# 17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

# QUALITY ASSURANCE: SAMPLING

CHRISTIAN KAESTNER

CARNEGIE MELLON UNIVERSITY

INSTITUTE FOR SOFTWARE RESEARCH

# READING ASSIGNMENT NOV 16

**Textbook Chapter 9**

**Nhlabatsi, Armstrong, Robin Laney, and Bashar Nuseibeh. "Feature interaction: the security threat from within software systems."** ***Progress in Informatics*** **5 (2008): 75-89.**

# LEARNING GOALS

**Understand the challenge of feature interactions**

**Understand the challenges from requirements level down to implementations**

**Implementation vs specification and emergent behavior**

**Apply mitigation strategies**

Printer Firmware

## Checking **Products**

2000 Features

100 Printers

30 New Printers per Year





# Checking **Products**



10000 Features

? Products

Linux
Kernel







# Checking **Product Line**

Implementation with 10000 Features

#ifdef, Frameworks, FOP, AOP, …

Linux Kernel







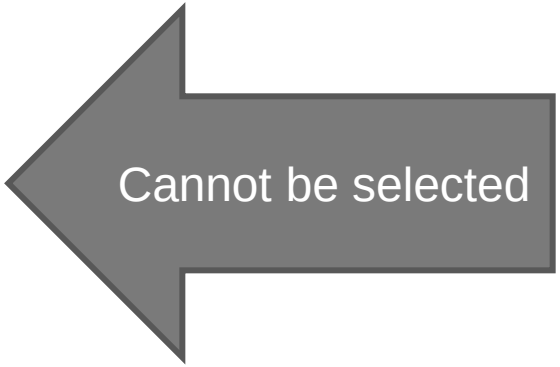
# DEAD IFDEF CODE DETECTION: UNDERTAKER

# PRESENCE CONDITIONS

```
line 1                  true
#ifdef A
line 2                  A
    #ifndef B
    line 3              A ∧ ¬ B
    #endif
line 4                  A
#elif defined(X)
line 5                  ¬ A ∧ X
#else
line 6                  ¬ A ∧ ¬ X
#endif
```

# DEAD CODE

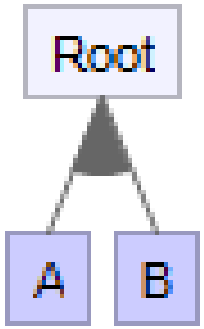
```
line 1                  true
#ifdef A                A
line 2                  A
    #ifndef A
    line 3              A ∧ ¬ A
    #endif              A
line 4                  ¬ A ∧ X
#elif defined(X)        ¬ A ∧ ¬ X
line 5
#else
line 6
#endif
```




Analysis:
SAT(PC(Block i))

Include feature model in reasoning!

# DEAD CODE




(A ∨ B) ∧ ¬(A ∧ B)

| | |
|---|---|
| line 1 | true |
| #ifdef A | |
| line 2 | A |
| #ifdef B | |
| line 3 | A ∧ B |
| #endif | |
| line 4 | A |
| #elif defined(X) | |
| line 5 | ¬A ∧ X |
| #else | |

Cannot be selected

Analysis:
SAT(FM ∧ PC(Block i))

Analyze feature model and implementation

# OTHER QUESTIONS

**Which files are never compiled?**

**Which feature modules cannot be included?**

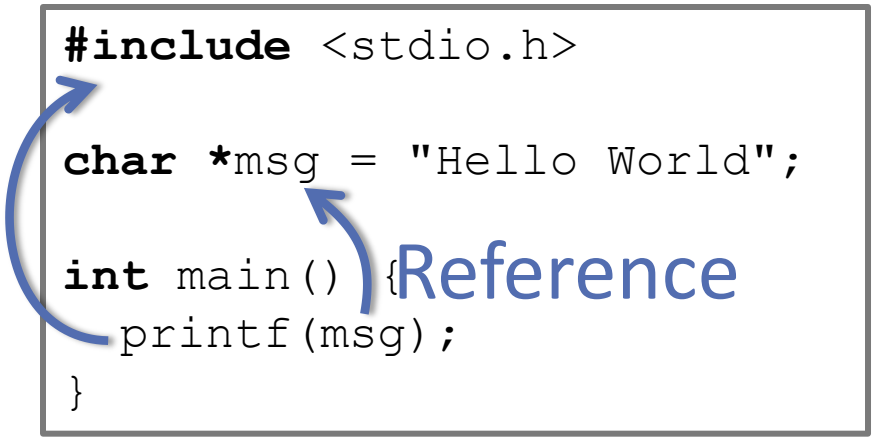
**Which plugins/packages always have conflicts?**

