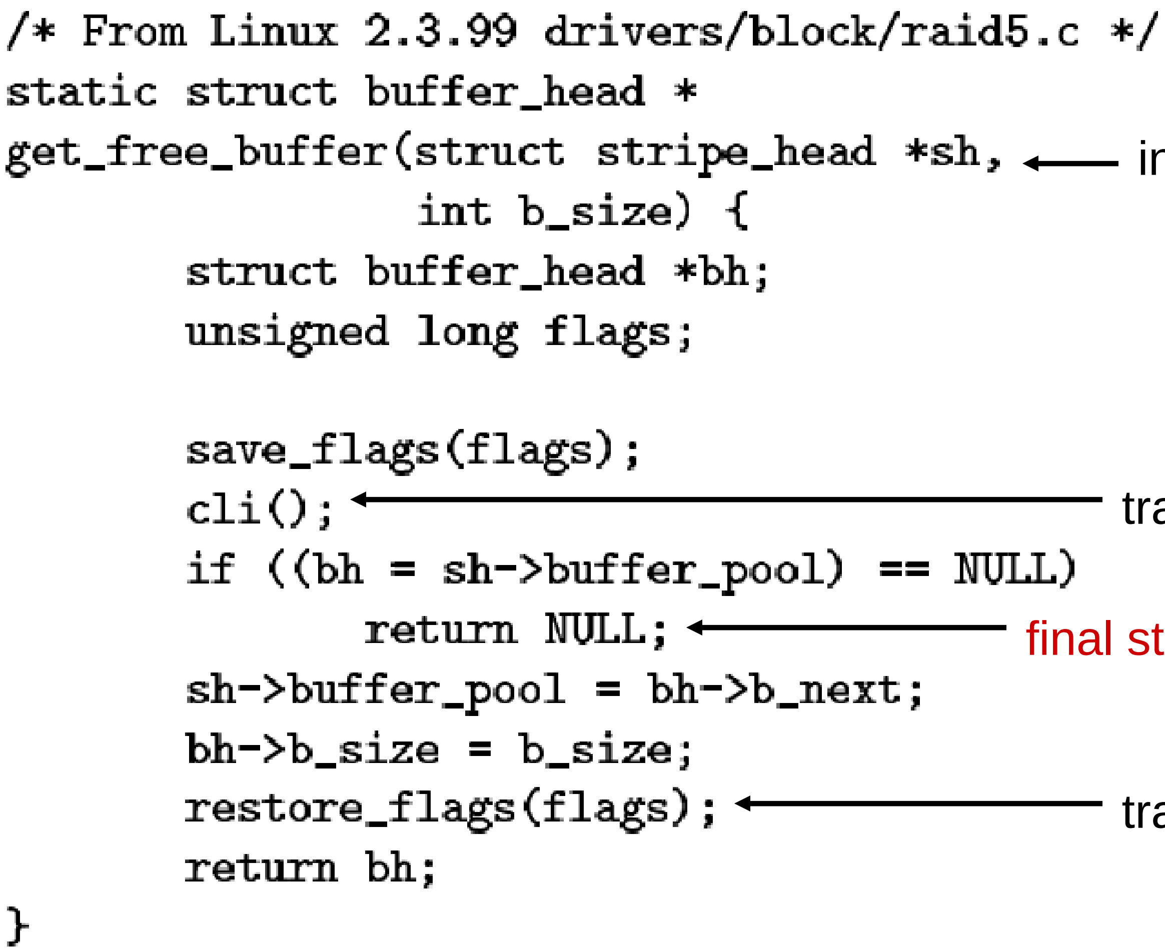
initial state is_enabled

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,
                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

