

# Short Overview of Software Product Lines

Sarah Nadi  
Software Technology Group  
[www.sarahnadi.org](http://www.sarahnadi.org)



# Examples of Software Product Lines



Mobile OS



Control SW



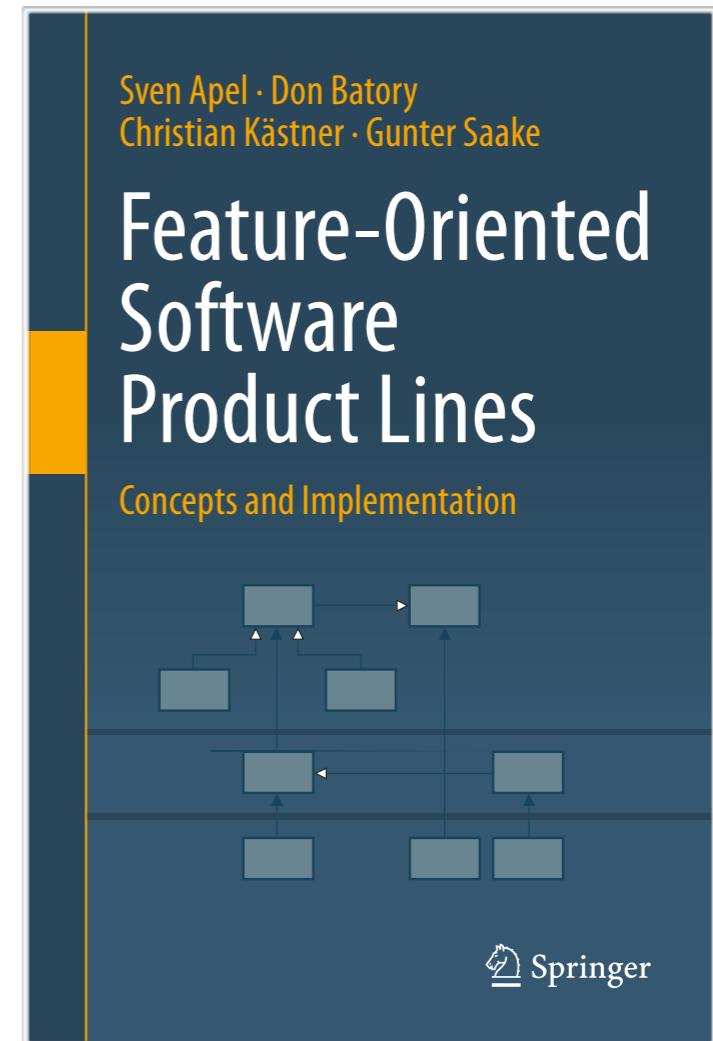
Printer Firmware



Linux Kernel

# Resources

- Slides largely based on:



# Software Product Lines

“A software product line (SPL) is a set of software-intensive systems that **share a common, managed set of features** satisfying the specific needs of a particular market segment or mission and that are **developed from a common set of core assets** in a prescribed way.”

— Software Engineering Institute  
Carnegie Mellon University

# Advantages of SPLs

- Tailor-made software
- Reduced cost
- Improved quality
- Reduced time to market

# Success Stories



# Challenges of SPLs

- Upfront cost for preparing reusable parts
- Deciding which products you can produce early on
- Thinking about multiple products at the same time
- Managing/testing/analyzing multiple products

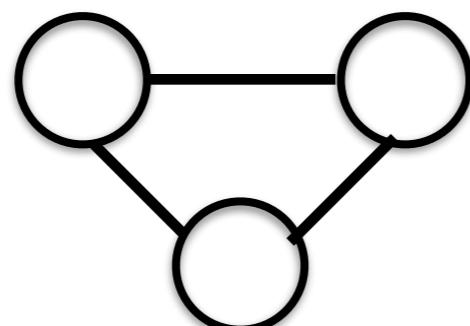
# Challenges of SPLs

- Upfront cost for preparing reusable parts
- Deciding which products you can produce early on
- Thinking about multiple products at the same time
- Managing/testing/analyzing multiple products

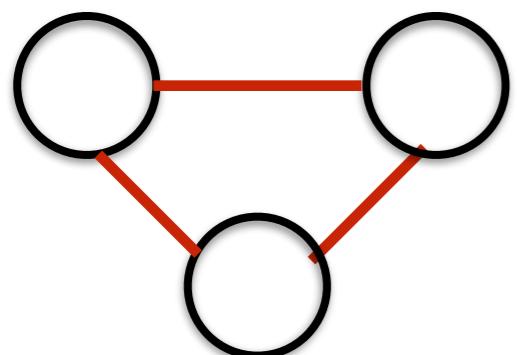
# Feature-oriented SPLs

- Thinking of your product line in terms of the features offered

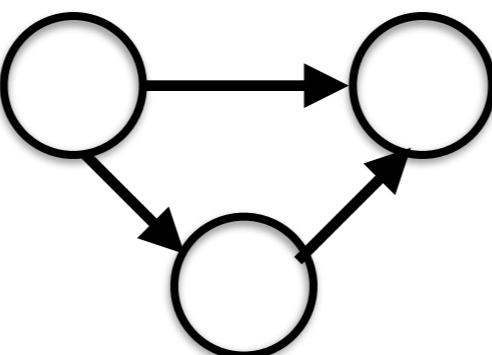
# Examples of a Feature



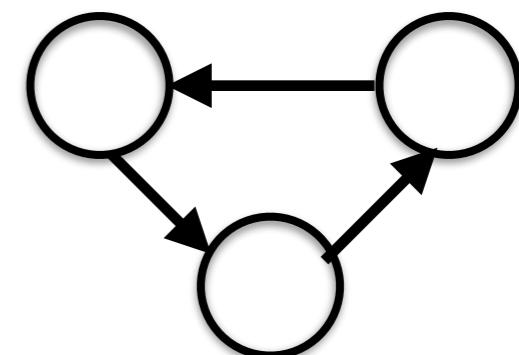
Graph product-line



feature:  
*edge color*



feature:  
*edge type*  
(Directed vs Undirected)



feature:  
*cycle detection*

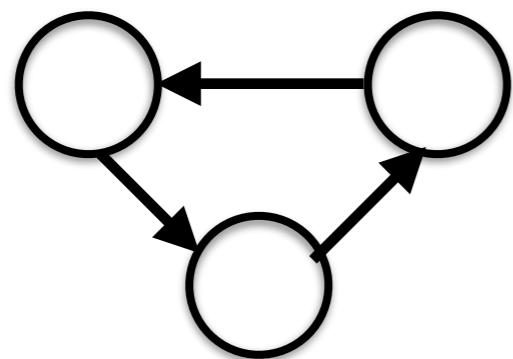
# Feature

**Definition 2.1** A *feature* is a characteristic or end-user-visible behavior of a software system. Features are used in product-line engineering to specify and communicate commonalities and differences of the products between stakeholders, and to guide structure, reuse, and variation across all phases of the software life cycle. □

Exercise:  
What features would a  
car SPL contain?

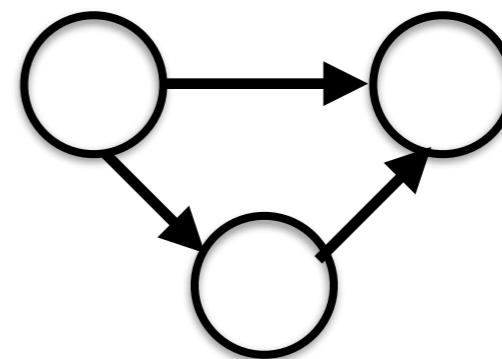
# Feature Dependencies

- Constraints on the possible feature selections



feature:  
cycle detection

depends on



feature:  
directed

# Product

**Definition 2.2** A *product* of a product line is specified by a valid feature selection (a subset of the features of the product line). A feature selection is *valid* if and only if it fulfills all *feature dependencies*. □

# Exercise:

## Which Product(s) are Invalid?

	Edge Color	Directed Edge	Cycle Detection
Product 1	✓	✓	✓
Product 2	✓		✓
Product 3		✓	✓

# Exercise:

## Which Product(s) are Invalid?

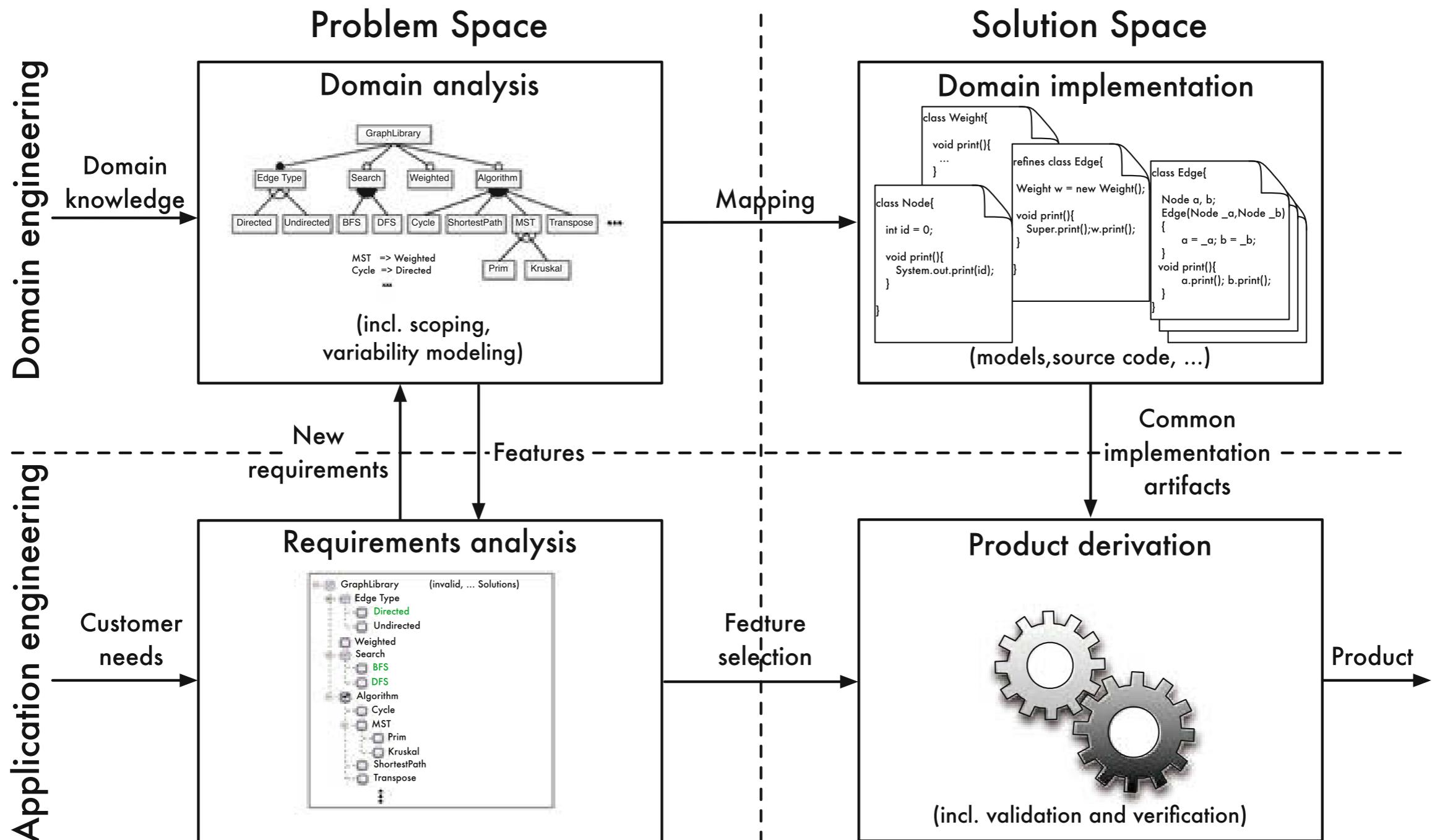
	Edge Color	Directed Edge	Cycle Detection
Product 1	✓	✓	✓
Product 2	✓		✓
Product 3		✓	✓