**Which features never influence the source code?**

**Which code fragments seem optional but are not?**

# FAMILY-BASED ANALYSIS

# TYPE CHECKING




**Type errors: referenced variable does not exist, …**

# VARIABILITY-AWARE TYPE CHECKING

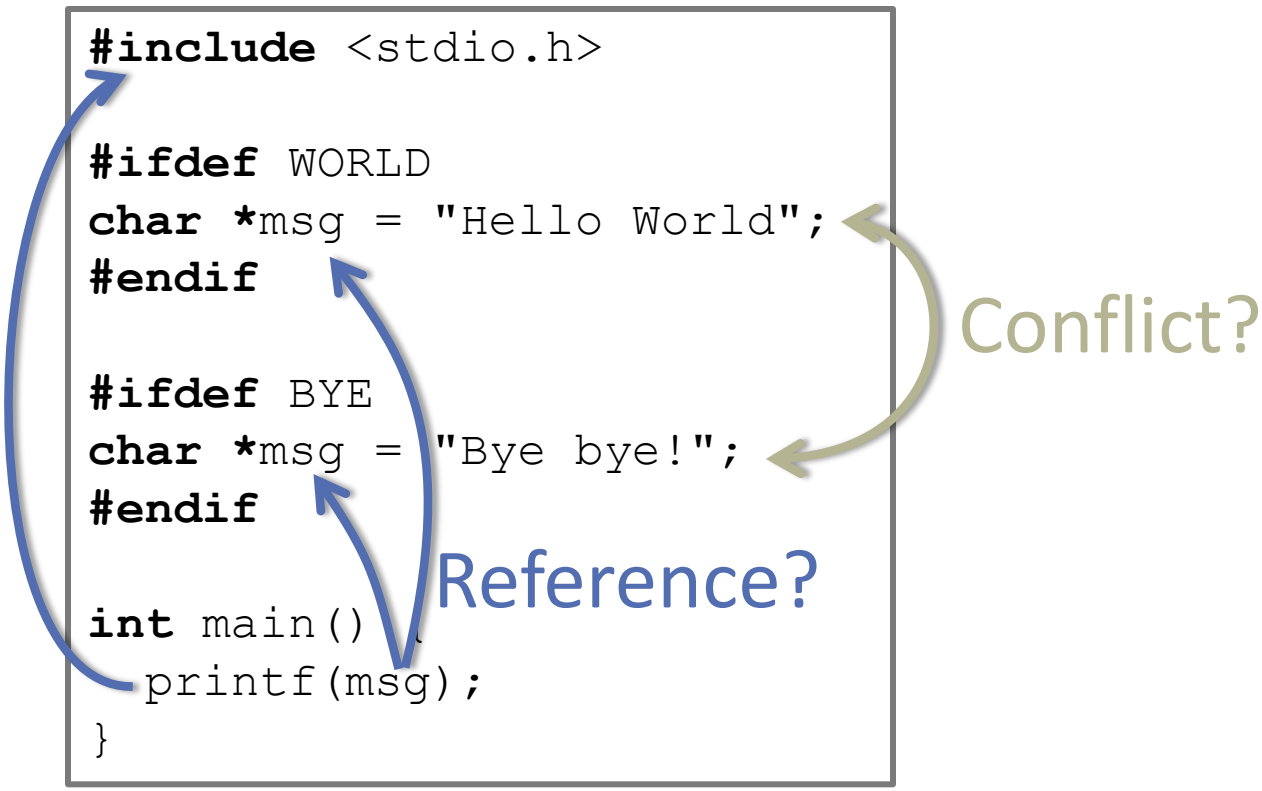
```
#include <stdio.h>

#ifdef WORLD
char *msg = "Hello World";
#endif

#ifdef BYE
char *msg = "Bye bye!";
#endif

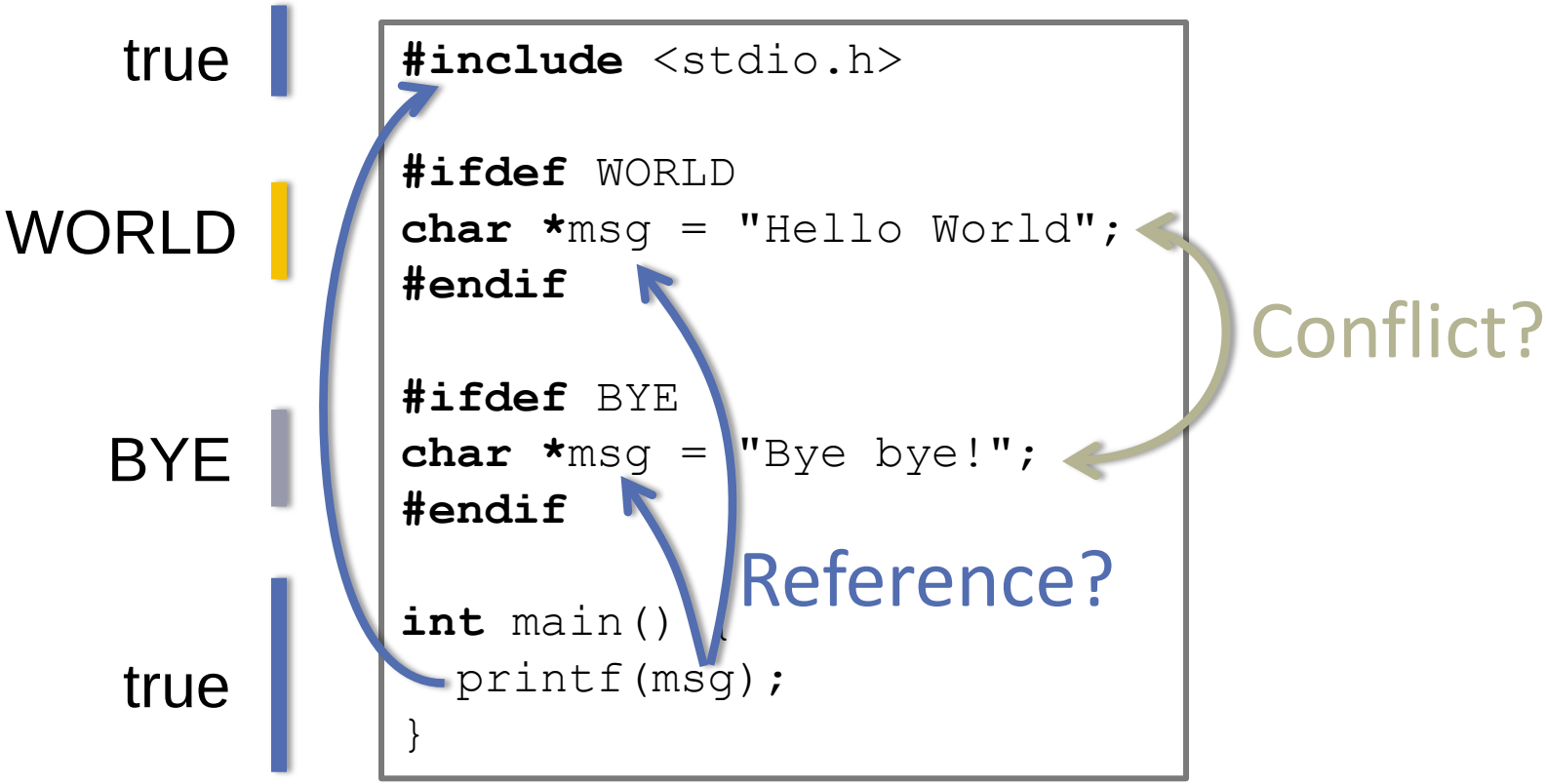
int main() {
  printf(msg);
}
```

