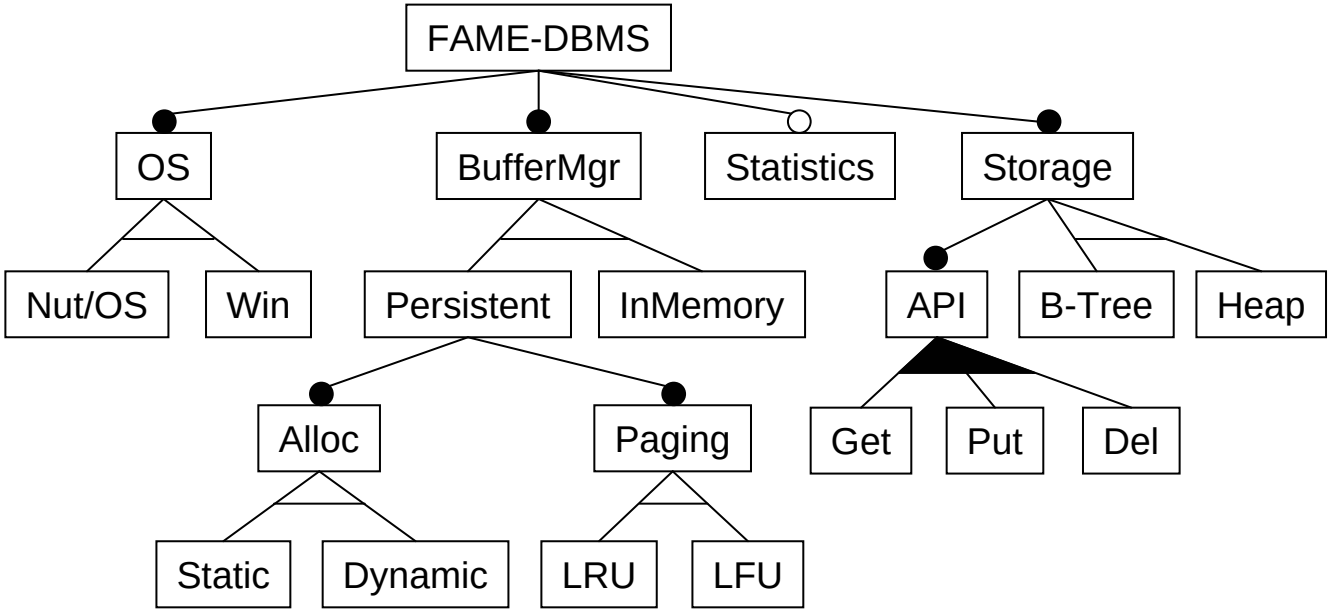
Safe Generation in Feature-Oriented Software Development

Sven Apel
Lehrstuhl für Programmierung
Universität Passau

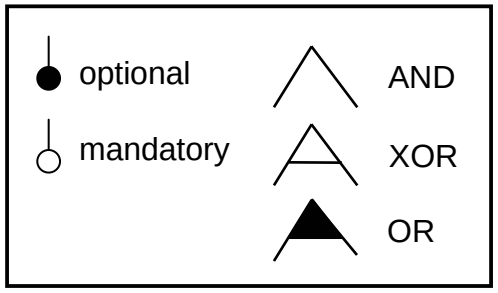
Department of
Informatics and
Mathematics