invalid  
product

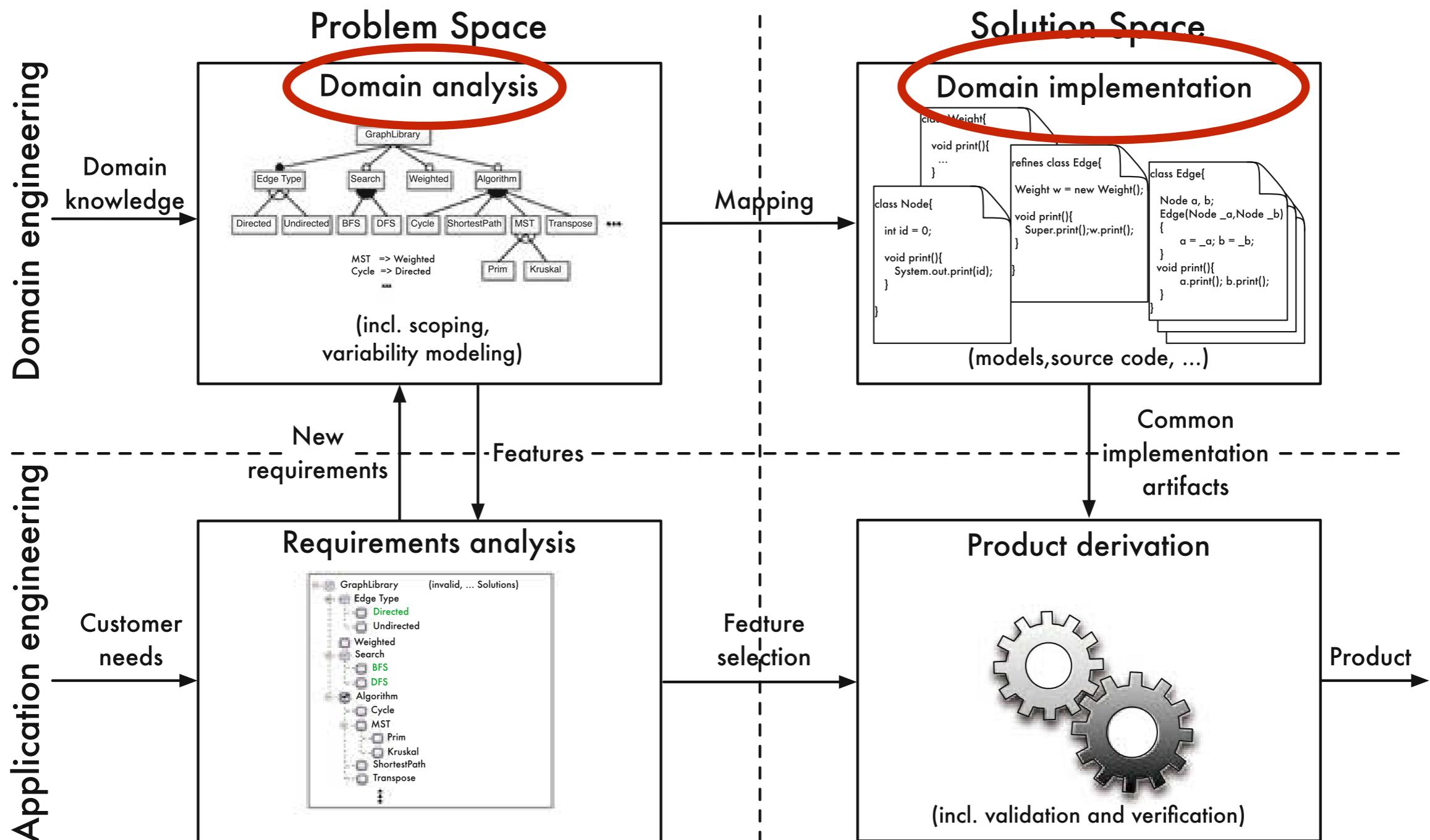
Cycle detection depends on Directed Edge

Exercise:  
What dependencies might exist  
between features in a car SPL?

# Software Product Line Engineering



# Software Product Line Engineering



# Domain Analysis

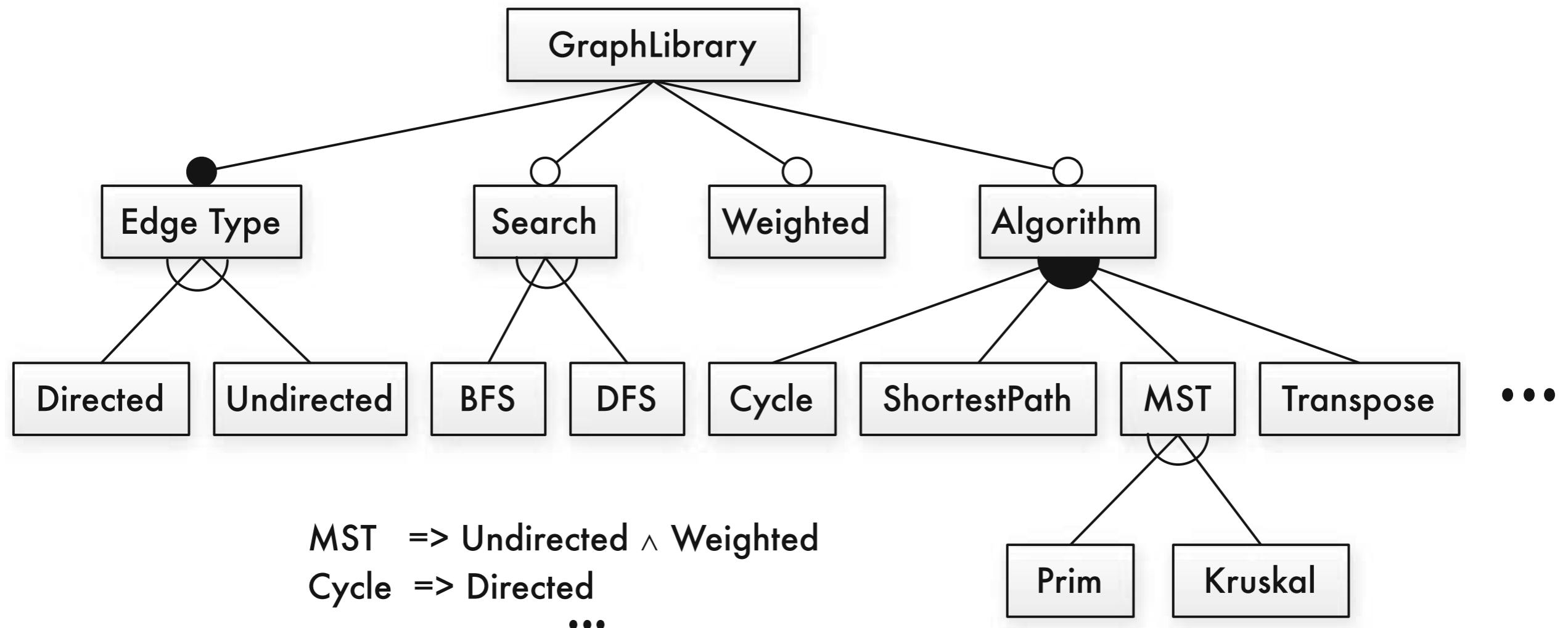
# Domain Analysis

- Domain scoping
  - Deciding on product line's extent or range
- Domain modeling
  - Captures & documents the commonalities & variabilities of the scoped domain
  - Often captured in a *feature model*