# VARIABILITY-AWARE TYPE CHECKING

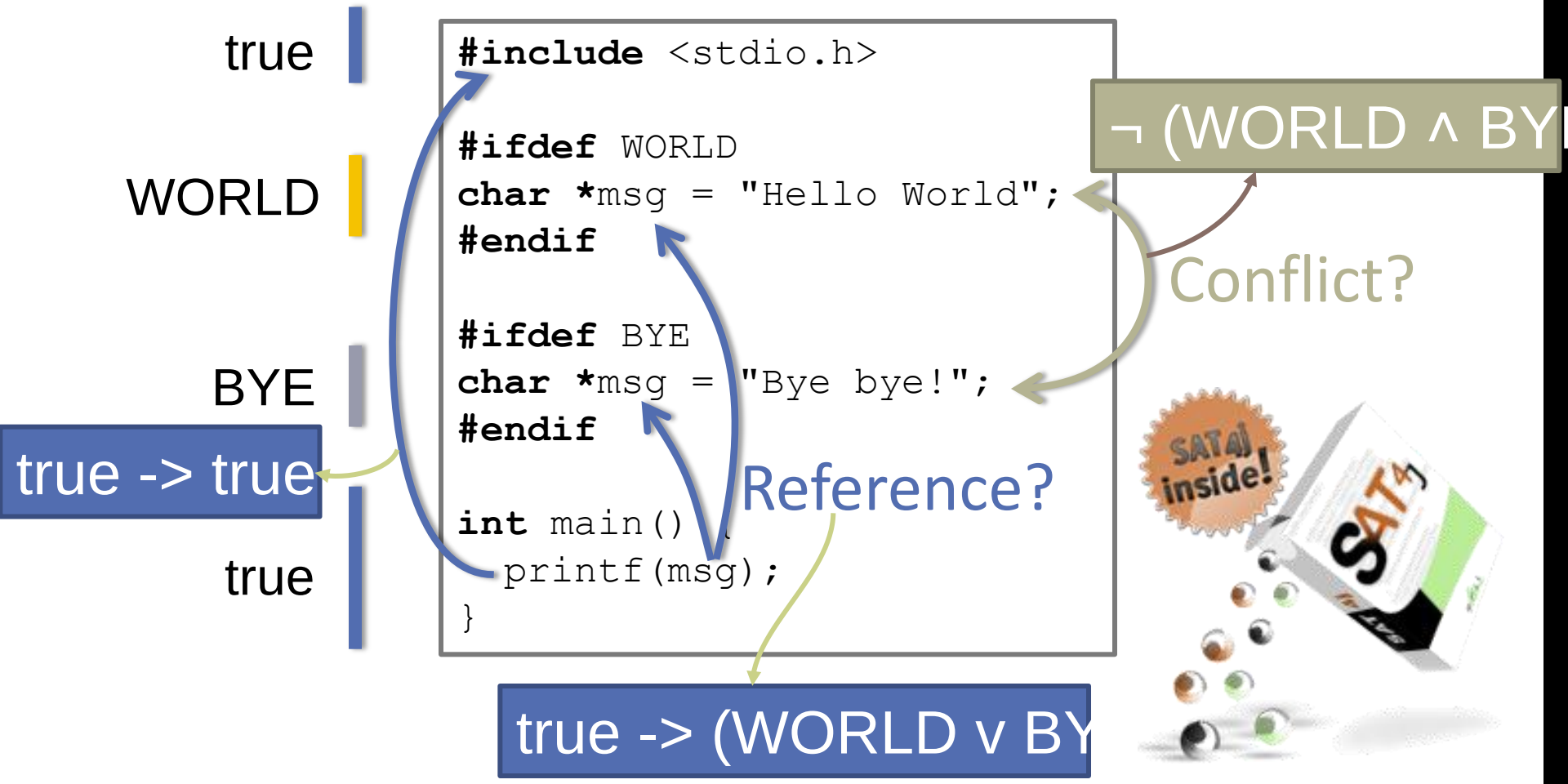


# VARIABILITY-AWARE TYPE CHECKING

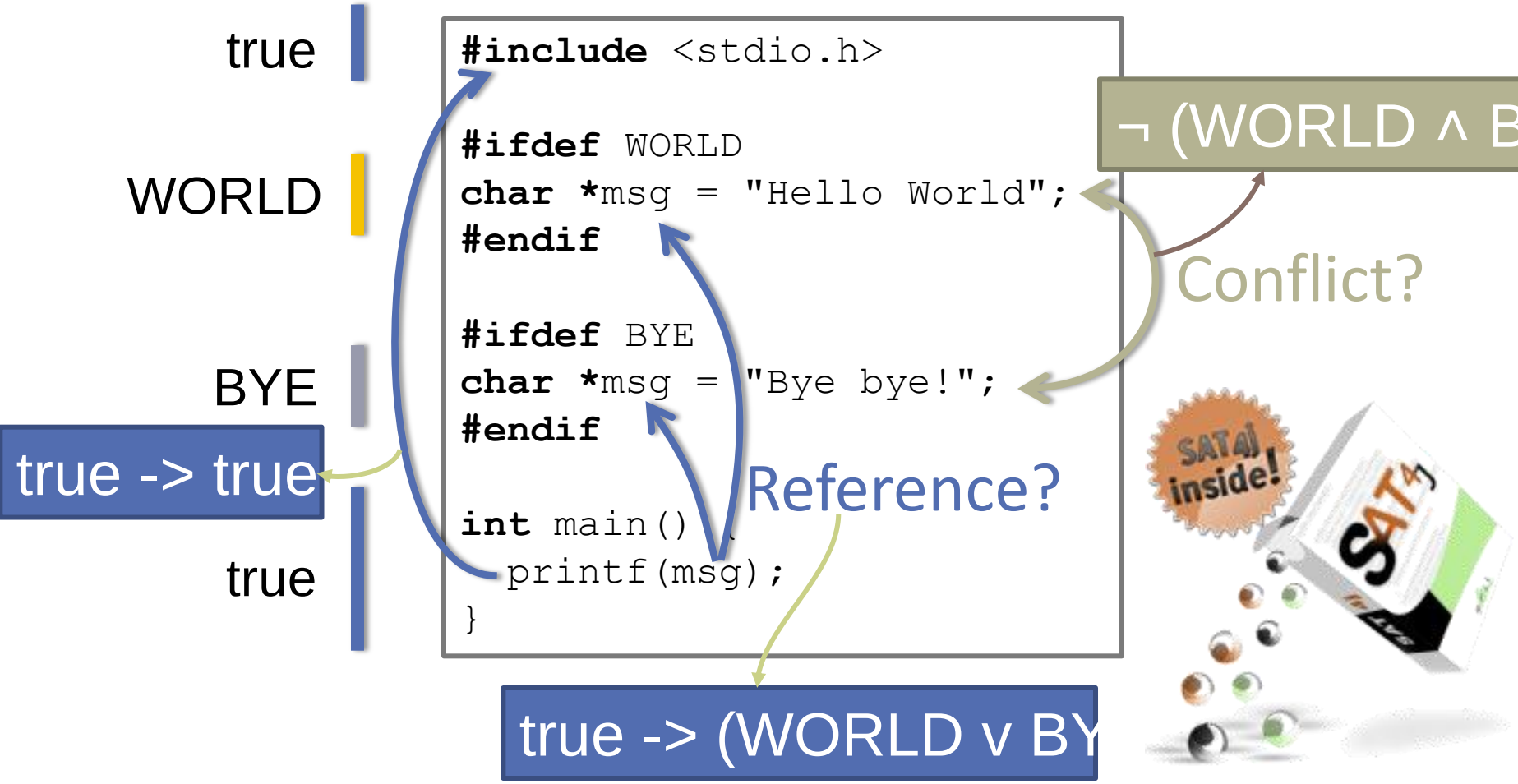
Presence conditions:



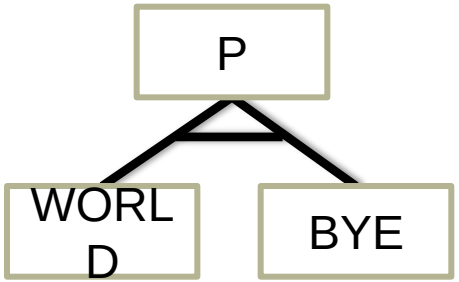
# VARIABILITY-AWARE TYPE CHECKING



# REACHABILITY: PC(SOURCE) -> PC(TARGET)
# CONFLICTS: ¬(PC(DEF1) ∧ PC(DEF2))



# INCLUDING THE FEATURE MODEL




true

WORLD

BYE

FM -> (true -> true)

true

```
#include <stdio.h>

#ifdef WORLD
char *msg = "Hello World";
#endif

#ifdef BYE
char *msg = "Bye bye!";
#endif

int main() {
  printf(msg);
}
```

FM -> ¬ (WORLD ∧ BYE)

Conflict?

Reference?

FM -> (true -> (WORLD v BYE))