Programming
Group

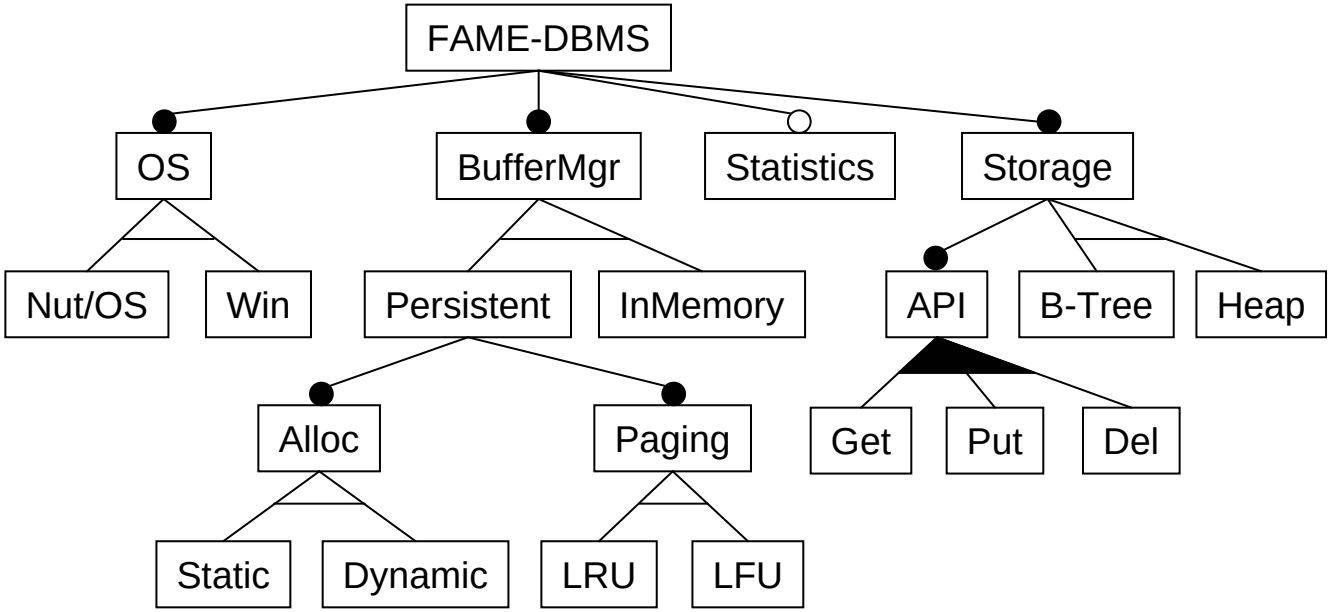
Software Product Line Engineering



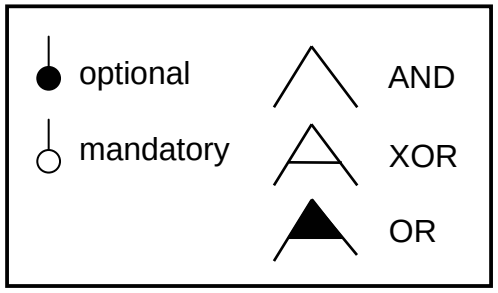
Domain Analysis



Software Product Line Engineering



Domain Analysis



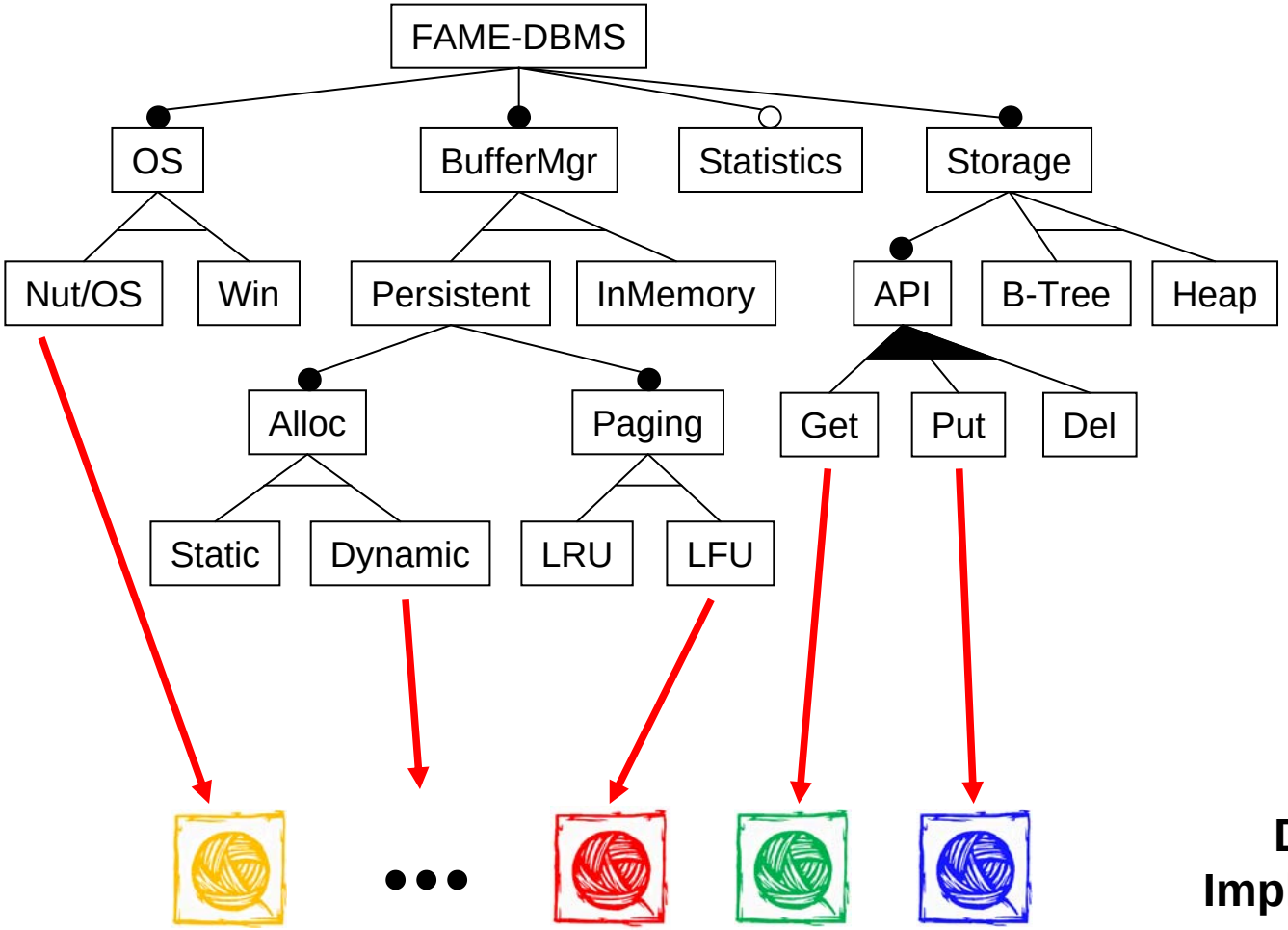
...

$OS \Rightarrow Nut/OS \oplus Win$

$API \Rightarrow Get \vee Put \vee Del$

...