# Feature Models

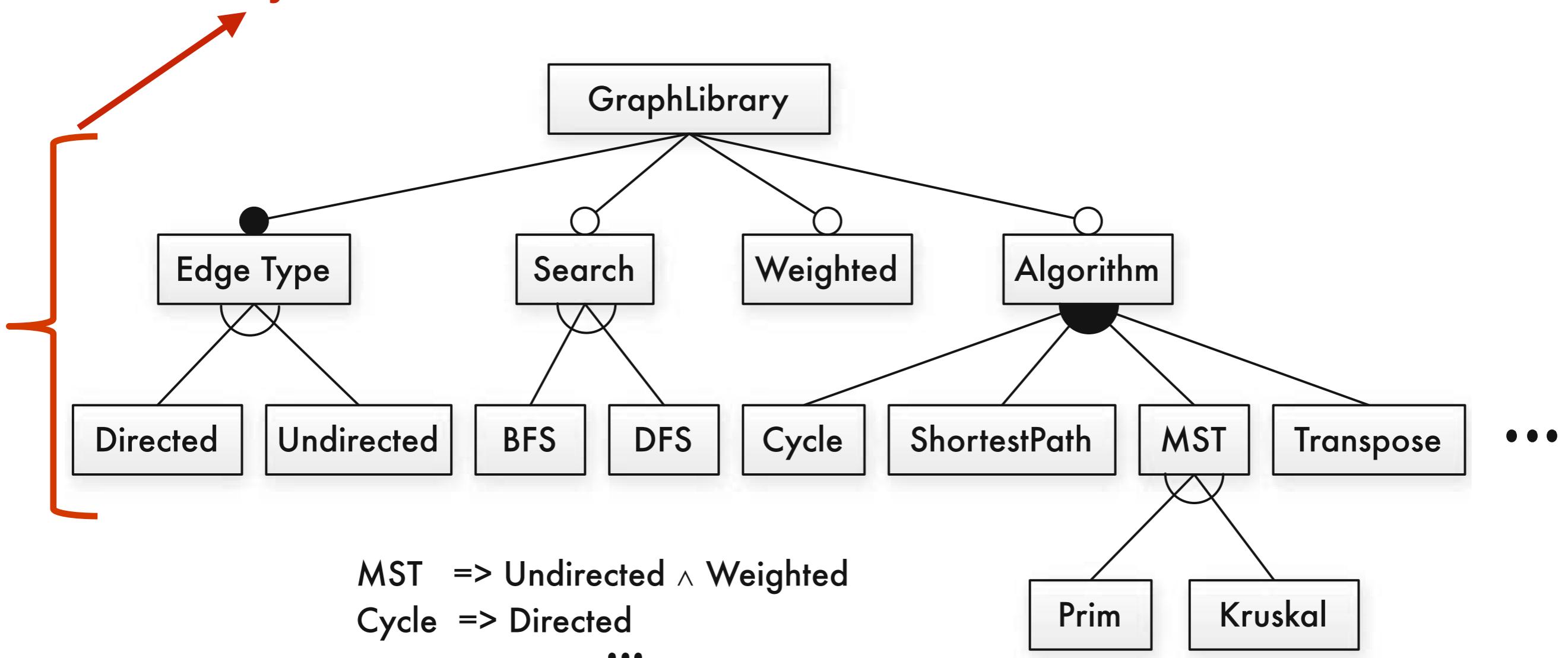
- Document the features of a product line & their relationships
- Can be translated into propositional logic