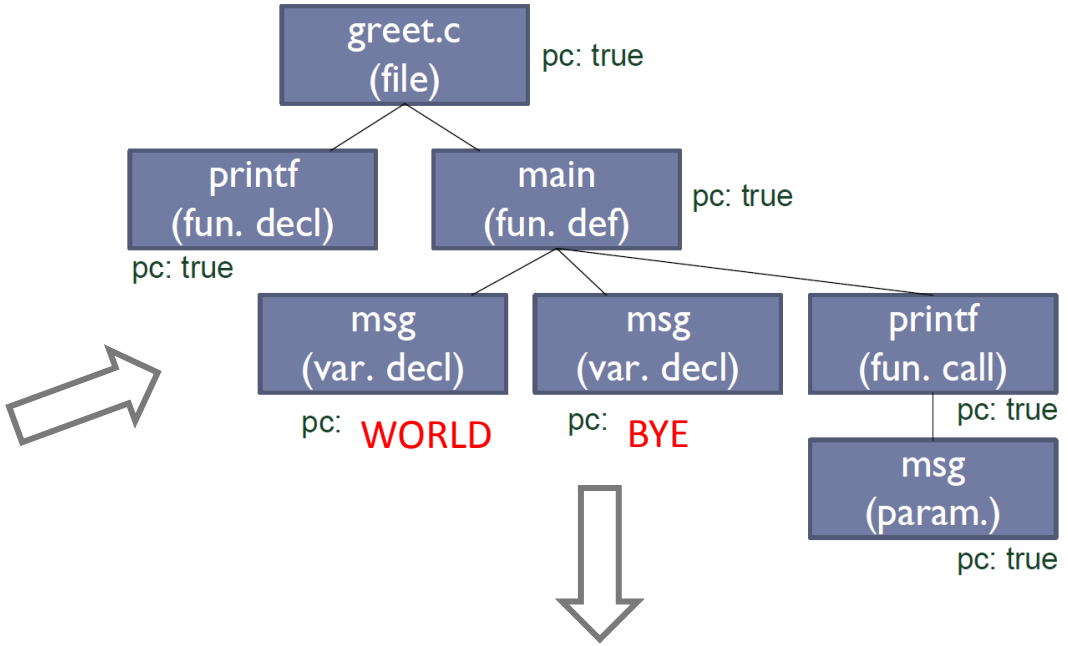


# AST with Variability Information

```
#include <stdio.h>

#ifdef WORLD
char * msg = "Hello World\n";
#endif
#ifdef BYE
char * msg = "Bye bye!\n";
#endif

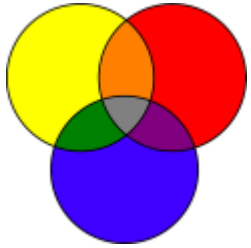
main() {
  printf(msg);
}
```



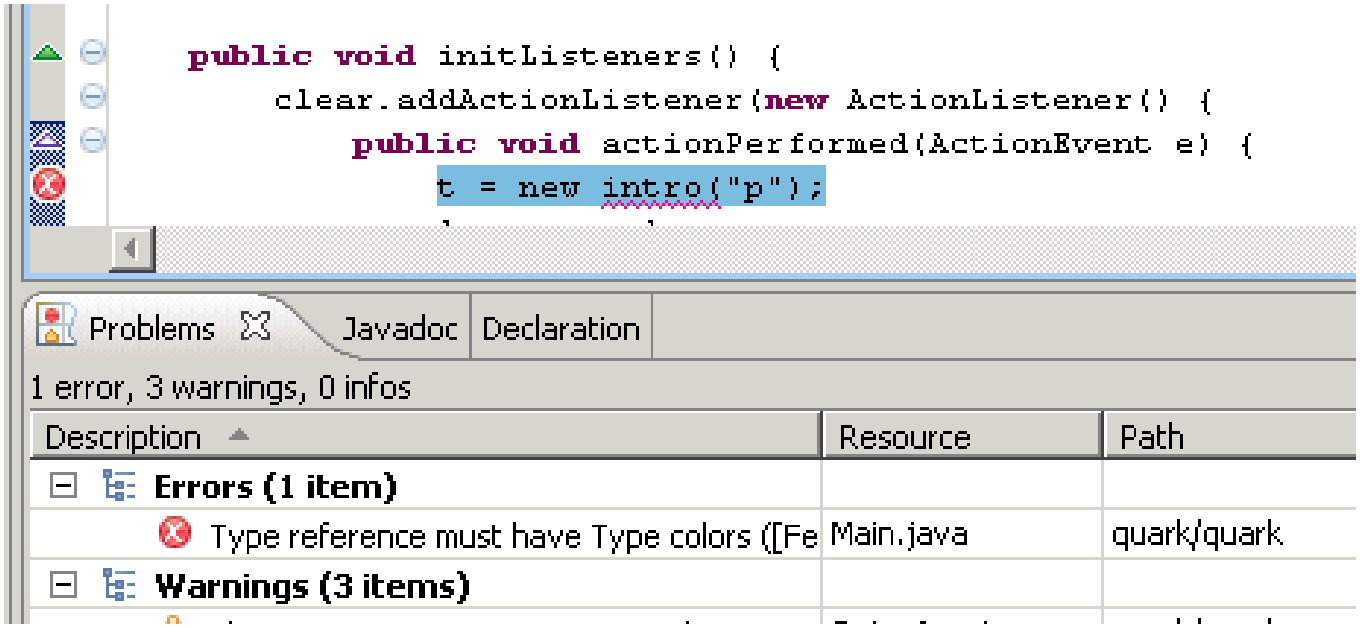

| Name | Type | Scope | Presence Condition |
|---|---|---|---|
| printf | char * → int | 0 | true |
| msg | char * | 0 | WORLD |
| msg | char * | 0 | BYE |

Extended **Reference lookup** mechanism

# TYPE SYSTEM IN CIDE



http://fosd.net/cide



# TYPE SYSTEM IN CIDE

```
public class Test {
        private static String msg_hi = "Hello world!”;
        private static String msg_bye = "Bye bye!";
        public static void main(String[] args) {
                System.out.println(msg_hi);
                System.out.println(msg_bye);
        }
}
```

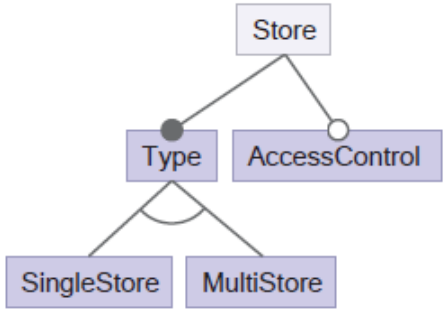
Similar implementations available for FOP, AOP




http://www.fosd.de/fuji

Feature *SingleStore*

```
1  class Store {
2    Object value;
3    Object read() { return value; }
4    void set(Object nvalue) { value = nvalue; }
5  }
```




Feature *MultiStore*

```
6  class Store {
7    LinkedList values = new LinkedList();
8    Object read() { return values.getFirst(); }
9    Object[] readAll() { return values.toArray(); }
10   void set(Object nvalue) { values.addFirst(nvalue); }
11 }
```

Feature *AccessControl*

```
12 refines class Store {
13   boolean sealed = false;
14   Object read() {
15     if (!sealed) { return Super().read(); }
16     else { throw new RuntimeException("Access denied!"); }
17   }
18   void set(Object nvalue) {
19     if (!sealed) { Super(Object).set(nvalue); }
20     else { throw new RuntimeException("Access denied!"); }
21   }
22 }
```