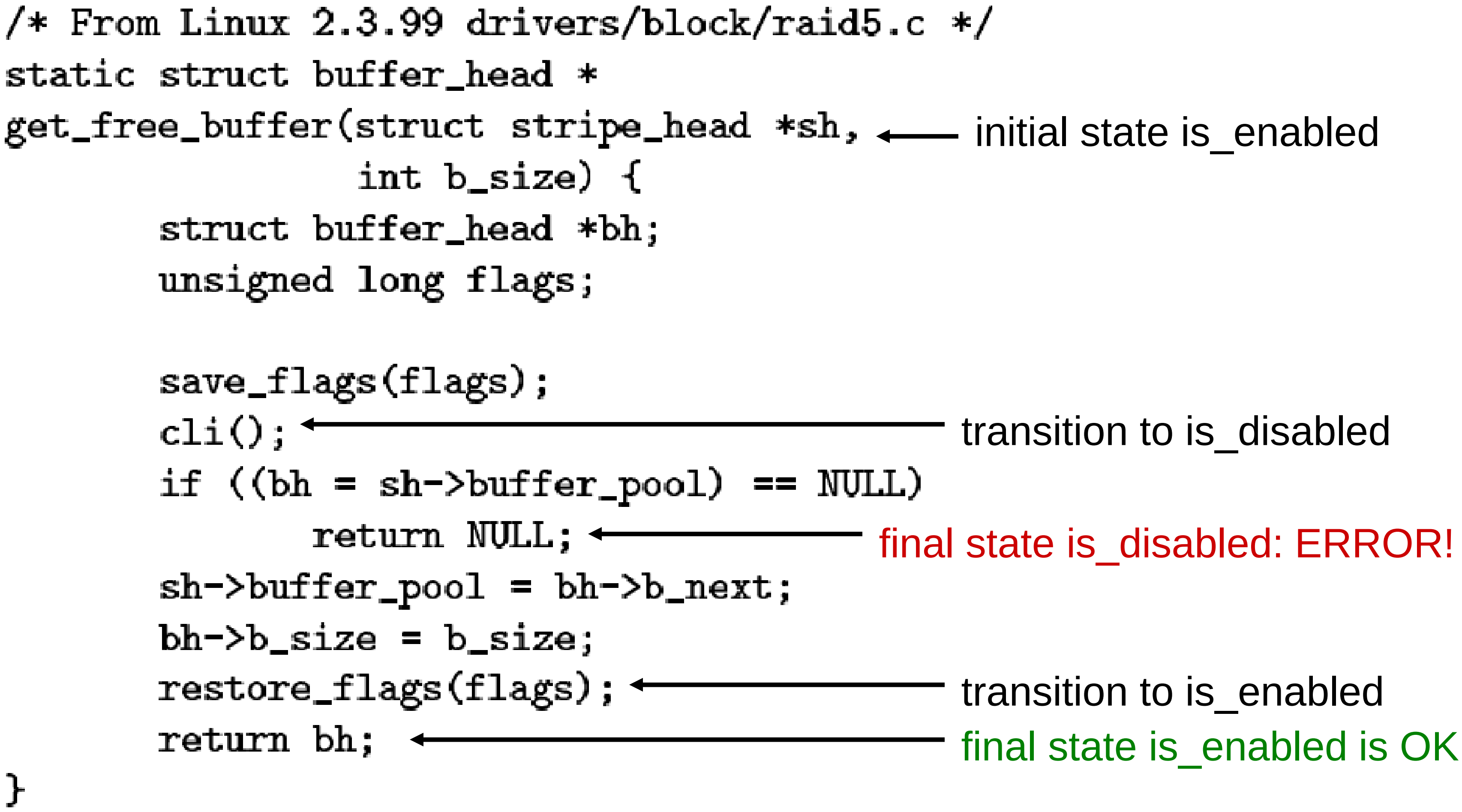
transition to is_disabled

final state is_disabled: ERROR!

transition to is_enabled

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

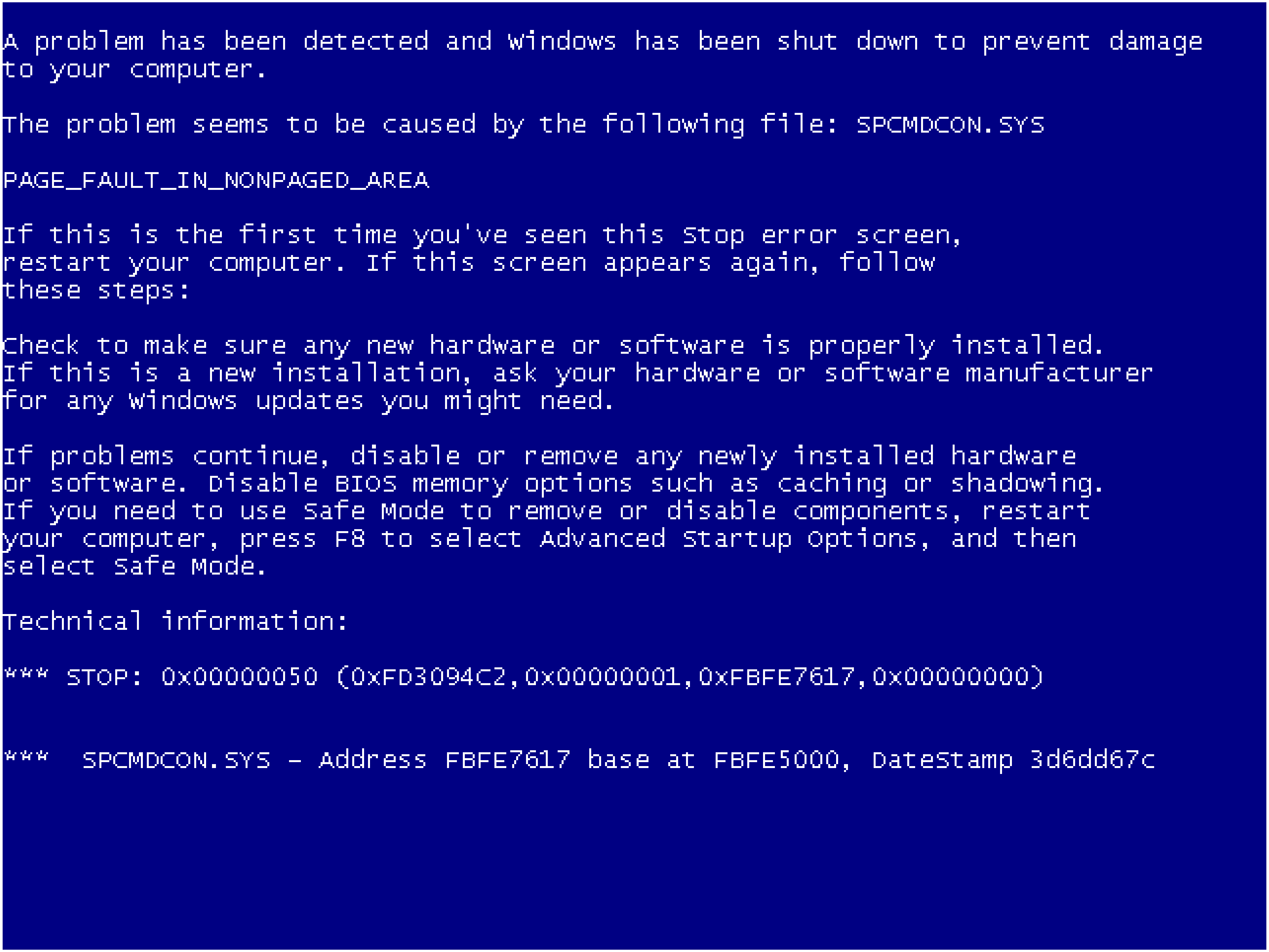
final state is_disabled: ERROR!