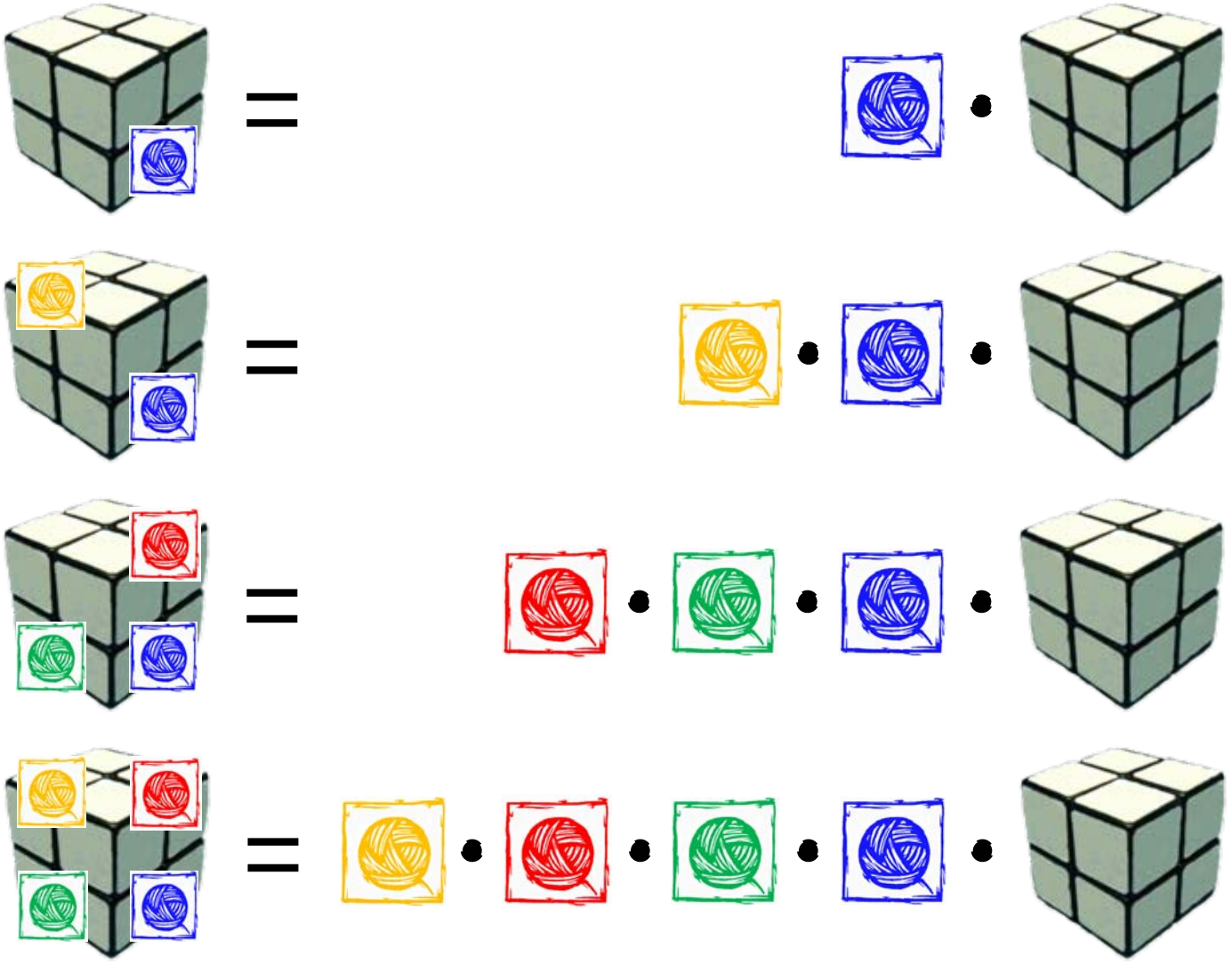
Feature-Oriented Software Development



Domain Analysis

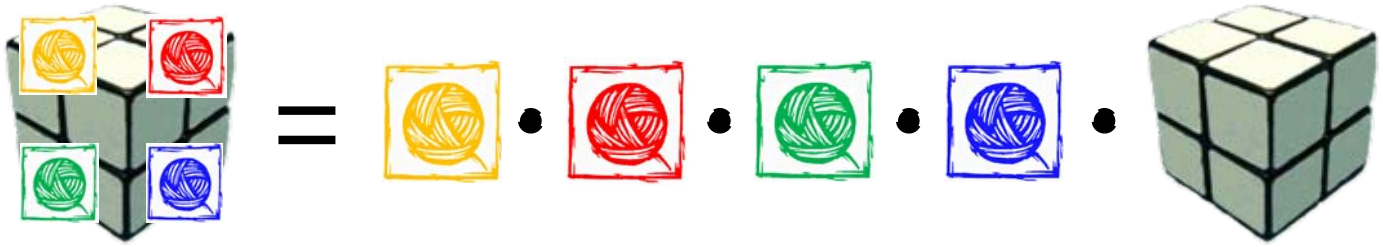
Design & Implementation

Compositional Approach



Safe Generation

- Is a generated product correct?
- Are all products of a product line correct?



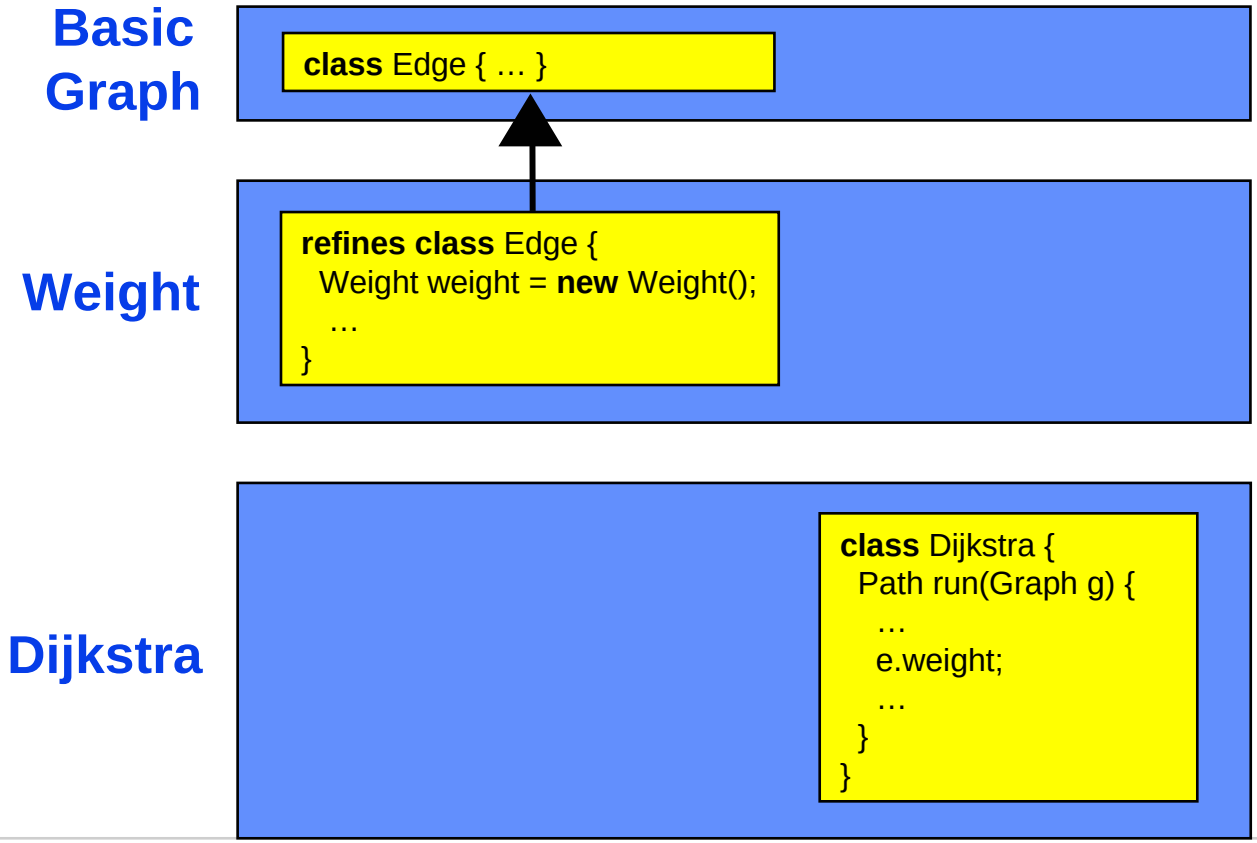
???