Feature *SingleStore*

```
1  class Store {
2    private Object value;
3    Object read() { return value; }
4    void set(Object nvalue) { value = nvalue; }
5  }
```

---

```
6  class Store {
7    private LinkedList values = new LinkedList();
8    Object read() { return values.getFirst(); }
9    Object[] readAll() { return values.toArray(); }
10   void set(Object nvalue) { values.addFirst(nvalue); }
11 }
```

---

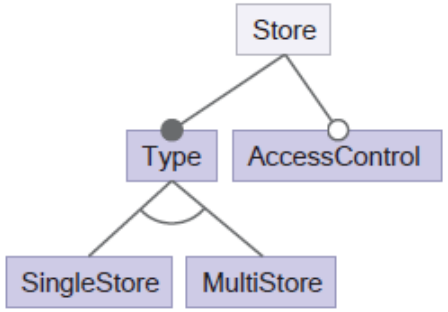
Feature *AccessControl'*

```
12 refines class Store {
13   boolean sealed = false;
14   Object read() {
15     if (!sealed) { return Super().read(); }
16     else { throw new RuntimeException("Access d
17   }
18   Object[] readAll() {
19     if (!sealed) { return Super().readAll(); }
20     else { throw new RuntimeException("Access denied!"); }
21   }
22   void set(Object nvalue) {
23     if (!sealed) { Super(Object).set(nvalue); }
24     else { throw new RuntimeException("Access denied!"); }
25   }
26 }
```

---

$$FM \Rightarrow$$
$$AccessControl \Rightarrow$$
$$SingleStore \vee MultiStore$$

$$FM \Rightarrow$$
$$AccessControl \Rightarrow$$
$$MultiStore$$




Glue-code module

# VARIABILITY ENCODING

```
1  class Node {
2    int id = 0;
3    private String name;
4    String getName() {
5        if (Conf.NAME) return name;
6        if (Conf.NONAME) return String.valueOf(id);
7        throw VariabilityException();
8    }
9    Color color = new Color();
10
11   void print() {
12     if (Conf.COLOR && Conf.NAME)
13         Color.setDisplayColor(color);
14     System.out.print(getName());
15   }
16 }
17
18 class Color {
19   static void setDisplayColor(Color c){/*...*/}
20 }
```

# Complete Analysis



**Product Configuration**



**Variability-Aware Analysis**

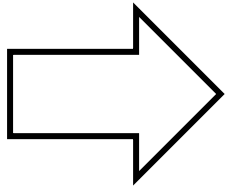
**Conventional Analysis**

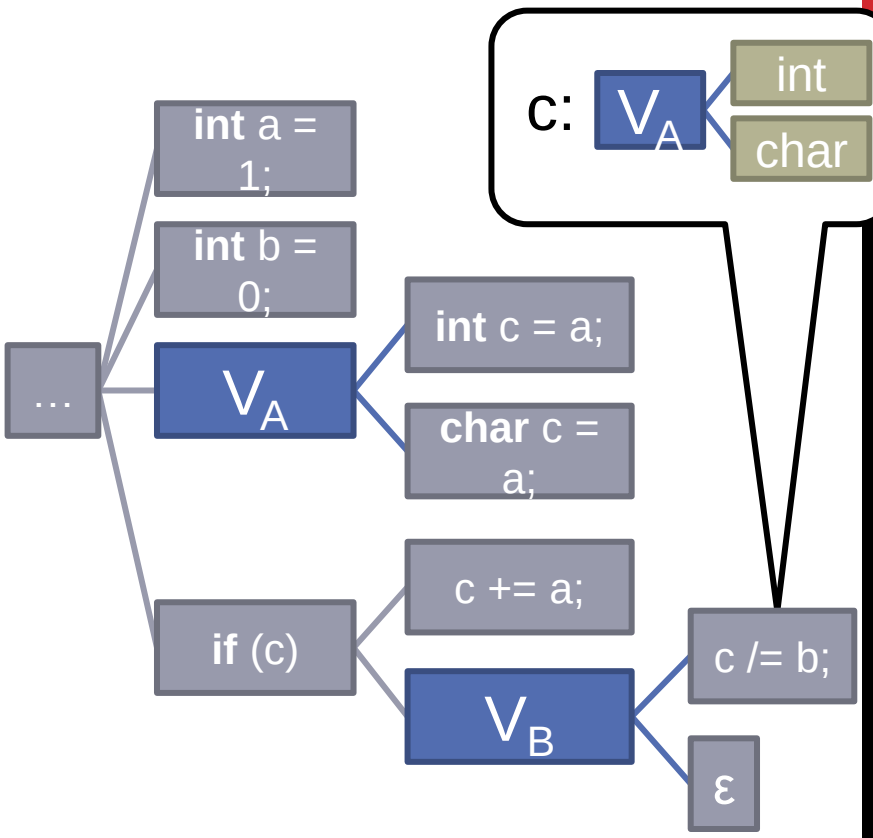




```
def getType(e: V[Expr]): V[Type]
```

```
1   int a = 1;
2   int b = 0;
3  #ifdef A
4   int c = a;
5  #else
6   char c = a;
7  #endif
8   if (c) {
9     c += a;
10 #ifdef B
11    c /= b;
12 #endif
13  }
14  ...
```