transition to is_enabled

final state is_enabled is OK

Outline

- Why program analysis?
  - The limits of testing and inspection
- What is program analysis?
- Course outline
- Representing programs
- AST-walking analyses



```
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)

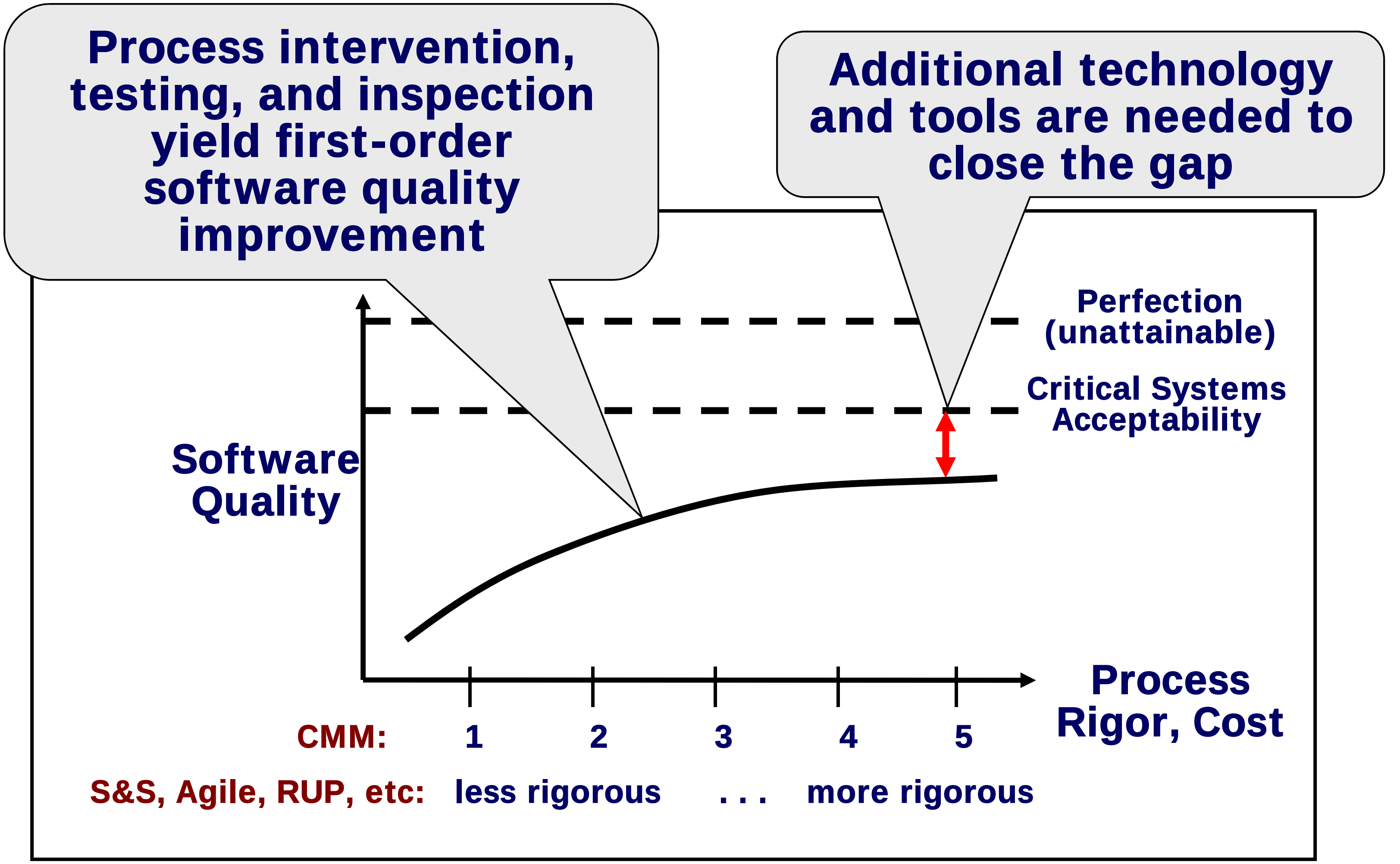
- Original process: manual code inspection
  - Effective when system and team are small
  - Too many paths to consider as system grew
- Early 1990s: add massive system and unit testing
  - Tests took weeks to run
    - Diversity of platforms and configurations
    - Sheer volume of tests
  - Inefficient 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 program analysis
  - More on this later

Process, Cost, and Quality

Slide: William Scherlis



Process intervention, testing, and inspection yield first-order software quality improvement

Additional technology and tools are needed to close the gap

Outline

- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Course outline
- Representing programs
- AST-walking analyses

Static Program Analysis Definition

- Static program analysis is the automated, systematic examination of an abstraction of a program’s state space

Static Program Analysis Definition

- Static program analysis is the automated, 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 Program Analysis Definition

- Static program analysis is the automated, 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

Static Program Analysis Definition

- Static program analysis is the automated, systematic examination of an abstraction of a program’s state space

- Mathematical properties (recurring theme)
  - Soundness
    - All reported results are true
      - **Verification: analysis says OK → correctness property holds**
      - Bug-finding: analysis says there is a bug → some (or all) real executions manifest that bug
  - Completeness
    - Everything that is true is reported
      - **Verification: the program is correct → the analysis will say so**
      - Bug-finding: some execution manifests a bug → the analysis will report it

Outline

- Why static analysis?
- What is static analysis?
    - Abstract state space exploration
- Course outline
- Representing programs
- AST-walking analyses

A Sample of Course Topics

- Dataflow analysis
- Abstract interpretation
- Interprocedural analysis
- Pointer analysis
- Symbolic execution
- Dynamic analysis
- Verification

Course Syllabus

- See web page

Outline

- Why static analysis?
- What is static analysis?
  - Abstract state space exploration
- Course outline
- Representing programs
- AST-walking analyses

Representing Programs: The WHILE Language

- A simple procedural language with:
  - assignment
  - statement sequencing
  - conditionals
  - while loops
- Used in early papers (e.g. Hoare 1969) as a “sandbox” for thinking about program semantics
- We will use it to illustrate several different kinds of analysis

WHILE Syntax

- Categories of syntax
  - $S \in$ **Stmt** statements
  - $a \in$ **AExp** arithmetic expressions
  - $x,y \in$ **Var** variables
  - $n \in$ **Num** number literals
  - $P \in$ **BExp** boolean expressions
- Syntax
  - $S ::= x := a \mid \text{skip} \mid S_1 ;\, S_2$
    $\mid\ \text{if } P \text{ then } S_1 \text{ else } S_2 \mid \text{while } P \text{ do } S$
  - $a ::= x \mid n \mid a_1\ op_a\ a_2$
  - $op_a ::= + \mid - \mid * \mid / \mid \ldots$
  - $P ::= \text{true} \mid \text{false} \mid \text{not } P \mid P_1\ op_b\ P_2 \mid a_1\ op_r\ a_2$
  - $op_b ::= \text{and} \mid \text{or} \mid \ldots$
  - $op_r ::= < \mid \leq \mid = \mid > \mid \geq \mid \ldots$

Example WHILE Program

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

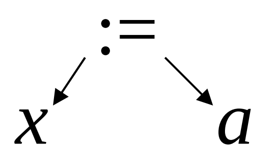
Computes the factorial function, with the input in x and the output in z

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

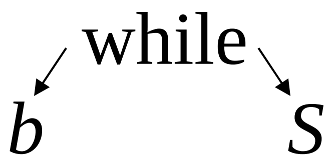
- A tree representation of source code
- Based on the language grammar
  - One type of node for each production
  - $S ::= x := a$ ➔



:=

x a

  - $S ::=$ while b do S ➔



while

b 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?

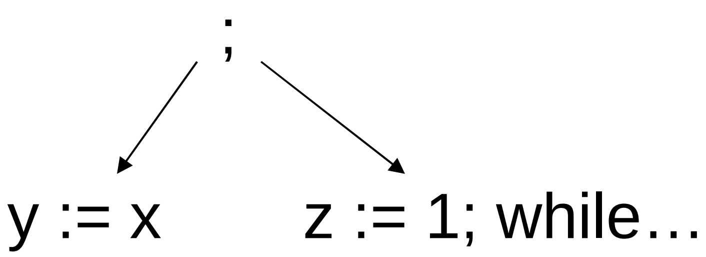
Parsing Example