What Do We Mean by Correctness?

- Syntactic correctness 
- Type correctness
- Behavioral correctness

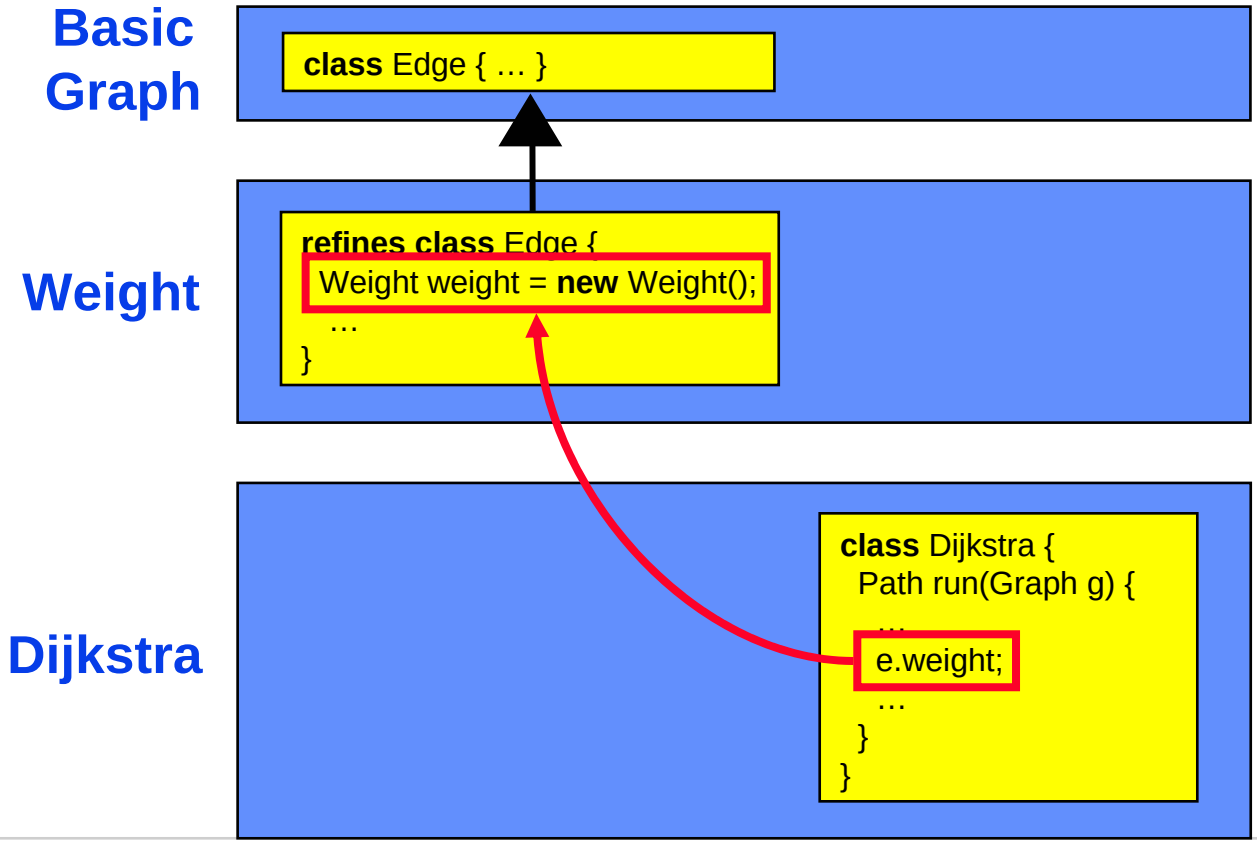
Type Correctness

- Do all valid feature selections produce products without type errors?