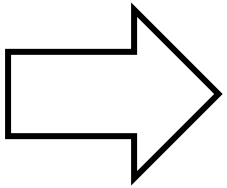


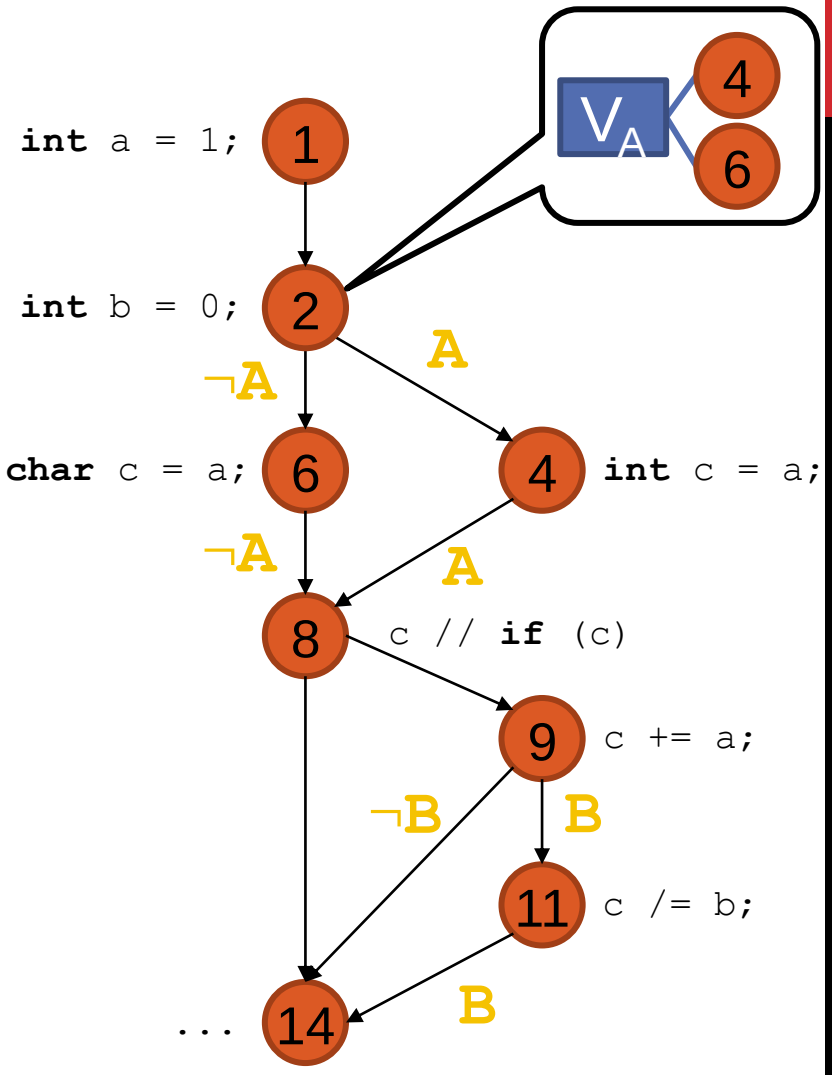



AST with Variability Information

```
def succ(s: Stmt): List[V[Stmt]]
```

```
1    int a = 1;
2    int b = 0;
3  #ifdef A
4    int c = a;
5  #else
6    char c = a;
7  #endif
8    if (c) {
9      c += a;
10 #ifdef B
11     c /= b;
12 #endif
13   }
14   ...
```

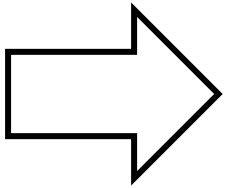


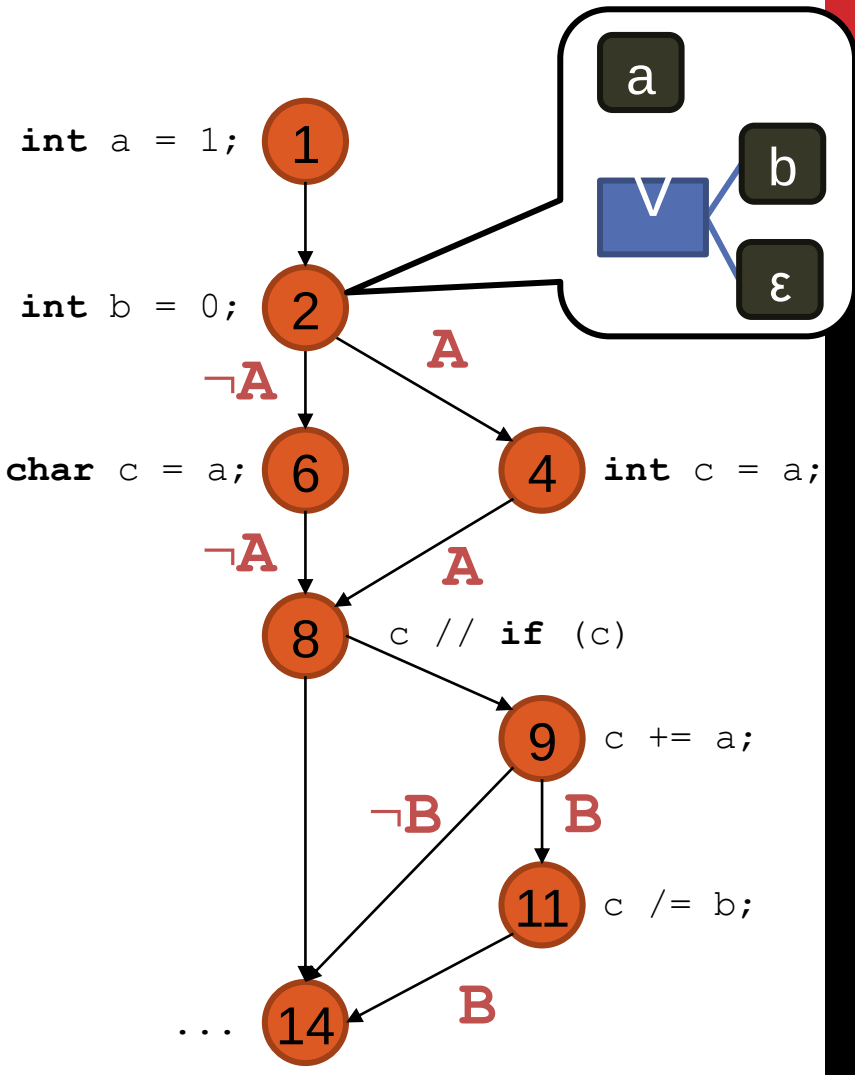



CFG with Variability Information

```
def live(s: Stmt): List[V[Var]]
```

```
1   int a = 1;
2   int b = 0;
3  #ifdef A
4   int c = a;
5  #else
6   char c = a;
7  #endif
8   if (c) {
9     c += a;
10 #ifdef B
11    c /= b;
12 #endif
13  }
14  ...
```