```
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

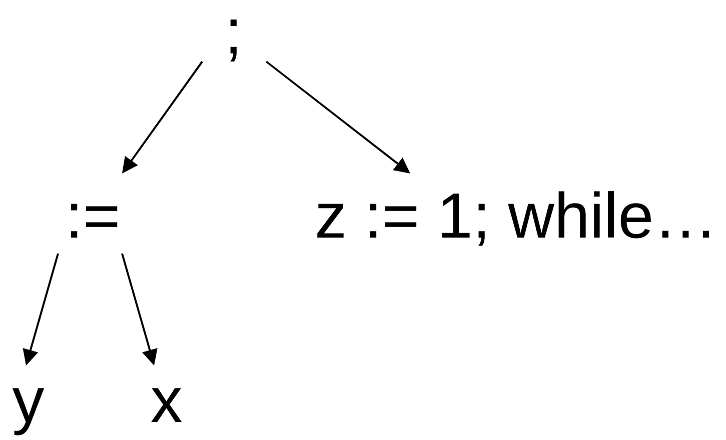
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- 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
```



;

:=

z := 1; while…

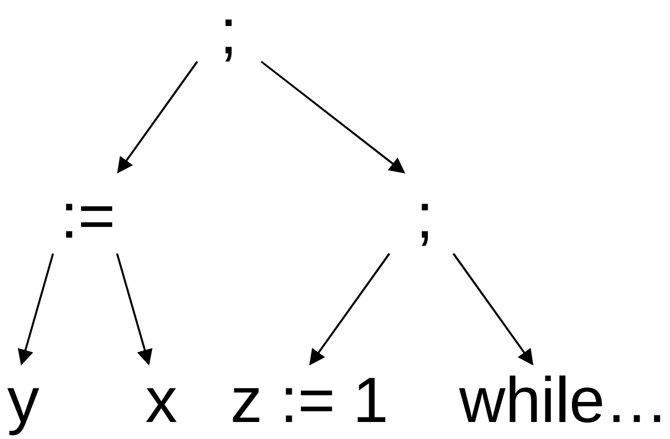
y

x

- 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
```



;

:=

;