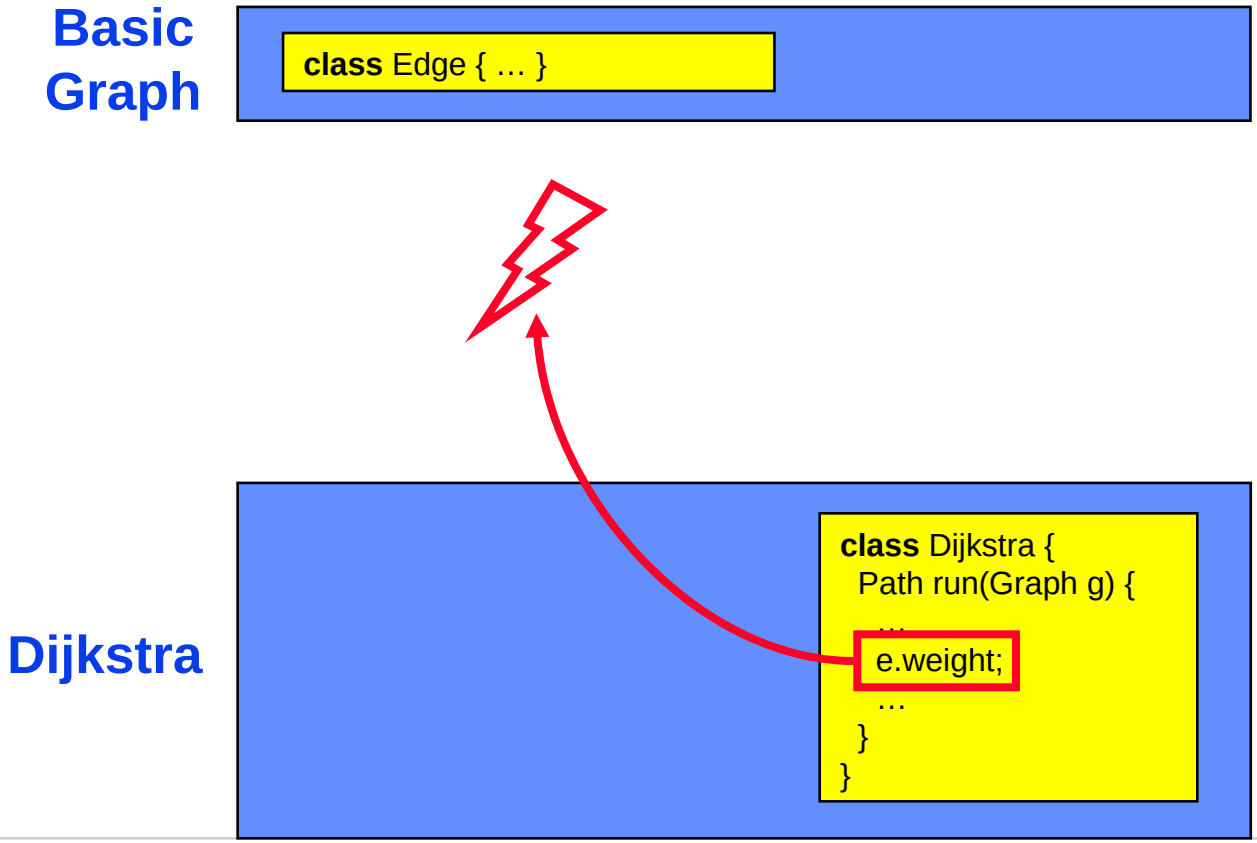
Type Correctness

- Do all valid feature selections produce products without type errors?



Type Correctness

- Do all valid feature selections produce products without type errors?

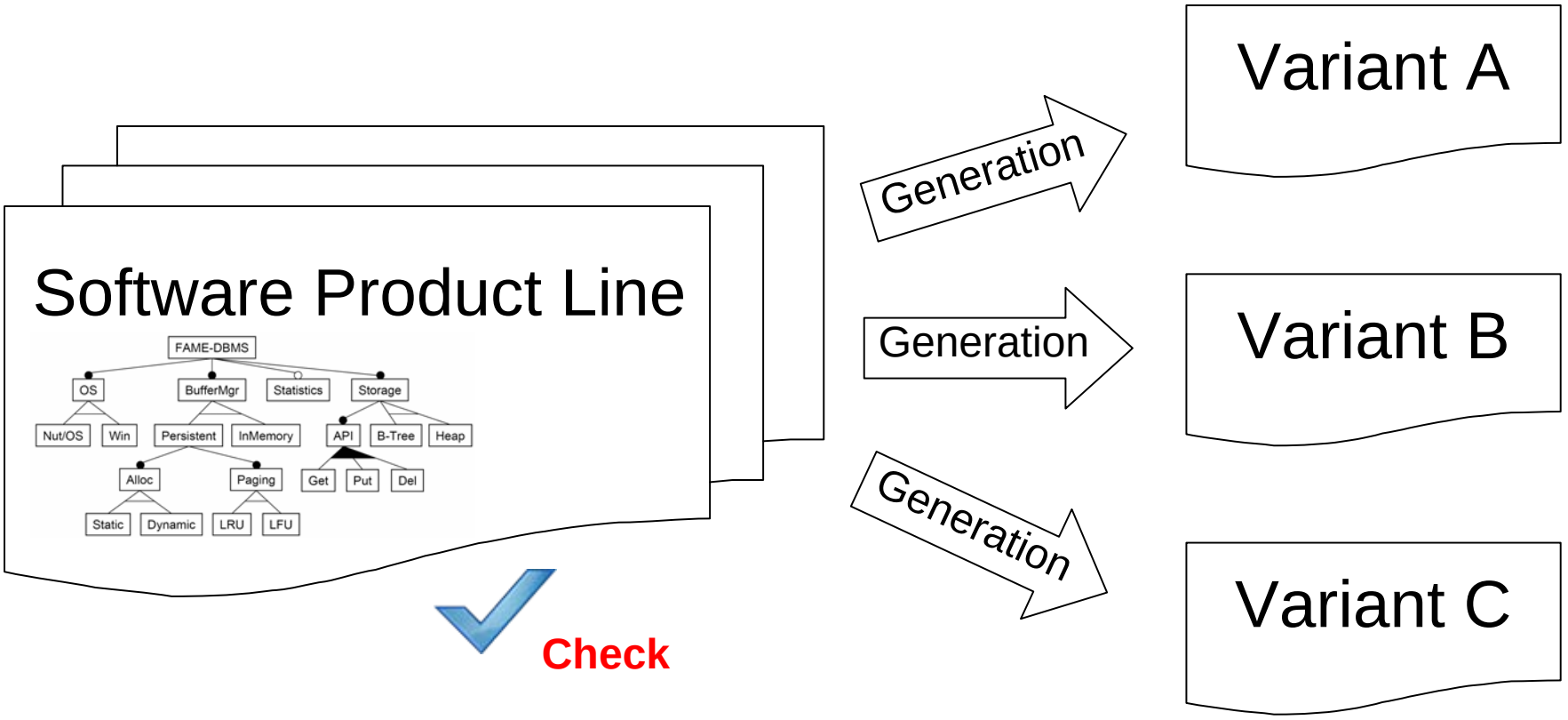


How Can We Check Type Correctness?