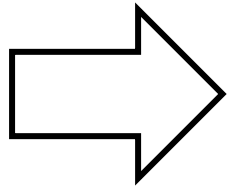





CFG with Variability Information

```
def forw(s: Stmt, p: State): List[V[State]]
```

```
1    int a = 1;
2    int b = 0;
3  #ifdef A
4    int c = a;
5  #else
6    char c = a;
7  #endif
8    if (c) {
9      c += a;
10 #ifdef B
11     c /= b;
12 #endif
13   }
14   ...
```






# ARG with Variability Informatio

# MODULAR ANALYSIS

## (FEATURE-BASED)

Feature module *Base*

```
1 class Stack { ...
2   void push(int v) {/*...*/}
3   int pop() {/*...*/}
4 }
```

```
interface Base;
export class Stack {
        void push(int v);
}
```

*Top*

```
1 refines class Stack {
2   int top() {/*...*/}
3 }
```

*Undo*

```
1 refines class Stack { ...
2   int backup;
3   void undo() {/*...*/}
4   void push(int v) {
5     backup=top();
6     original(v); //calling push of
7   }
8 }
```

```
interface Undo;
import class Stack {
        void push(int v);
        void top();
}
export class Stack {
        void undo();
}
```

```
1  layer BasicGraph;
2
3  class Graph {
4    private Vector nodes = new Vector();
5    private Vector edges = new Vector();
6    Edge add(Node n, Node m) {
7      Edge e = new Edge(n, m);
8      nodes.add(n);
9      nodes.add(m);
10     edges.add(e);
11     return e;
12   }
13   void print() { ... }
14 }
```

```
15 layer Weighted;
16
17 refines class Graph {
18   Edge add(Node n, Node m) {
19     Edge e = Super.add(n, m);
20     e.weight = new Weight();
21     return e;
22   }
23   Edge add(Node n, Node m, Weight w) {
24     Edge e = add(n, m);
25     e.weight = w;
26     return e;
27   }
28 }
```

```
29 layer AccessControl;
30
31 class Graph {
32   boolean sealed = false;
33
34   Edge add(Node n, Node m) {
35     if (!sealed) return Super.add(n, m);
36     else throw new RuntimeException(...);
37   }
38   Edge add(Node n, Node m, Weight w) {
39     if (!sealed) return Super.add(n,m,w);
40     else throw new RuntimeException(...);
41   }
42 }
```

⇒

```
1  layer BasicGraph;
2
3  provides class Graph {
4    provides Edge add(Node,Node)
5    provides void print()
6  }
```

```
7  layer Weighted;
8
9  requires class Graph {
10   requires Edge add(Node,Node)
11   provides Edge add(Node,Node,Weight)
12 }
```

```
13 layer AccessControl;
14
15 requires class Graph {
16   provides boolean sealed
17   requires Edge add(Node,Node)
18   requires Edge add(Node,Node,Weight)
19 }
```

# DISCUSSION

# PRINCIPLES

**Preserve sharing**

**Late splitting, early joining**

**Consider variability during analysis**

# ANALYSES

**Product-based**

**Family-based**

**Feature-based**

**Combinations thereof**

Thüm, T., Apel, S., Kästner, C., Schaefer, I., & Saake, G. (2014). A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys (CSUR)*, *47*(1), 6.

# LOAD-TIME VARIABILITY

# LOAD-TIME OPTIONS

**Read from configuration file/command-line options/…**

**Assumptions:**

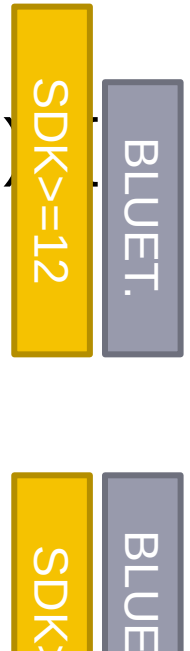
- Constant at program start, does not change after
- Used differently from other variables (assigned, used in if statements; not part of computations)

**(Tool: LoTrack based on taint analysis)**

# Data-flow interactions?

```
class ProxyService {
  static boolean NATIVE_PROXY_SUPPORTED
                    = Build.VERSION.SDK_INT >= 12;
  public void onSharedPreferenceChanged() {
    boolean a = false;
    if (!NATIVE_PROXY_SUPPORTED) {
        if (Context.getSystemService("bluetooth")
            a = true;
    }}

    if (a) {
```



# FURTHER READING

**Krzysztof Czarnecki, Krzysztof Pietroszek:
Verifying feature-based model templates against well-formedness OCL constraints. GPCE 2006: 211-220
– origin of whole-product line analysis**

**Christian Kästner, Sven Apel, Thomas Thüm, and Gunter Saake. 2012. Type checking annotation-based product lines. *ACM Trans. Softw. Eng. Methodol.* 21, 3, Article 14 (July 2012), 39 pages.**

**-- application to conditional compilation in Java code**

**TypeChef papers – application to ifdefs in C code, scaling to Linux**

**Thüm, T., Apel, S., Kästner, C., Schaefer, I., & Saake, G. (2014). A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys (CSUR)*, *47*(1), 6.**