y

x

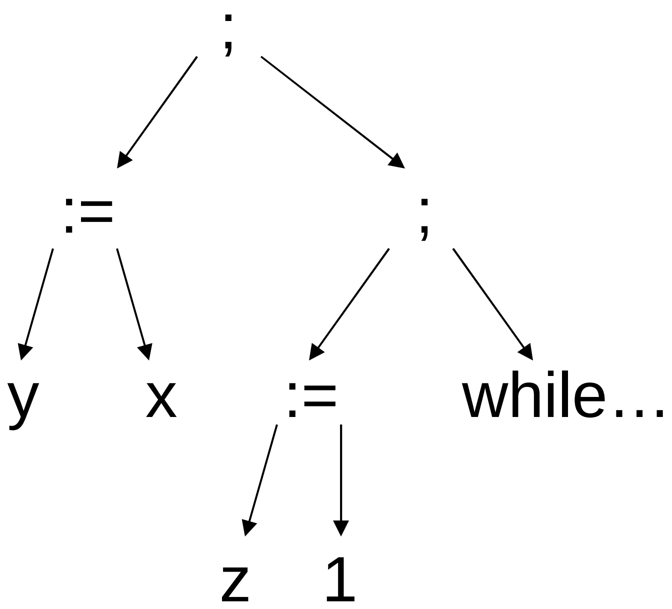
z := 1

while…

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

Parsing Example

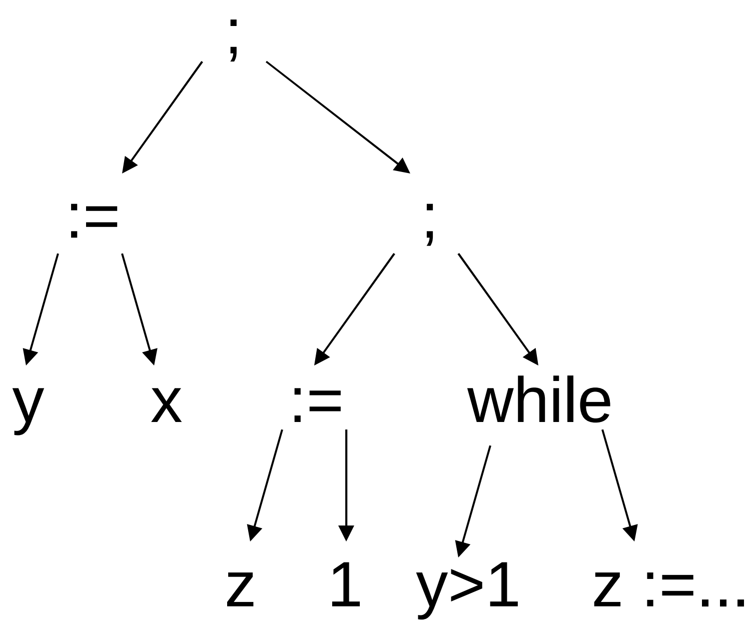
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



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

Parsing Example

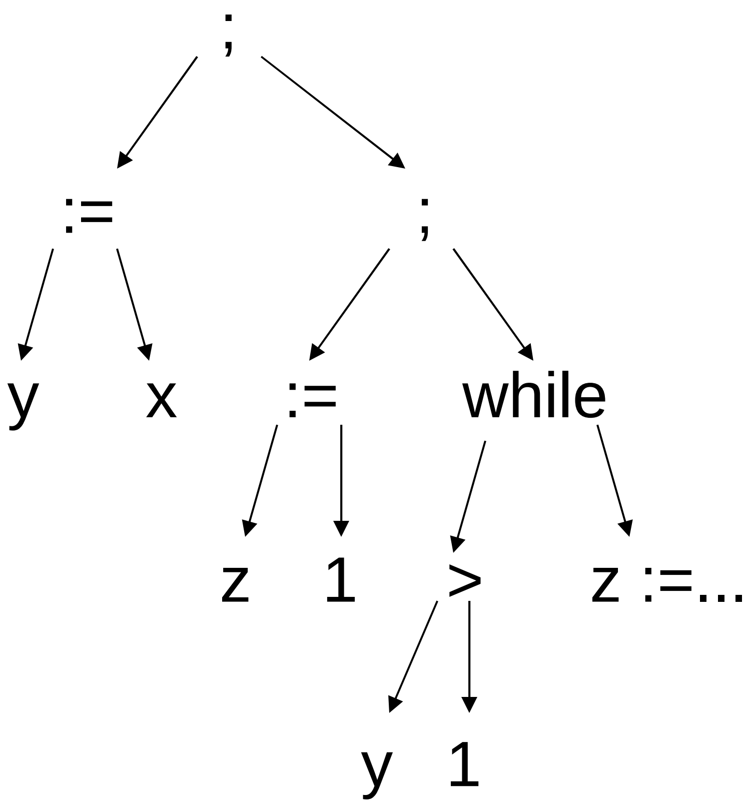
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