# Graph Library Feature Model



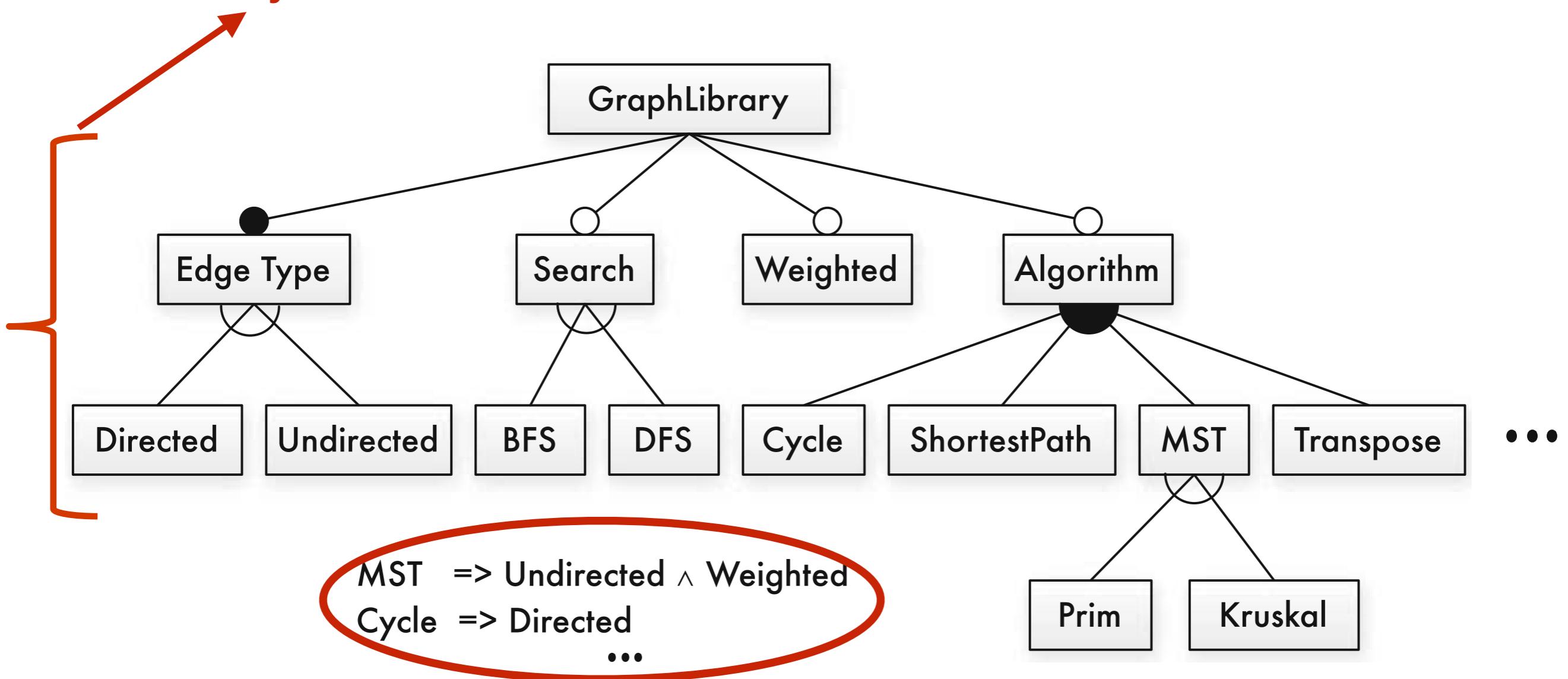
# Graph Library Feature Model

## Hierarchy Constraints



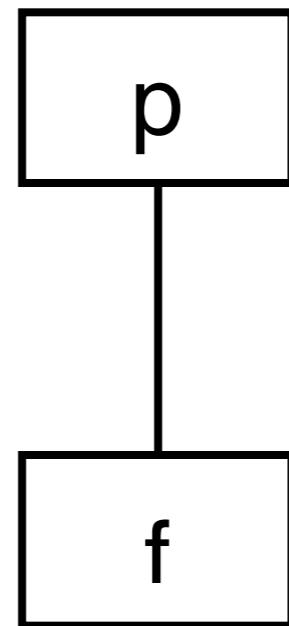
# Graph Library Feature Model

## Hierarchy Constraints



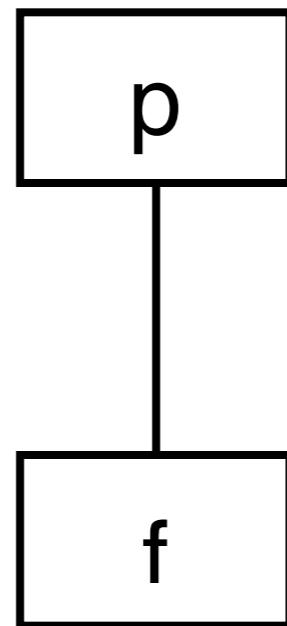
## Cross-tree Constraints

# Hierarchal Relationships

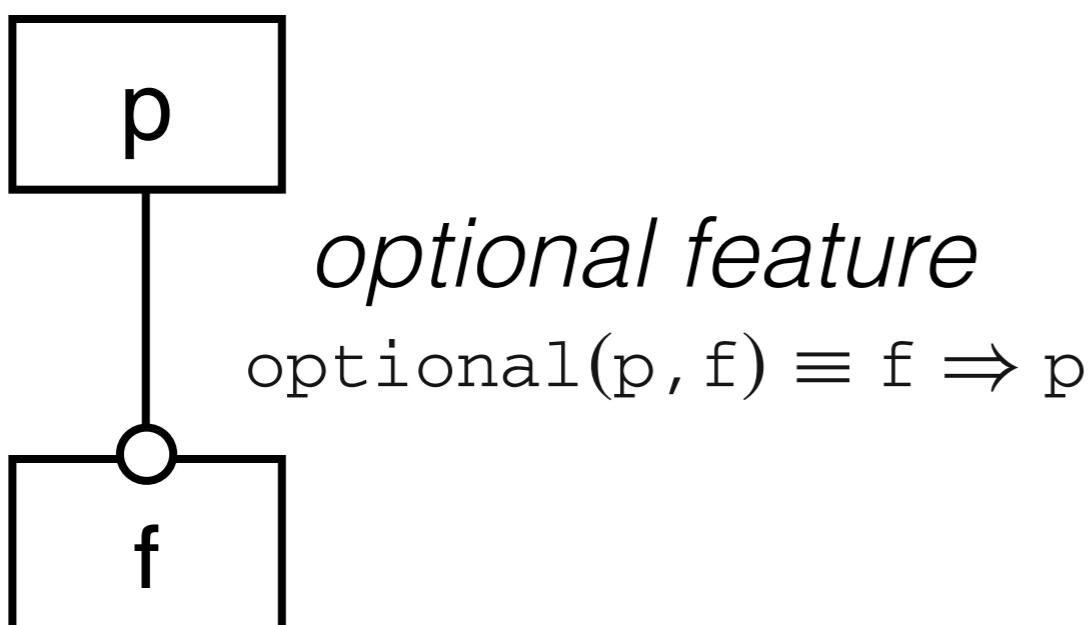


- Parent/child relationship
- Child cannot be selected unless parent is selected

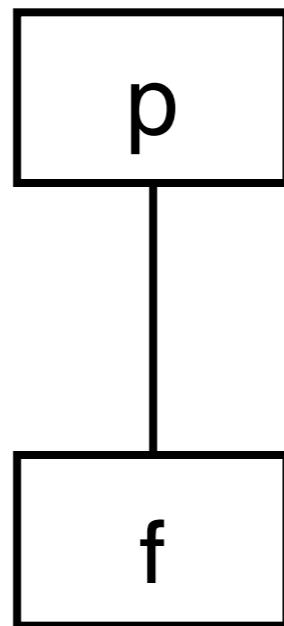
# Hierarchal Relationships



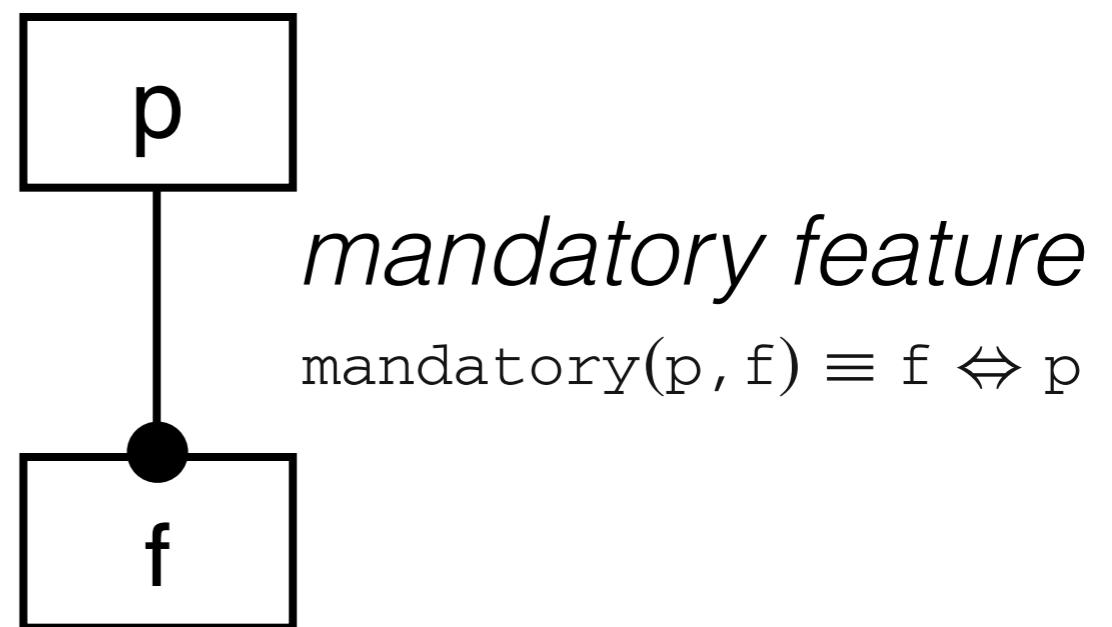
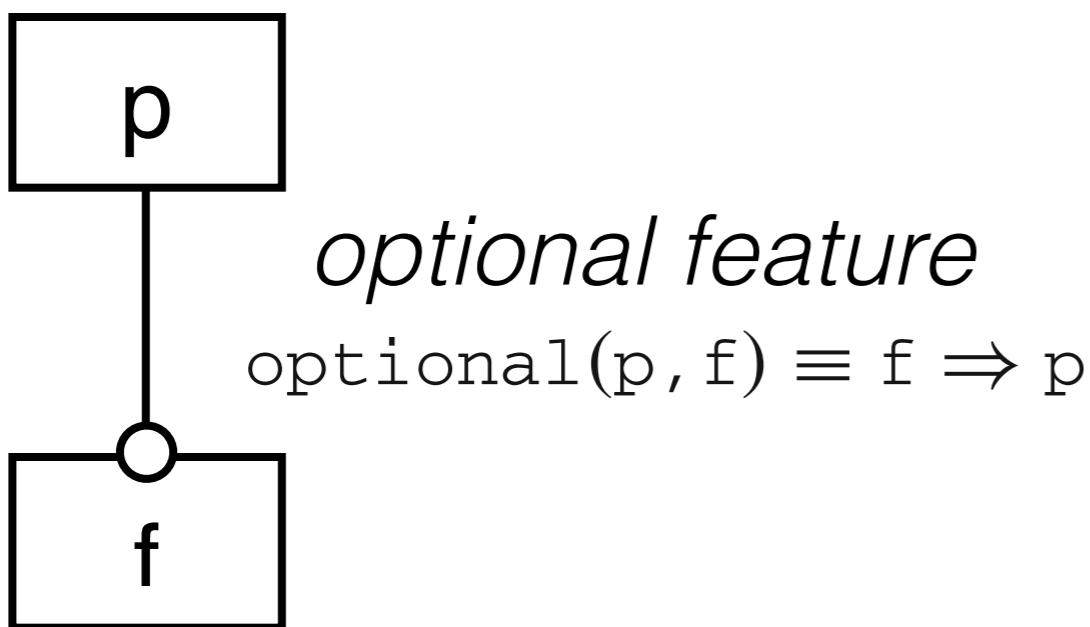
- Parent/child relationship
- Child cannot be selected unless parent is selected



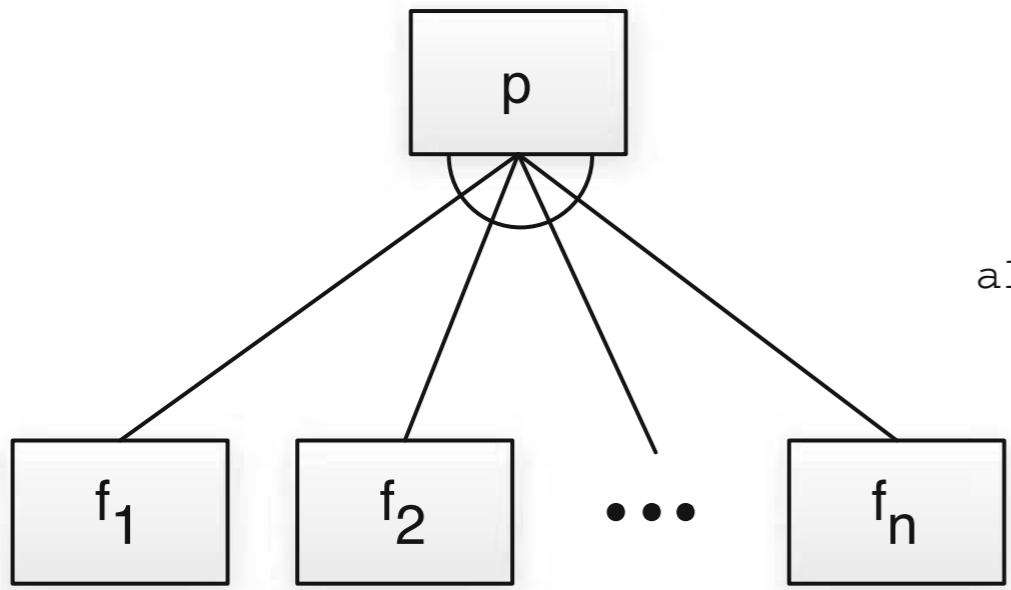
# Hierarchal Relationships



- Parent/child relationship
- Child cannot be selected unless parent is selected

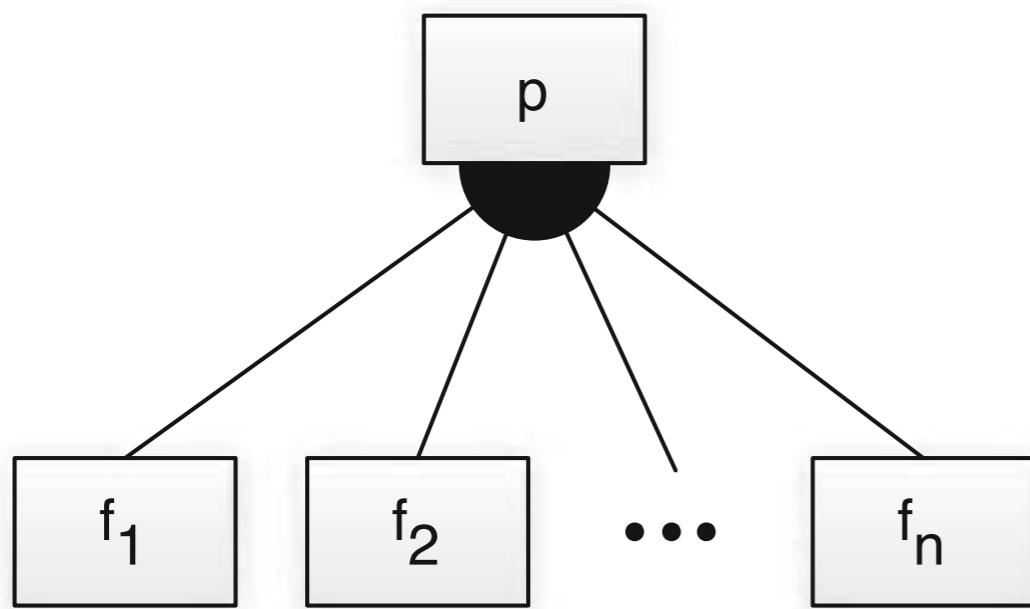


# Hierarchal Relationships (Groups)



xor group

$$\text{alternative}(p, \{f_1, \dots, f_n\}) \equiv ((f_1 \vee \dots \vee f_n) \Leftrightarrow p) \wedge \bigwedge_{i < j} \neg(f_i \wedge f_j)$$



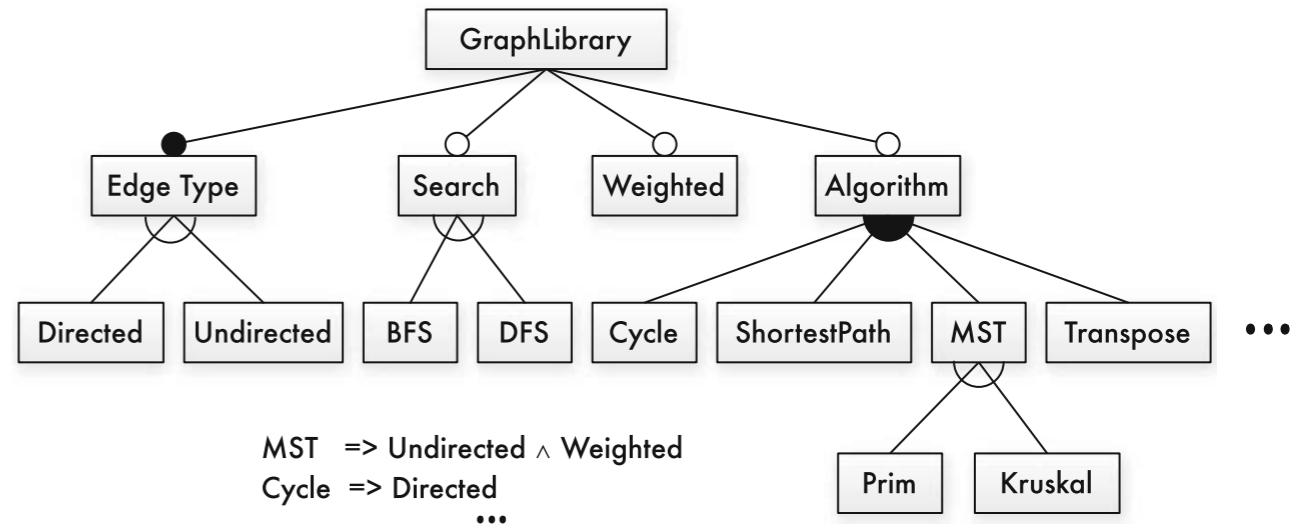
or group

$$\text{or}(p, \{f_1, \dots, f_n\}) \equiv (f_1 \vee \dots \vee f_n) \Leftrightarrow p$$