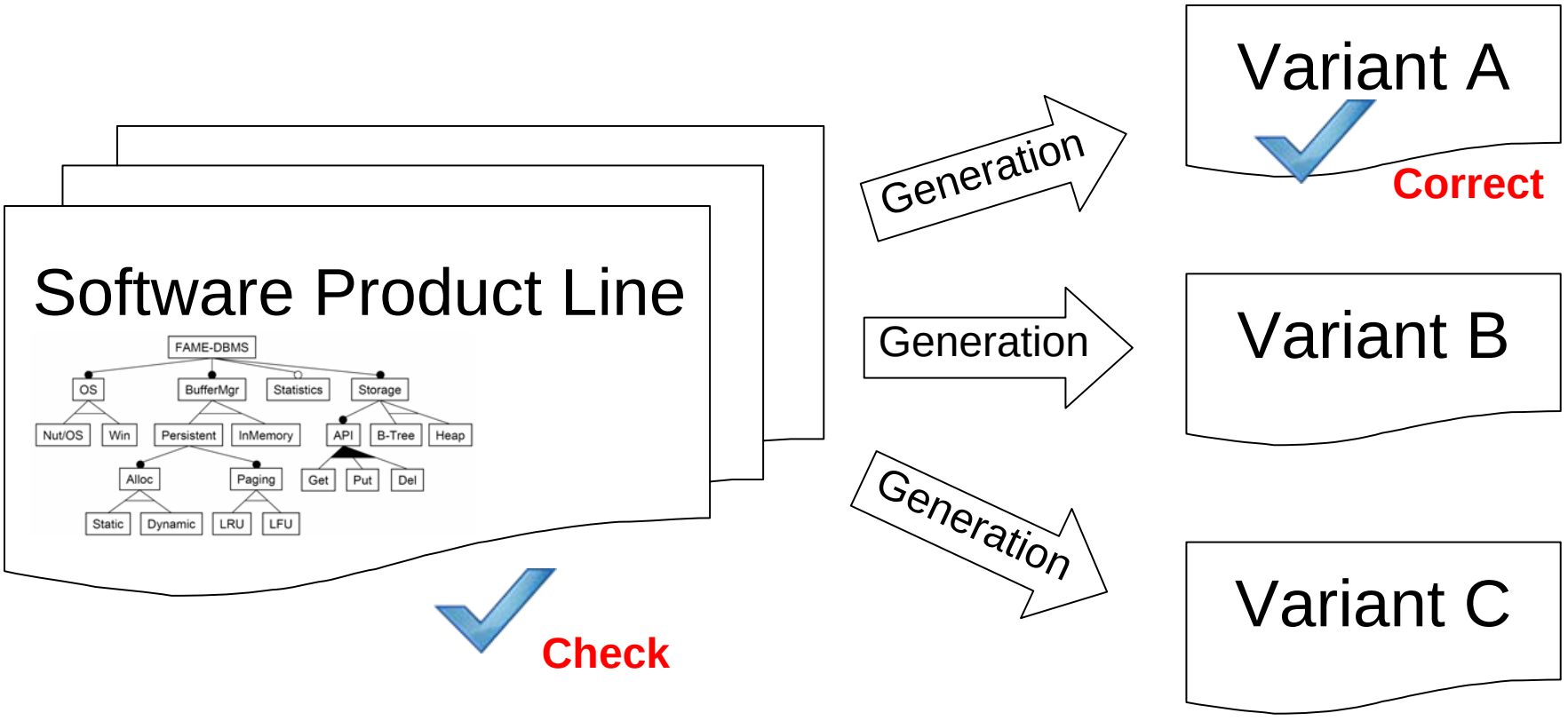
- Generate all valid programs and use an ordinary type system, e.g., the Java compiler for AHEAD
  - Not feasible for a growing number of features

- Check the product line's code base and use information on valid feature combinations ➔ product line type system