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

Parsing Example

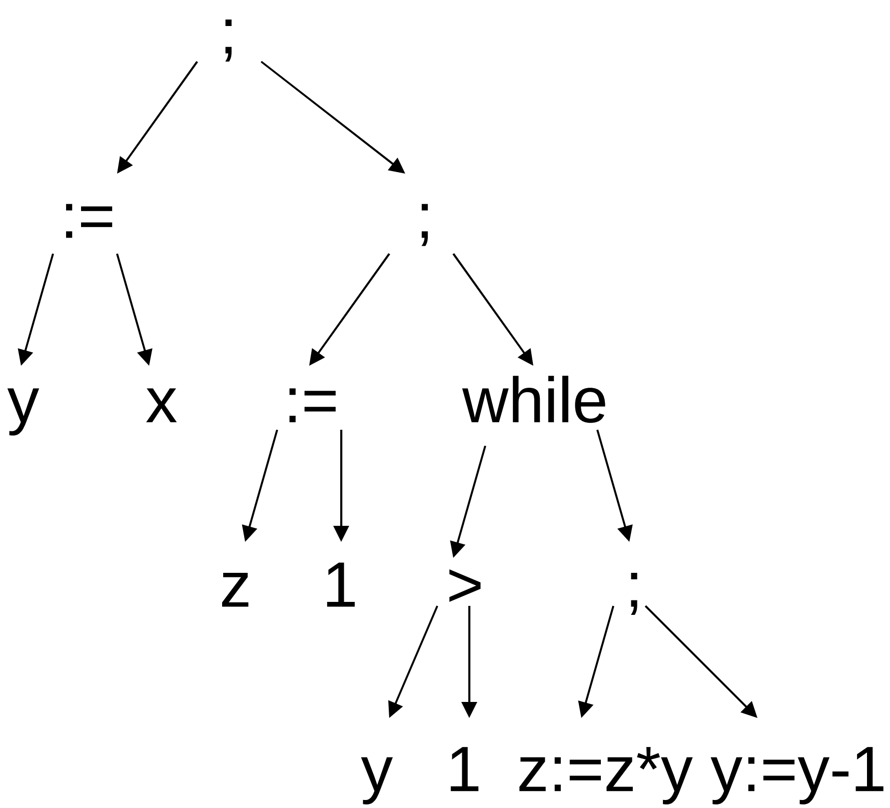
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- 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
```



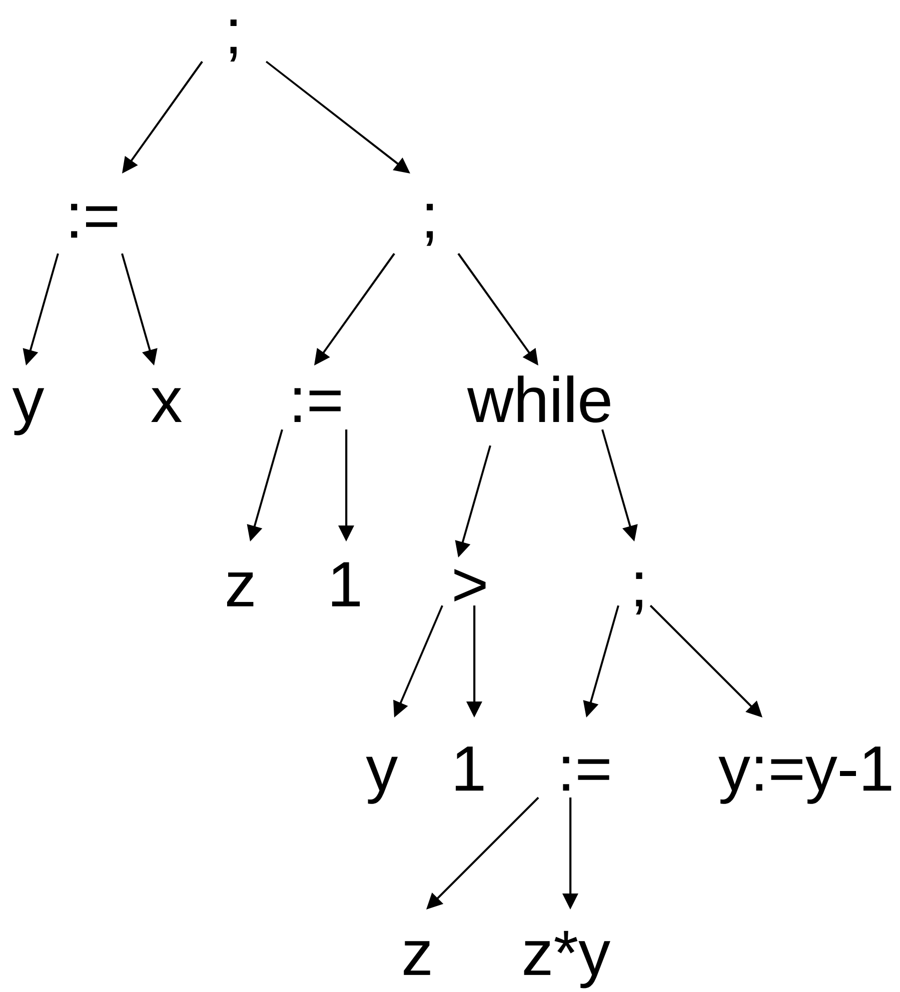
;
:=
;
y
x
:=
while
z
1
>
;
y
1
z:=z*y
y:=y-1

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