# Feature Model in Propositional Logic

GraphLibrary

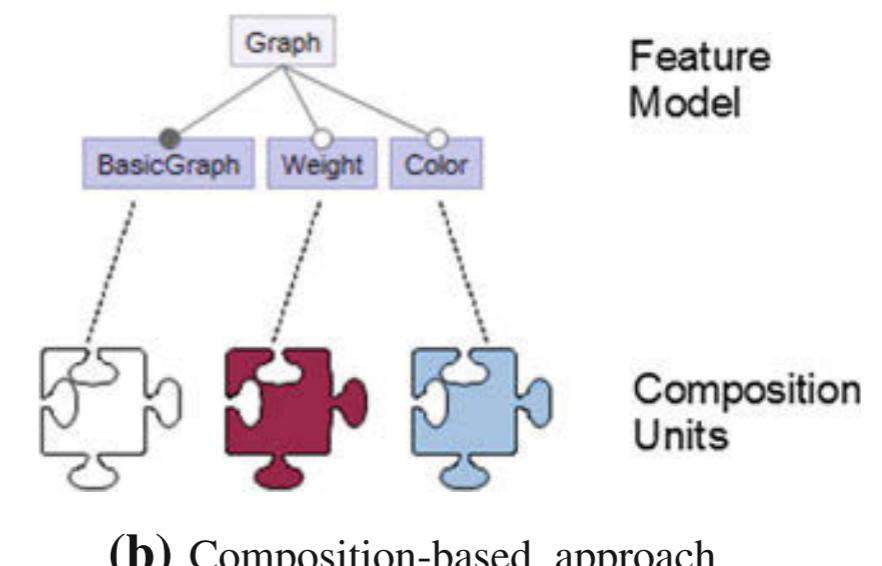
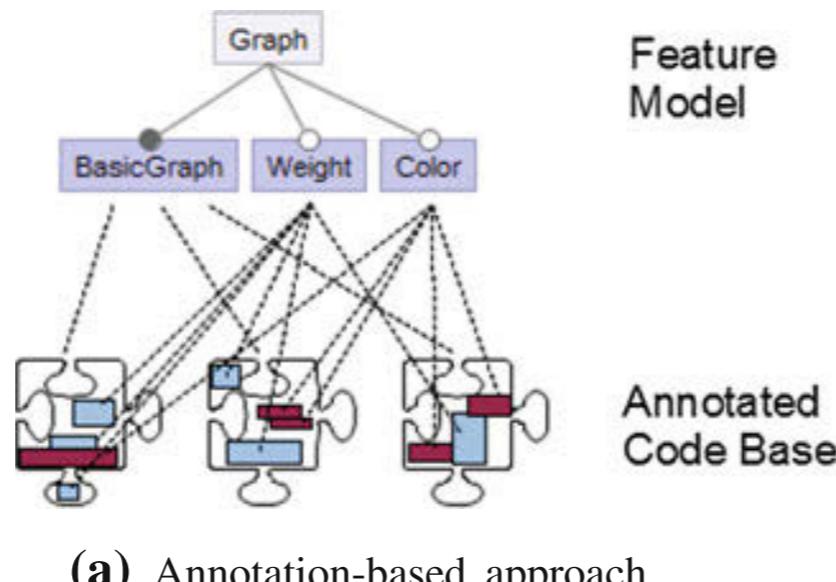
- $\wedge (\text{EdgeType} \Leftrightarrow \text{GraphLibrary})$
- $\wedge (\text{Search} \Rightarrow \text{EdgeType})$
- $\wedge (\text{Weighted} \Rightarrow \text{EdgeType})$
- $\wedge (\text{Algorithm} \Rightarrow \text{EdgeType})$
- $\wedge (((\text{Directed} \vee \text{Undirected}) \Leftrightarrow \text{EdgeType}) \wedge \neg(\text{Directed} \wedge \text{Undirected}))$
- $\wedge ((\text{BFS} \vee \text{DFS}) \Leftrightarrow \text{Search})$
- $\wedge ((\text{Cycle} \vee \text{ShortestPath} \vee \text{MST} \vee \text{Transpose}) \Leftrightarrow \text{Algorithm})$
- $\wedge (((\text{Prim} \vee \text{Kruskal}) \Leftrightarrow \text{MST}) \wedge \neg(\text{Prim} \wedge \text{Kruskal}))$
- $\wedge (\text{MST} \Rightarrow \text{Weighted})$
- $\wedge (\text{Cycle} \Rightarrow \text{Directed})$
- $\wedge (\dots)$



# Domain Implementation

# Domain Implementation

- Underlying code must be *variable*
- Dimensions of implementation techniques
  - *Binding times*: compile-time binding, load-time binding, and run-time binding.
  - *Representation*: annotation vs composition



# Variability Implementation

- Parameters
- Build systems
- Preprocessors
- ...



# Working Example: Basic Graph Library (Java)

---

```
1 class Graph {  
2     Vector nodes = new Vector();  
3     Vector edges = new Vector();  
4     Edge add(Node n, Node m) {  
5         Edge e = new Edge(n,m);  
6         nodes.add(n);  
7         nodes.add(m);  
8         edges.add(e);  
9         return e;  
10    }  
11    void print() {  
12        for(int i=0; i<edges.size(); i++){  
13            ((Edge) edges.get(i)).print();  
14            if(i < edges.size() - 1)  
15                System.out.print(" , ");  
16        }  
17    }  
18 }
```

---

```
19 class Node {  
20     int id = 0;  
21     Node (int _id) { id = _id; }  
22     void print() {System.out.print(id);}  
23 }  
24  
25  
26 class Edge {  
27     Node a, b;  
28     Edge(Node _a, Node _b) {a=_a; b=_b;}  
29     void print() {  
30         System.out.print(" (");  
31         a.print();  
32         System.out.print(" , ");  
33         b.print();  
34         System.out.print(") ");  
35     }  
36 }
```

---

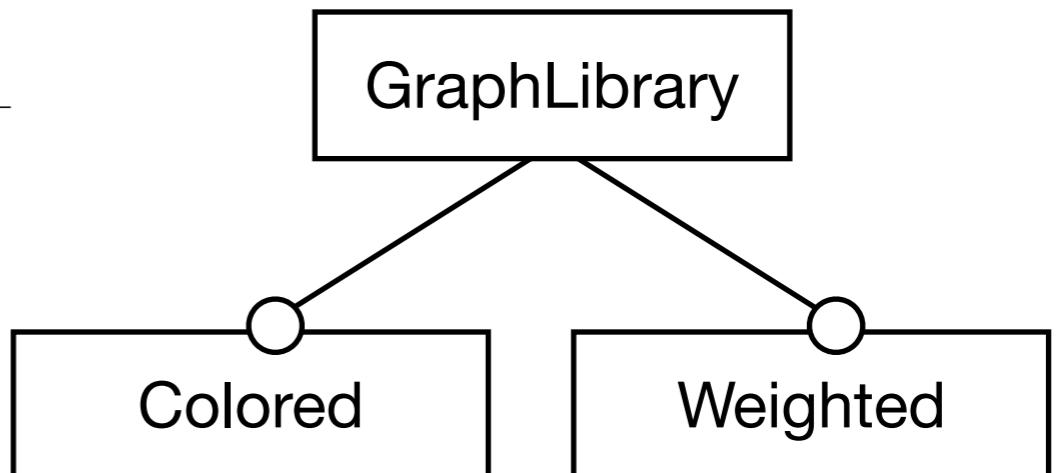
# Working Example: Basic Graph Library (Java)

---

```
1 class Graph {  
2     Vector nodes = new Vector();  
3     Vector edges = new Vector();  
4     Edge add(Node n, Node m) {  
5         Edge e = new Edge(n,m);  
6         nodes.add(n);  
7         nodes.add(m);  
8         edges.add(e);  
9         return e;  
10    }  
11    void print() {  
12        for(int i=0; i<edges.size(); i++){  
13            ((Edge) edges.get(i)).print();  
14            if(i < edges.size() - 1)  
15                System.out.print(" , ");  
16        }  
17    }  
18 }
```

---

```
19 class Node {  
20     int id = 0;  
21     Node (int _id) { id = _id; }  
22     void print() {System.out.print(id);}  
23 }  
24  
25  
26 class Edge {  
27     Node a, b;  
28     Edge(Node _a, Node _b) {a=_a; b=_b;}  
29     void print() {  
30         System.out.print(" (");  
31         a.print();  
32         System.out.print(" , ");  
33         b.print();  
34         System.out.print(" )");  
35     }  
36 }
```



# Parameters

# Variability using Parameters

---

```
1 class Conf {
2     public static boolean COLORED = true;
3     public static boolean WEIGHTED = false;
4 }
5
6
7 class Graph {
8     Vector nodes = new Vector();
9     Vector edges = new Vector();
10    Edge add(Node n, Node m) {
11        Edge e = new Edge(n,m);
12        nodes.add(n);
13        nodes.add(m);
14        edges.add(e);
15        if (Conf.WEIGHTED)
16            e.weight = new Weight();
17        return e;
18    }
19    Edge add(Node n, Node m, Weight w) {
20        if (!Conf.WEIGHTED)
21            throw new RuntimeException();
22        Edge e = new Edge(n, m);
23        e.weight = w;
24        nodes.add(n);
25        nodes.add(m);
26        edges.add(e);
27        return e;
28    }
29    void print() {
30        for(int i=0; i<edges.size(); i++){
31            ((Edge) edges.get(i)).print();
32            if(i < edges.size() - 1)
33                System.out.print(" , ");
34        }
35    }
36 }
```

---

```
37 class Node {
38     int id = 0;
39     Color color = new Color();
40     Node (int _id) { id = _id; }
41     void print() {
42         if (Conf.COLORED)
43             Color.setDisplayColor(color);
44         System.out.print(id);
45     }
46 }
47
48
49 class Edge {
50     Node a, b;
51     Color color = new Color();
52     Weight weight;
53     Edge(Node _a, Node _b) {a=_a; b=_b;}
54     void print() {
55         if (Conf.COLORED)
56             Color.setDisplayColor(color);
57         System.out.print(" (" );
58         a.print();
59         System.out.print(" , ");
60         b.print();
61         System.out.print(" ) ");
62         if (Conf.WEIGHTED) weight.print();
63     }
64 }
65
66
67 class Color {
68     static void setDisplayColor(Color c)...
69 }
70 class Weight {
71     void print() { ... }
72 }
```