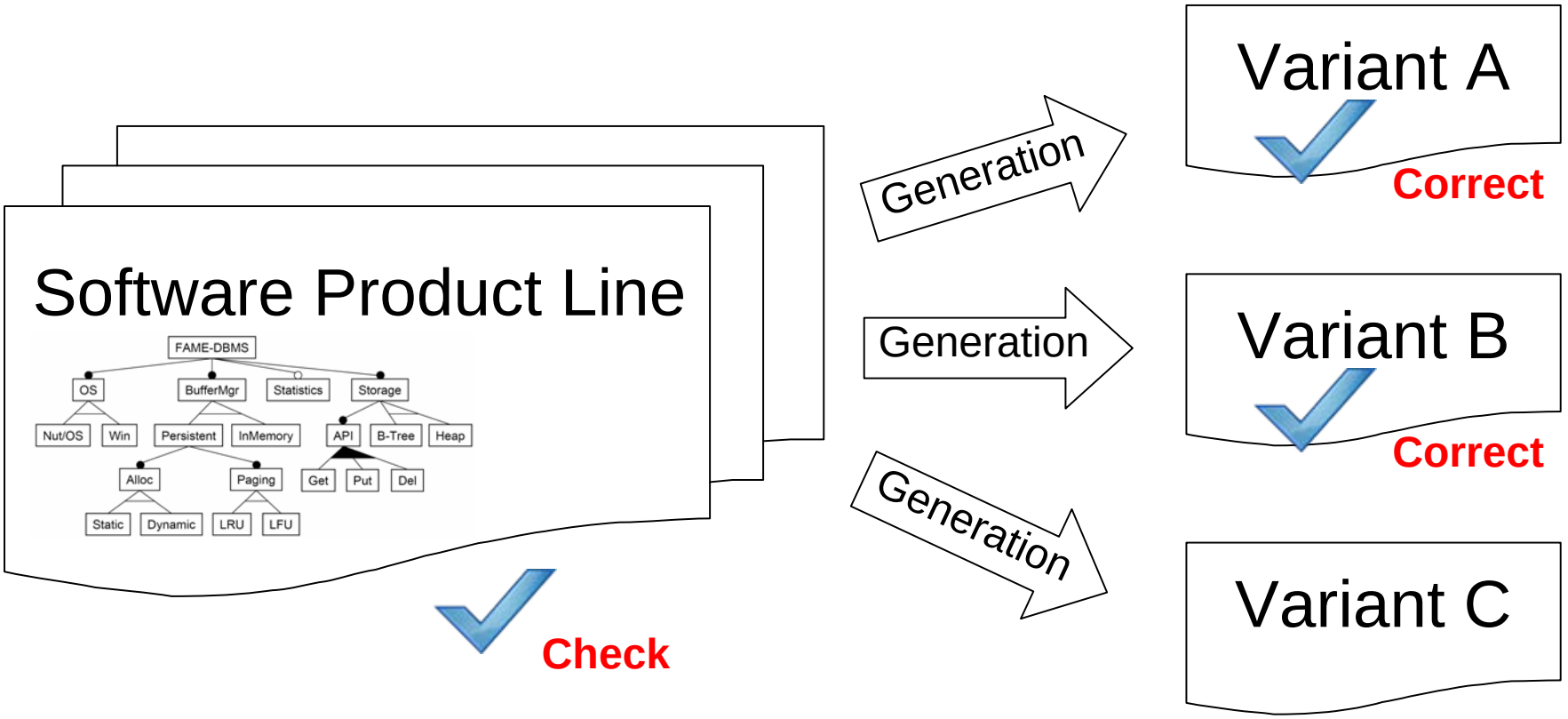
Product Line Type System



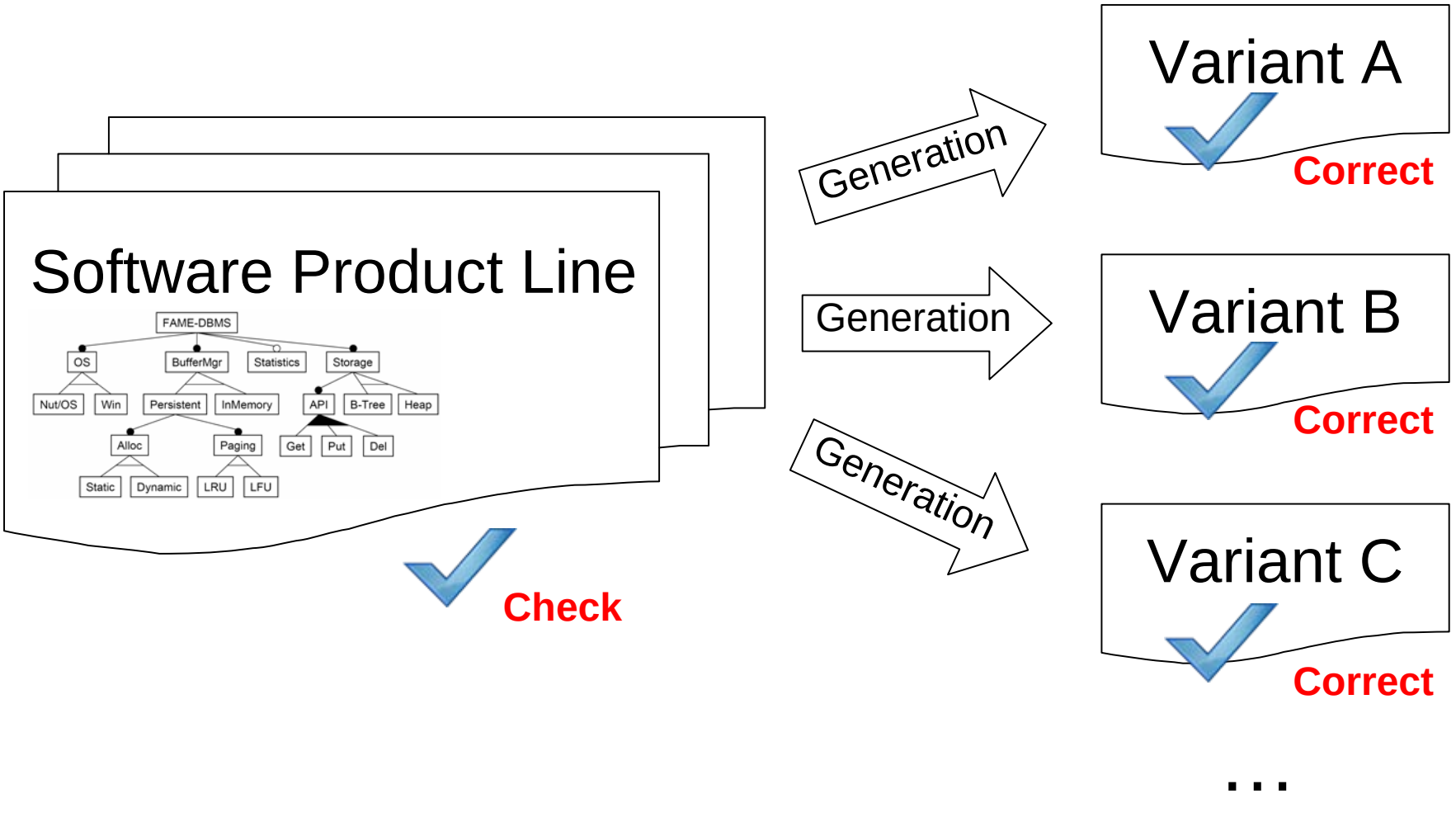
Product Line Type System



Product Line Type System



Product Line Type System



Product Line Type System – Details

- Enrich type rules with presence checks
  - Method invocation
  - Field access
  - Object instantiation
  - Class well-formedness
  - Method well-formedness
  - Subtyping
  - …

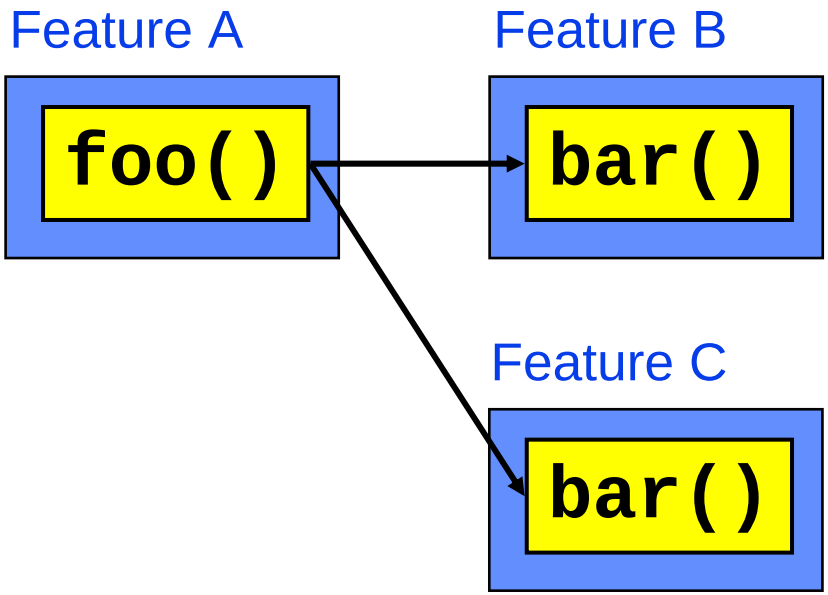
Pattern for additional type checks:

> Referenced element or type must be present in every valid variant in which also the feature of the referencing element or type is present.

Product Line Type System – Details

- Enrich type rules with presence checks

> Method **bar** must be present in every valid variant in which also method **foo** is present.
> ➔ solving a simple SAT problem.