Parsing Example

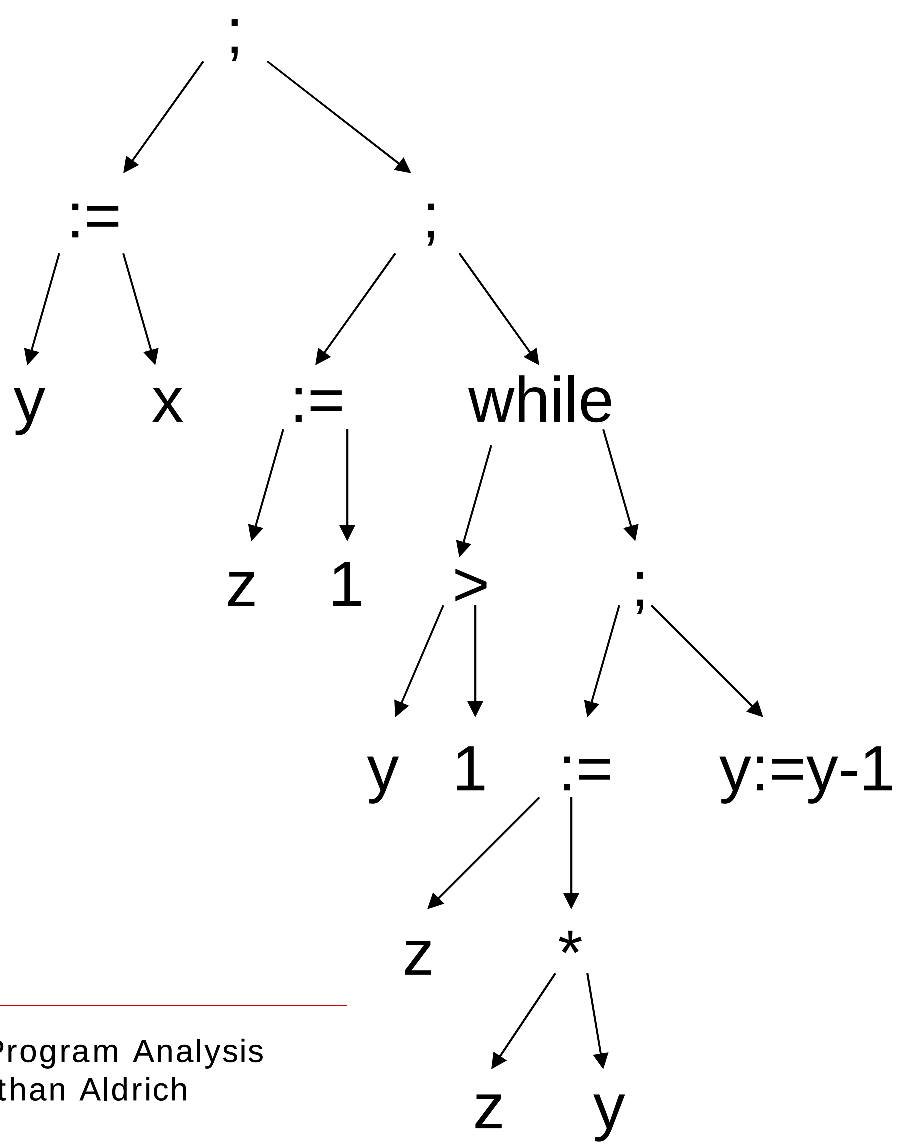
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

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



Parsing Example

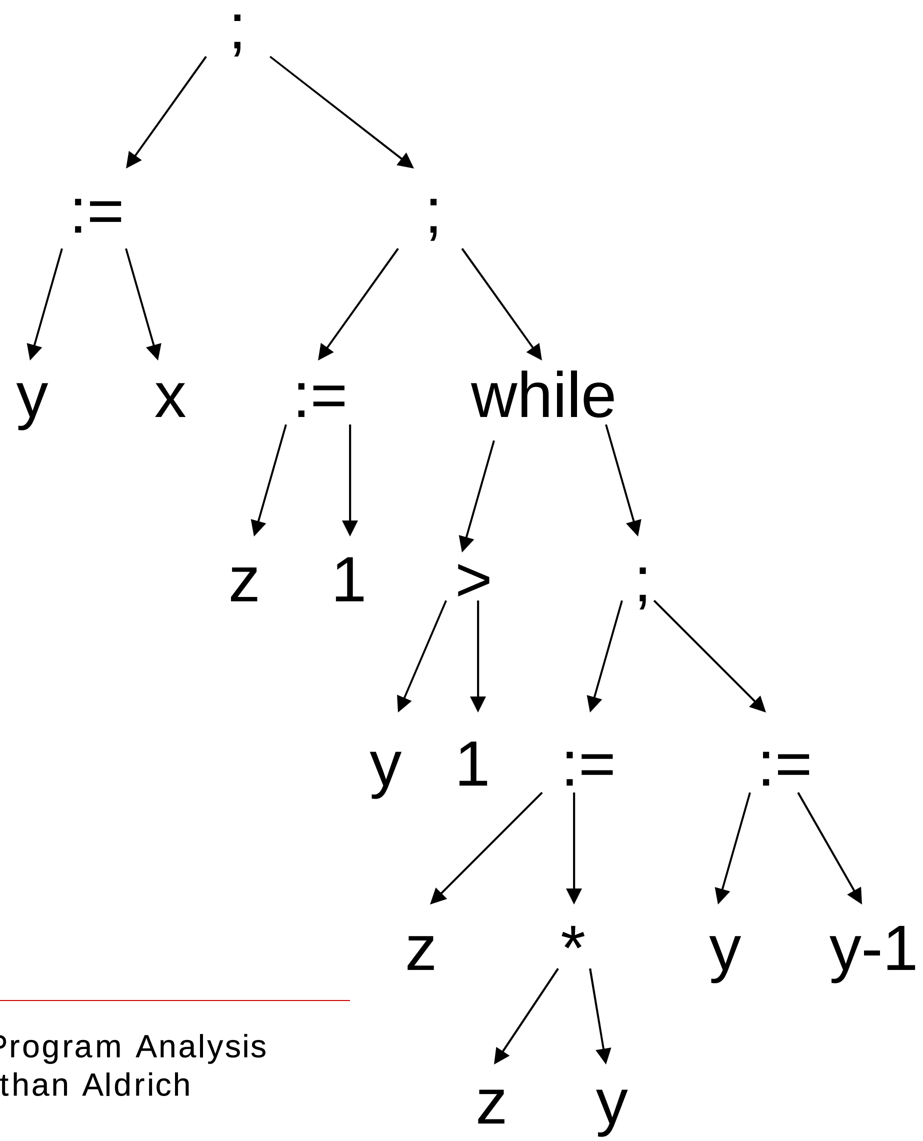
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



- 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
```