# Variability using Parameters

```
1 class Conf {
2     public static boolean COLORED = true;
3     public static boolean WEIGHTED = false;
4 }
5
6
7 class Graph {
8     Vector nodes = new Vector();
9     Vector edges = new Vector();
10    Edge add(Node n, Node m) {
11        Edge e = new Edge(n,m);
12        nodes.add(n);
13        nodes.add(m);
14        edges.add(e);
15        if (Conf.WEIGHTED)
16            e.weight = new Weight();
17        return e;
18    }
19    Edge add(Node n, Node m, Weight w) {
20        if (!Conf.WEIGHTED)
21            throw new RuntimeException();
22        Edge e = new Edge(n, m);
23        e.weight = w;
24        nodes.add(n);
25        nodes.add(m);
26        edges.add(e);
27        return e;
28    }
29    void print() {
30        for(int i=0; i<edges.size(); i++){
31            ((Edge) edges.get(i)).print();
32            if(i < edges.size() - 1)
33                System.out.print(" , ");
34        }
35    }
36 }
```

---

```
37 class Node {
38     int id = 0;
39     Color color = new Color();
40     Node (int _id) { id = _id; }
41     void print() {
42         if (Conf.COLORED)
43             Color.setDisplayColor(color);
44         System.out.print(id);
45     }
46 }
47
48
49 class Edge {
50     Node a, b;
51     Color color = new Color();
52     Weight weight;
53     Edge(Node _a, Node _b) {a=_a; b=_b;}
54     void print() {
55         if (Conf.COLORED)
56             Color.setDisplayColor(color);
57         System.out.print(" (" );
58         a.print();
59         System.out.print(" , ");
60         b.print();
61         System.out.print(" ) ");
62         if (Conf.WEIGHTED) weight.print();
63     }
64 }
65
66
67 class Color {
68     static void setDisplayColor(Color c)...
69 }
70 class Weight {
71     void print() { ... }
72 }
```

# Variability using Parameters

```
1 class Conf {
2     public static boolean COLORED = true;
3     public static boolean WEIGHTED = false;
4 }
5
6
7 class Graph {
8     Vector nodes = new Vector();
9     Vector edges = new Vector();
10    Edge add(Node n, Node m) {
11        Edge e = new Edge(n,m);
12        nodes.add(n);
13        nodes.add(m);
14        edges.add(e);
15        if (Conf.WEIGHTED)
16            e.weight = new Weight();
17        return e;
18    }
19    Edge add(Node n, Node m, Weight w) {
20        if (!Conf.WEIGHTED)
21            throw new RuntimeException();
22        Edge e = new Edge(n, m);
23        e.weight = w;
24        nodes.add(n);
25        nodes.add(m);
26        edges.add(e);
27        return e;
28    }
29    void print() {
30        for(int i=0; i<edges.size(); i++){
31            ((Edge) edges.get(i)).print();
32            if(i < edges.size() - 1)
33                System.out.print(" , ");
34        }
35    }
36 }
```

```
37 class Node {
38     int id = 0;
39     Color color = new Color();
40     Node (int _id) { id = _id; }
41     void print() {
42         if (Conf.COLORED)
43             Color.setDisplayColor(color);
44         System.out.print(id);
45     }
46 }
47
48
49 class Edge {
50     Node a, b;
51     Color color = new Color();
52     Weight weight;
53     Edge(Node _a, Node _b) {a=_a; b=_b;}
54     void print() {
55         if (Conf.COLORED)
56             Color.setDisplayColor(color);
57         System.out.print(" (" );
58         a.print();
59         System.out.print(" , ");
60         b.print();
61         System.out.print(" ) ");
62         if (Conf.WEIGHTED) weight.print();
63     }
64 }
65
66
67 class Color {
68     static void setDisplayColor(Color c)...
69 }
70 class Weight {
71     void print() { ... }
72 }
```

# Variability using Parameters

```
1 class Conf {
2     public static boolean COLORED = true;
3     public static boolean WEIGHTED = false;
4 }
5
6
7 class Graph {
8     Vector nodes = new Vector();
9     Vector edges = new Vector();
10    Edge add(Node n, Node m) {
11        Edge e = new Edge(n,m);
12        nodes.add(n);
13        nodes.add(m);
14        edges.add(e);
15        if (Conf.WEIGHTED)
16            e.weight = new Weight();
17        return e;
18    }
19    Edge add(Node n, Node m, Weight w) {
20        if (!Conf.WEIGHTED)
21            throw new RuntimeException();
22        Edge e = new Edge(n, m);
23        e.weight = w;
24        nodes.add(n);
25        nodes.add(m);
26        edges.add(e);
27        return e;
28    }
29    void print() {
30        for(int i=0; i<edges.size(); i++){
31            ((Edge) edges.get(i)).print();
32            if(i < edges.size() - 1)
33                System.out.print(" , ");
34        }
35    }
36 }
```

```
37 class Node {
38     int id = 0;
39     Color color = new Color();
40     Node (int _id) { id = _id; }
41     void print() {
42         if (Conf.COLORED)
43             Color.setDisplayColor(color);
44         System.out.print(id);
45     }
46 }
47
48
49 class Edge {
50     Node a, b;
51     Color color = new Color();
52     Weight weight;
53     Edge(Node _a, Node _b) {a=_a; b=_b;}
54     void print() {
55         if (Conf.COLORED)
56             Color.setDisplayColor(color);
57         System.out.print(" (" );
58         a.print();
59         System.out.print(" , ");
60         b.print();
61         System.out.print(" ) ");
62         if (Conf.WEIGHTED) weight.print();
63     }
64 }
65
66
67 class Color {
68     static void setDisplayColor(Color c)...
69 }
70 class Weight {
71     void print() { ... }
72 }
```

# Variability using Parameters

- + simple
- + flexible
- + language support
- code bloat
- computing overhead
- non-modular solution

# Variability using Parameters

- + simple
- + flexible
- + language support
- code bloat
- computing overhead
- non-modular solution



# Build Systems

# Variability Using Build Scripts

---

```
1 #!/bin/bash -e
2
3 rm *.class
4 javac Graph.java Edge.java Node.java \
5     Color.java
6 jar cvf graph.jar *.class
```

---

No variability

```
1 #!/bin/bash -e
2
3 if test "$1" = "--withColor"; then
4     cp Edge_withColor.java Edge.java
5     cp Node_withColor.java Node.java
6 else
7     cp Edge_withoutColor.java Edge.java
8     cp Node_withoutColor.java Node.java
9 fi
10
11 rm *.class
12 javac Graph.java Edge.java Node.java
13 if test "$1" = "--withColor"; then
14     javac Color.java
15 fi
16
17 jar cvf graph.jar *.class
```

---

With variability

# Variability Using Build Scripts

---

```
1 #!/bin/bash -e
2
3 rm *.class
4 javac Graph.java Edge.java Node.java \
5     Color.java
6 jar cvf graph.jar *.class
```

---

No variability

```
1 #!/bin/bash -e
2
3 if test "$1" = "--withColor"; then
4     cp Edge_withColor.java Edge.java
5     cp Node_withColor.java Node.java
6 else
7     cp Edge_withoutColor.java Edge.java
8     cp Node_withoutColor.java Node.java
9 fi
10
11 rm *.class
12 javac Graph.java Edge.java Node.java
13 if test "$1" = "--withColor"; then
14     javac Color.java
15 fi
16
17 jar cvf graph.jar *.class
```

---

With variability

# Variability Using Build Scripts

*Annotation  
Compile-time*

- + simple if features can be mapped into files
- + can control other types of parameters
- code duplication if finer level of granularity needed
- hard to analyze

# Preprocessors

# Variable Preprocessors Using

```
1 class Graph {
2     Vector nodes = new Vector();
3     Vector edges = new Vector();
4     Edge add(Node n, Node m) {
5         Edge e = new Edge(n,m);
6         nodes.add(n);
7         nodes.add(m);
8         edges.add(e);
9         /*IF[FEAT_COLORED]*/
10        e.weight = new Weight();
11        /*END[FEAT_COLORED]*/
12        return e;
13    }
14    /*IF[FEAT_COLORED]*/
15    Edge add(Node n, Node m, Weight w) {
16        Edge e = new Edge(n, m);
17        e.weight = w;
18        nodes.add(n);
19        nodes.add(m);
20        edges.add(e);
21        return e;
22    }
23    /*END[FEAT_COLORED]*/
24    void print() {
25        for(int i=0; i<edges.size(); i++){
26            ((Edge) edges.get(i)).print();
27            if(i < edges.size() - 1)
28                System.out.print(" , ");
29        }
30    }
31 }
32
33 /*IF[FEAT_COLORED]*/
34 class Color {
35     static void setDisplayColor(Color c)...
36 }
37 /*END[FEAT_COLORED]*/
38
39 class Node {
40     int id = 0;
41     /*IF[FEAT_COLORED]*/
42     Color color = new Color();
43     /*END[FEAT_COLORED]*/
44     Node (int _id) { id = _id; }
45     void print() {
46         /*IF[FEAT_COLORED]*/
47         Color.setDisplayColor(color);
48         /*END[FEAT_COLORED]*/
49         System.out.print(id);
50     }
51 }
52
53 class Edge {
54     Node a, b;
55     /*IF[FEAT_COLORED]*/
56     Color color = new Color();
57     /*END[FEAT_COLORED]*/
58     /*IF[FEAT_COLORED]*/
59     Weight weight;
60     /*END[FEAT_COLORED]*/
61     Edge(Node _a, Node _b) {a=_a; b=_b;}
62     void print() {
63         /*IF[FEAT_COLORED]*/
64         Color.setDisplayColor(color);
65         /*END[FEAT_COLORED]*/
66         System.out.print(" (");
67         a.print();
68         System.out.print(" , ");
69         b.print();
70         System.out.print(" ) ");
71         /*IF[FEAT_COLORED]*/
72         weight.print();
73         /*END[FEAT_COLORED]*/
74     }
75 }
76
77 /*IF[FEAT_COLORED]*/
78 class Weight {
79     void print() { ... }
80 }
81 /*END[FEAT_COLORED]*/
```

# Variability using the C Preprocessor

Can you spot the error?

---

```
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 #ifdef B
10     c += a;
11     c /= b;
12 }
13 #endif
```

---

# Variability using the C Preprocessor

Can you spot the error?

```
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 #ifdef B  
10     c += a;  
11     c /= b;  
12 }  
13 #endif
```

Compile time:  
no matching closing  
braces when B is not  
selected

# Variability using the C Preprocessor

Can you spot the error?

```
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 #ifdef B  
10     c += a;  
11     c /= b;  
12 }  
13 #endif
```

Compile time:  
no matching closing  
braces when B is not  
selected

Runtime:  
division by zero  
when B is selected

# Variability Using Preprocessors

- + Easy to use, well-known
- + Compile-time customization removes unnecessary code
- + Supports arbitrary levels of granularity
- No separation of concerns (lots of scattering & tangling)
- Can be used in an undisciplined fashion
- Prone to simple (syntactic) errors

# Variability Using Preprocessors

- + Easy to use, well-known
- + Compile-time customization removes unnecessary code
- + Supports arbitrary levels of granularity
- No separation of concerns (lots of scattering & tangling)
- Can be used in an undisciplined fashion
- Prone to simple (syntactic) errors

**Annotation  
Compile-time**

# Sample Research Topics

# Detecting Inconsistencies

PCCARD => HOTPLUG  
PCMCIA => PCCARD

Feature Model

```
#ifdef HOTPLUG
//B1
#else
//B2
#endif
```

ds.c  
Code

ds.c <=> PCMCIA

Build Files

# Detecting Inconsistencies

3  
4

PCCARD => HOTPLUG  
PCMCIA => PCCARD

1

```
#ifdef HOTPLUG  
//B1  
#else  
//B2  
#endif
```

2

ds.c <=> PCMCIA

Feature Model

ds.c  
Code

Build Files

1  
2  
3  
4

B2  $\wedge$   
B2 <=> !HOTPLUG  $\wedge$   
PCMCIA  $\wedge$   
PCMCIA => PCCARD  $\wedge$   
PCCARD => HOTPLUG

[ Nadi & Holt: CSMR '12 ]

# Detecting Inconsistencies

3  
4

PCCARD => HOTPLUG  
PCMCIA => PCCARD

1

```
#ifdef HOTPLUG  
//B1  
#else  
//B2  
#endif
```

2

ds.c <=> PCMCIA

Feature Model

ds.c  
Code

Build Files

B2  $\wedge$   
1 B2  $\Leftrightarrow$  !HOTPLUG  $\wedge$   
2 PCMCIA  $\wedge$   
3 PCMCIA => PCCARD  $\wedge$   
4 PCCARD => HOTPLUG

[ Nadi & Holt: CSMR '12 ]

# Detecting Inconsistencies

3  
4

PCCARD => HOTPLUG  
PCMCIA => PCCARD

1

```
#ifdef HOTPLUG  
//B1  
#else  
//B2  
#endif
```

2

ds.c <=> PCMCIA

Feature Model

ds.c  
Code

Build Files

1  
2  
3  
4

B2  $\wedge$   
B2 <=> !HOTPLUG  $\wedge$   
PCMCIA  $\wedge$   
PCMCIA => PCCARD  $\wedge$   
PCCARD => HOTPLUG

B2 is  
dead

[ Nadi & Holt: CSMR '12 ]

# Detecting Configuration Constraints

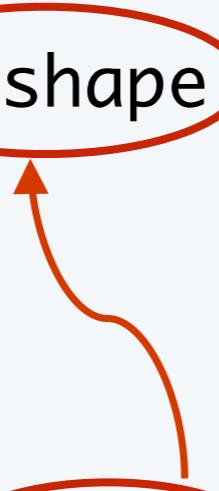
```
#ifdef SHAPE
    static shape *myshape;
#endif

int main(){
#ifdef AREA
    double area = myshape-> area;
#endif
}
```

# Detecting Configuration Constraints

```
#ifdef SHAPE
    static shape *myshape;
#endif

int main(){
    #ifdef AREA
        double area = myshape-> area;
    #endif
}
```



# Detecting Configuration Constraints

```
#ifdef SHAPE
    static shape *myshape;
#endif

int main(){
    #ifdef AREA
        double area = myshape-> area;
    #endif
}
```

Type error if  
AREA  $\wedge$  !SHAPE

# Detecting Configuration Constraints

```
#ifdef SHAPE  
    static shape *myshape;  
#endif  
  
int main(){  
    #ifdef AREA  
        double area = myshape-> area;  
    #endif  
}
```

Type error if  
 $\text{AREA} \wedge \neg \text{SHAPE}$

Feature model should enforce  $\neg (\text{AREA} \wedge \neg \text{SHAPE})$

# Detecting Configuration Constraints

```
#ifdef SHAPE  
static shape *myshape;  
#endif  
  
int main(){  
#ifdef AREA  
    double area = myshape-> area;  
#endif  
}
```

Type error if  
 $\text{AREA} \wedge \neg \text{SHAPE}$

Constraint:  
 $\text{AREA} \Rightarrow \text{SHAPE}$

Feature model should enforce  $\neg (\text{AREA} \wedge \neg \text{SHAPE})$

# Detecting Configuration Constraints (Underlying Analysis)

```
#include <stdio.h>

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

#ifndef BYE
char * msg = "Bye bye!\n";
#endif

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

# Detecting Configuration Constraints (Underlying Analysis)

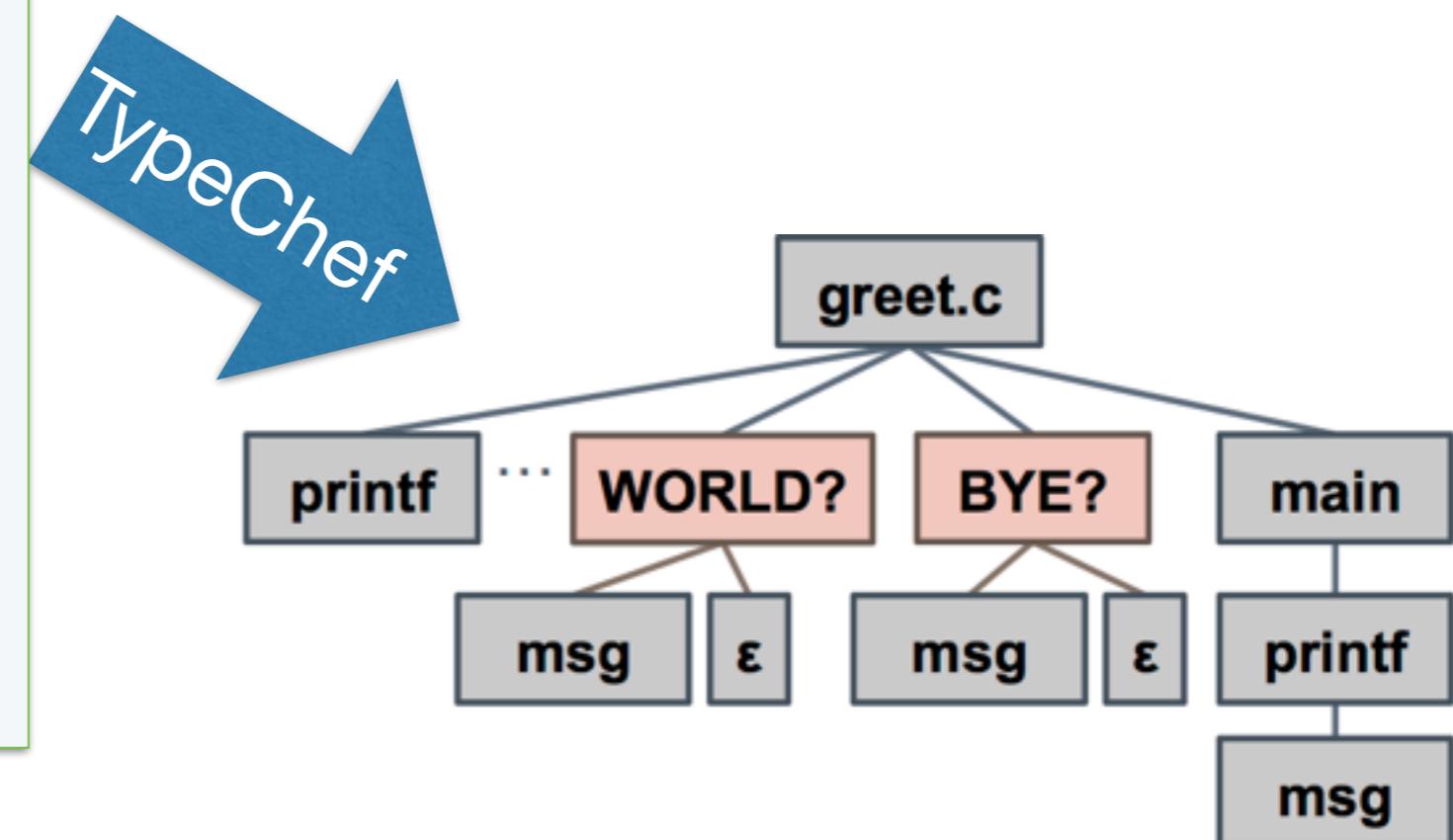
```
#include <stdio.h>