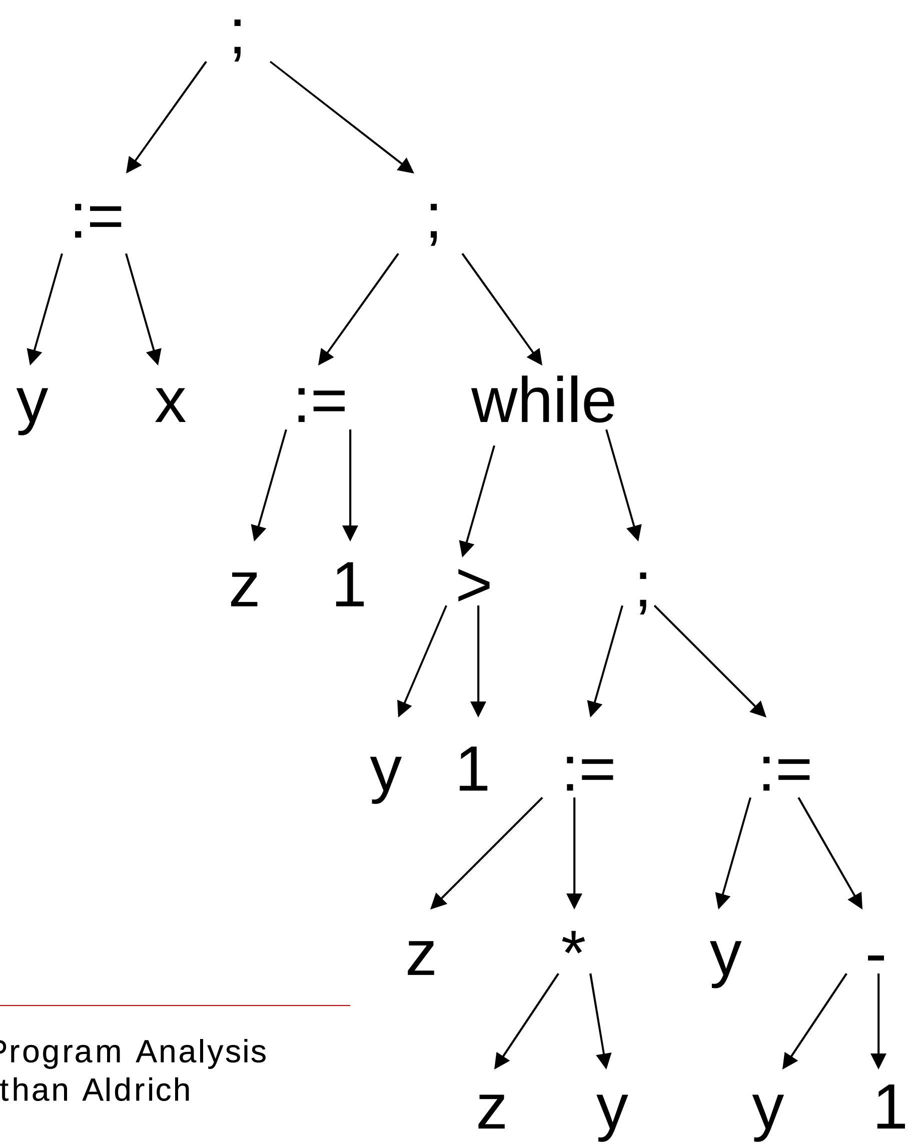


- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

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 …



Exercise

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
- $S \quad ::= x := a$

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

WHILE ASTs in Java

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

  $\quad\quad | \text{ skip}$

```
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
  | skip
  | S_1; S_2
  | if b then S_1 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

  | skip

  | S_1; S_2

  | if b then S_1 else S_2

  | 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
- Course Outline
- Representing programs
- AST-walking analyses

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

Example: Shifting by more than 31 bits

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

        super.visitShiftExpression(e);
    }
```

Practice: String concatenation in a loop

- Write pseudocode for a simple AST-walker analysis that warns when string concatenation occurs in a loop
  - In Java and .NET it is more efficient to use a StringBuffer
  - Assume any appropriate AST elements

To get you started:

```
class StringConcatLoopAnalysis extends Visitor {
        void visitStringConcat(StringConcat e) {



        }



}
```

Practice: String concatenation in a loop

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

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

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

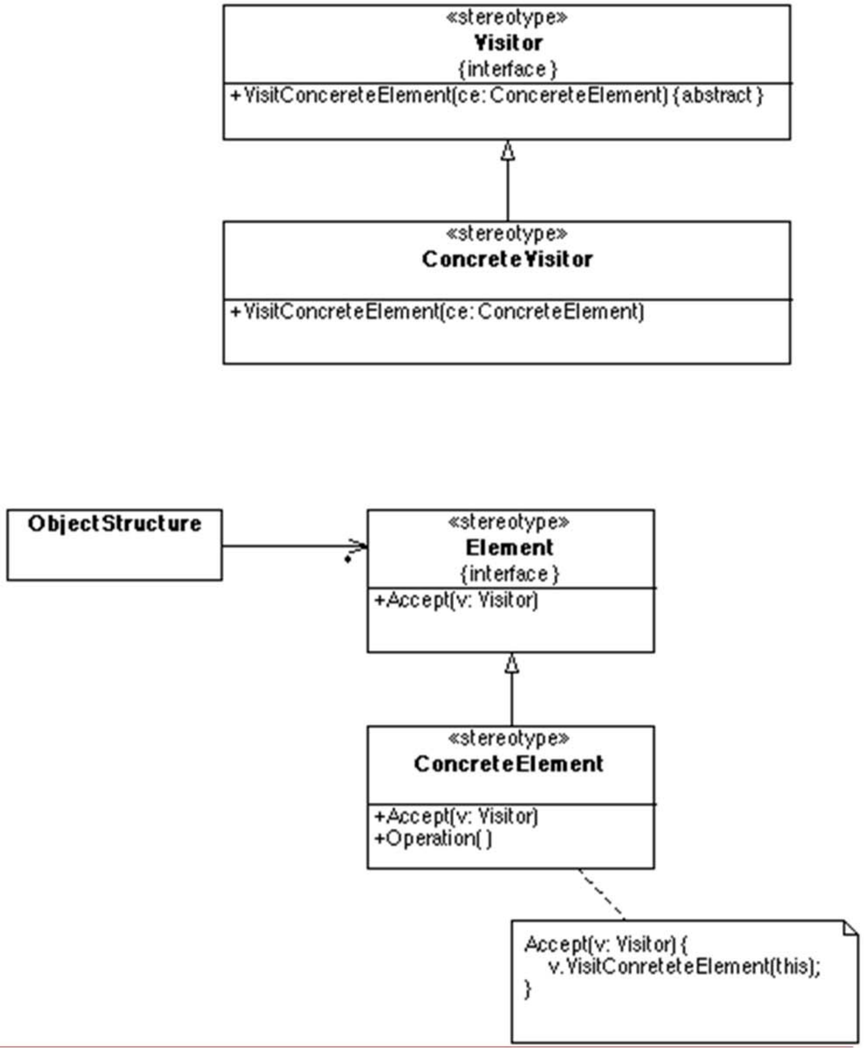
Bonus slides

- We did not get to these in class

The Visitor Design Pattern

*often used in analysis tools written in an OO language

- 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 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