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

#ifndef BYE
char * msg = "Bye bye!\n";
#endif

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

<https://github.com/ckaestne/TypeChef>



AST with variability information

# Detecting Configuration Constraints (Underlying Analysis)

```
#include <stdio.h>

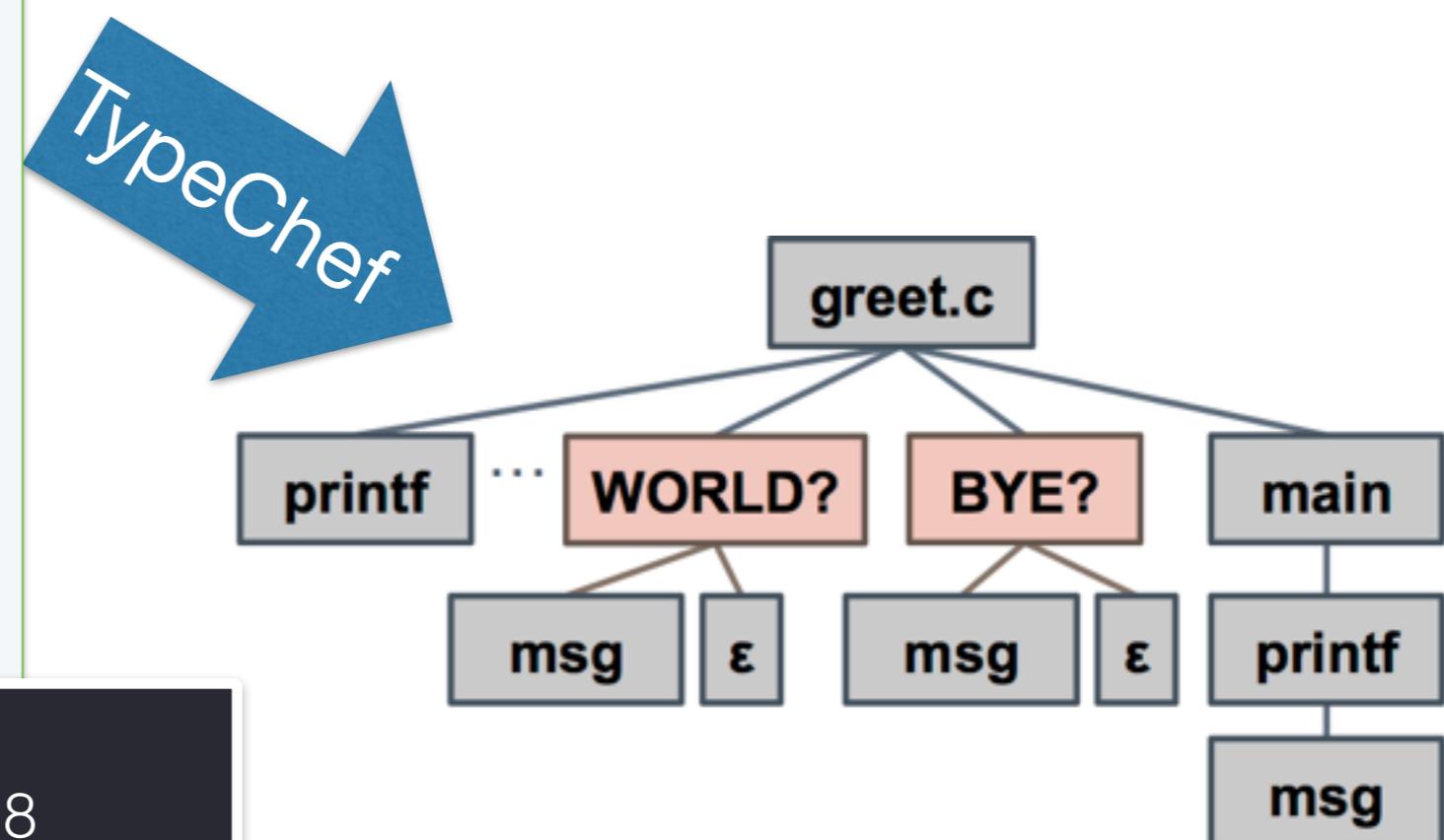
#ifndef WORLD
char * msg = "Hello World";
#endif

#ifndef BYE
char * msg = "Bye bye!\n";
#endif

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

Found 2 type errors:  
- [WORLD & BYE] file greet.c:7:8  
 redefinition of msg  
- [|WORLD & !BYE] file greet.c:11:8  
 msg undeclared

<https://github.com/ckaestne/TypeChef>



AST with variability information

# Feature Interactions



Weather



Smiley

# Feature Interactions



Weather



Smiley

**Weather Updates:**

Mostly cloudy today. It's currently 20°C

# Feature Interactions



Weather



Smiley

**Weather Updates:**

Mostly cloudy today. It's currently 20°C



# Feature Interactions



Weather



Smiley

**Weather Updates:**

Mostly cloudy today. It's currently 20°C



Weather



Smiley

# Feature Interactions



Weather



Smiley

**Weather Updates:**

Mostly cloudy today. It's currently 20°C



Weather



Smiley

**Weather Updates:**

Mostly cloudy today. It's currently [:Temperature]



# Feature Interactions



Weather



Smiley



Weather



Smiley

**Weather Updates:**

Mostly cloudy today. It's currently 20°C

Temperature not  
displaying properly

**Weather Updates:**

Mostly cloudy today. It's currently [:Temperature



# Feature Interactions



Weather



Smiley



Weather



Smiley

## Weather Updates:

Mostly cloudy today. It's currently 20°C

Temperature not  
displaying properly

## Weather Updates:

Mostly cloudy today. It's currently [:Temperature]

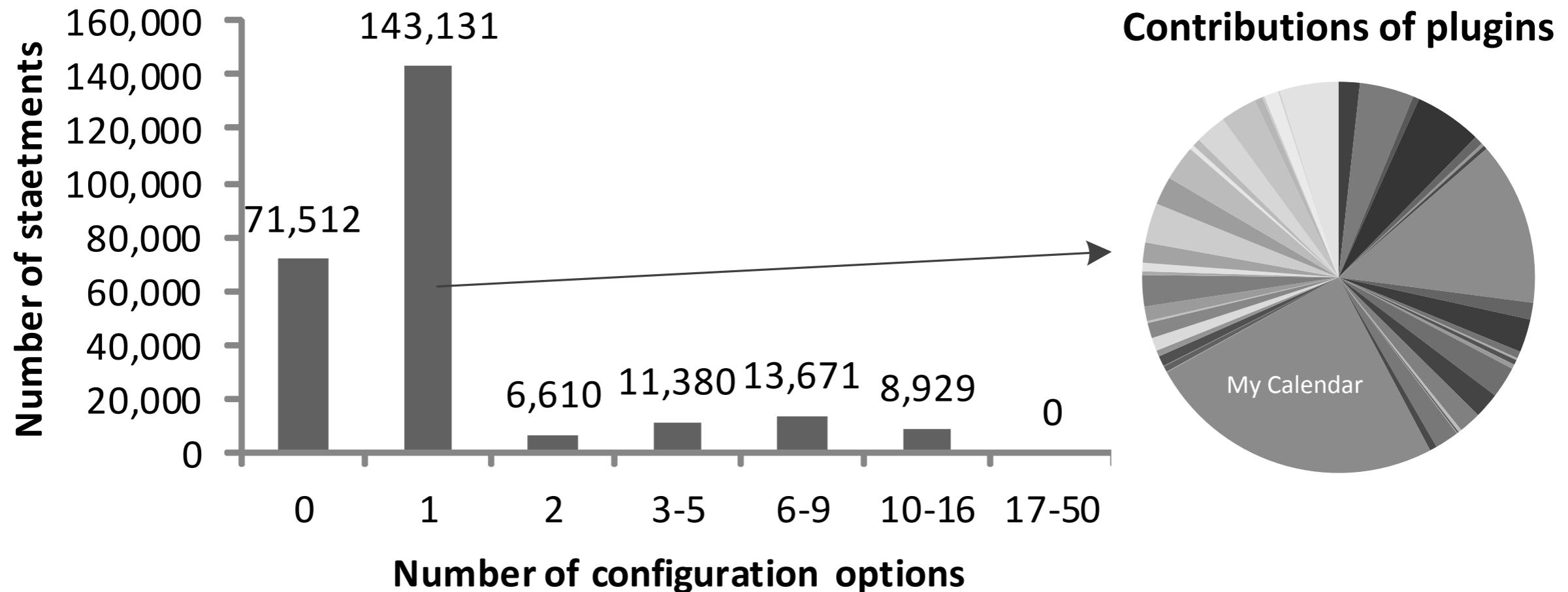


Weather replaces  
[:Temperature:] with value  
while Smiley replaces :] with  
a smiley face

# Detecting Feature Interactions



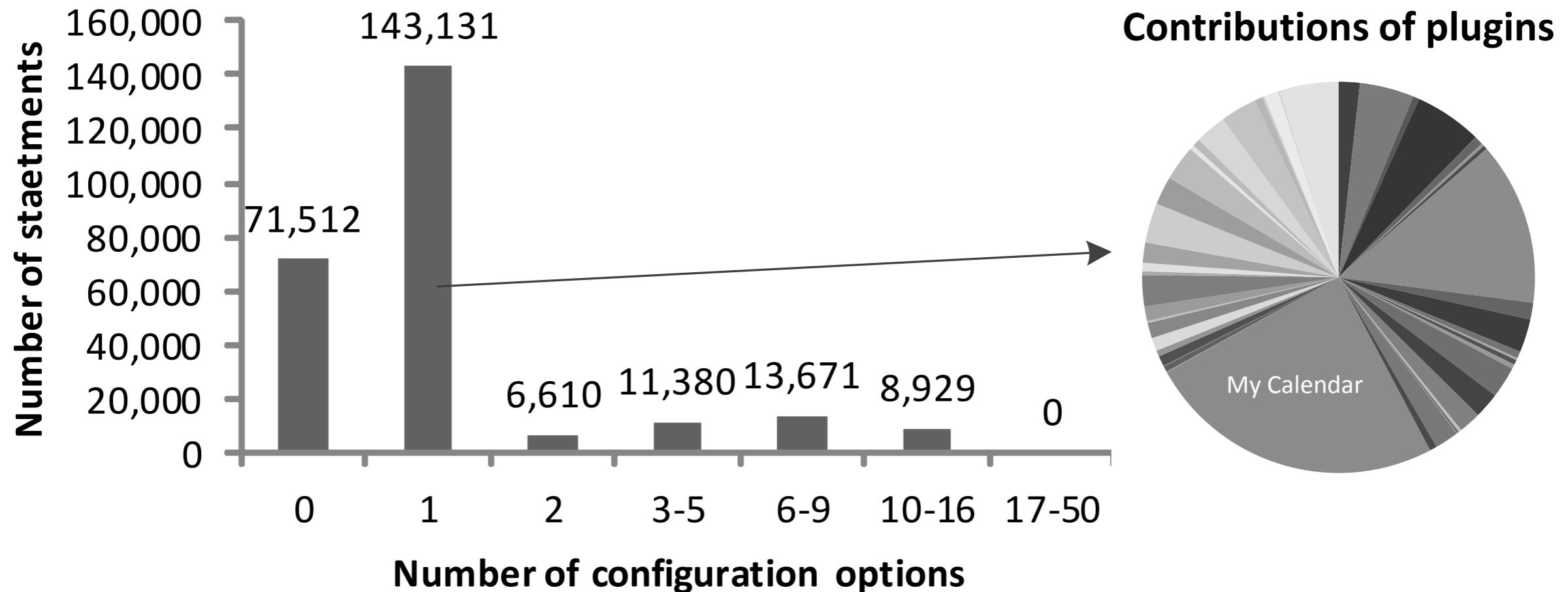
WORDPRESS



# Detecting Feature Interactions



WORDPRESS



Intended  
vs  
Unintended?

# Possible Project Topics

- Heuristics to detect *unintended* feature interactions
- Features vs options: nature of configurability in the Linux kernel
- Variability-aware refactoring during SPL migration
- Better code and/or build system comparison tools to identify commonality/variability?
- Does variability modeling of plug-in build dependencies provide any advantage?

# Short Overview of Software Product Lines

Sarah Nadi  
Software Technology Group  
[www.sarahnadi.org](http://www.sarahnadi.org)