Feature Model

$$A \Rightarrow B \oplus C$$

Product Line Type System – Formally

- Feature Featherweight Java
  [Apel et al., GPCE'08], [Apel et al., J. ASE'10]

Term typing $\boxed{\Gamma \vdash t : \overline{C} \dashv \Phi}$

$$\frac{x : C \in \Gamma}{\Gamma \vdash x : C \dashv \Phi} \quad (\text{T-VAR}_{PL})$$

$$\frac{\forall E \in \overline{E} : \mathit{validref}(\Phi, E.f) \quad \Gamma \vdash t_0 : \overline{E} \dashv \Phi \quad \mathit{fields}(\Phi, \overline{\mathit{last}(E)}) = \overline{\overline{\mathcal{F}, C\ f, \mathcal{G}}}}{\Gamma \vdash t_0.f : C_{11}, \ldots, C_{n1}, \ldots, C_{1m}, \ldots, C_{nm} \dashv \Phi} \quad (\text{T-FIELD}_{PL})$$

$$\frac{\forall E \in \overline{E} : \mathit{validref}(\Phi, E.m) \quad \forall \overline{C} \in \overline{\overline{C}}, \forall \overline{D} \in \overline{\overline{D}} \in \overline{\overline{D}} : \overline{C <: D} \dashv \Phi \quad \Gamma \vdash t_0 : \overline{E} \dashv \Phi \quad \Gamma \vdash \overline{t : C} \dashv \Phi \quad \mathit{mtype}(\Phi, m, \overline{\mathit{last}(E)}) = \overline{\overline{D \to B}}}{\Gamma \vdash t_0.m(\bar{t}) : B_{11}, \ldots, B_{n1}, \ldots, B_{1m}, \ldots, B_{nm} \dashv \Phi} \quad (\text{T-INVK}_{PL})$$

$$\frac{\mathit{validref}(\Phi, C) \quad \forall \overline{D\ g} \in \overline{\mathcal{F}}, \forall \overline{C} \in \overline{\overline{C}} : \overline{C <: D} \dashv \Phi \quad \Gamma \vdash \overline{t : C} \dashv \Phi \quad \mathit{fields}(\Phi, \mathit{last}(C)) = \overline{\mathcal{F}} \quad @ \notin \overline{\mathcal{F}}}{\Gamma \vdash \text{new } C(\bar{t}) : C \dashv \Phi} \quad (\text{T-NEW}_{PL})$$

$$\frac{\mathit{validref}(\Phi, C) \quad \Gamma \vdash t_0 : \overline{E} \dashv \Phi \quad \forall E \in \overline{E} : (E <: C \dashv \Phi \vee C <: E \dashv \Phi)}{\Gamma \vdash (C)t_0 : C \dashv \Phi} \quad (\text{T-UDCAST}_{PL})$$

$$\frac{\mathit{validref}(\Phi, C) \quad \mathit{stupid\ warning} \quad \Gamma \vdash t_0 : \overline{E} \dashv \Phi \quad \exists E \in \overline{E} : (C \not<: E \dashv \Phi \wedge E \not<: C \dashv \Phi)}{\Gamma \vdash (C)t_0 : C \dashv \Phi} \quad (\text{T-SCAST}_{PL})$$

Method typing $\boxed{M\ \text{OK} \dashv \Phi.C}$

$$\frac{\overline{x : B}, \text{this} : C \vdash t_0 : \overline{E} \dashv \Phi \quad \forall E \in \overline{E} : E <: B_0 \dashv \Phi \quad \mathit{validref}(\Phi, \overline{B}) \quad \mathit{introduce}(\Phi, m, \mathit{last}(D)) \quad CT(\Phi.C) = \text{class C extends D } \{ \overline{C\ f}; \overline{M} \}}{B_0\ m(\overline{B\ x})\ \{ \text{return } t_0; \}\ \text{OK} \dashv \Phi.C}$$

$$\frac{\overline{x : B}, \text{this} : C \vdash t_0 : \overline{E} \dashv \Phi \quad \forall E \in \overline{E} : E <: B_0 \dashv \Phi \quad \mathit{validref}(\Phi, \overline{B}) \quad \mathit{override}(\Phi, m, \mathit{last}(D), \overline{B \to B_0}) \quad CT(\Phi.C) = \text{class C extends D } \{ \overline{C\ f}; \overline{M} \}}{\text{overrides } B_0\ m(\overline{B\ x})\ \{ \text{return } t_0; \}\ \text{OK} \dashv \Phi.C}$$

$$\frac{\overline{x : B}, \text{this} : C \vdash t_0 : \overline{E} \dashv \Phi \quad \forall E \in \overline{E} : E <: B_0 \dashv \Phi \quad \mathit{validref}(\Phi, \overline{B}) \quad \mathit{introduce}(\Phi, m, \mathit{pred}(\Phi.C)) \quad CT(\Phi.C) = \text{refines class C } \{ \overline{C\ f}; \overline{M} \}}{B_0\ m(\overline{B\ x})\ \{ \text{return } t_0; \}\ \text{OK} \dashv \Phi.C}$$

$$\frac{\overline{x : B}, \text{this} : C \vdash t_0 : \overline{E} \dashv \Phi \quad \forall E \in \overline{E} : E <: B_0 \dashv \Phi \quad \mathit{validref}(\Phi, \overline{B}) \quad \mathit{override}(\Phi, m, \mathit{pred}(\Phi.C), \overline{B \to B_0}) \quad CT(\Phi.C) = \text{refines class C } \{ \overline{C\ f}; \overline{M} \}}{\text{overrides } B_0\ m(\overline{B\ x})\ \{ \text{return } t_0; \}\ \text{OK} \dashv \Phi.C}$$

Class typing $\boxed{L\ \text{OK} \dashv \Phi}$

$$\frac{\forall f \in \bar{f} : \mathit{introduce}(\Phi, f, \mathit{last}(D)) \quad \mathit{validref}(\Phi, D) \quad \mathit{validref}(\Phi, \overline{C}) \quad \mathit{introduce}(\Phi, \Phi.C) \quad \overline{M}\ \text{OK} \dashv \Phi.C}{\text{class C extends D } \{ \overline{C\ f}; \overline{M} \}\ \text{OK} \dashv \Phi}$$

Refinement typing $\boxed{R\ \text{OK} \dashv \Phi}$

$$\frac{\forall f \in \bar{f} : \mathit{introduce}(\Phi, f, \mathit{pred}(\Phi.C)) \quad \mathit{validref}(\Phi, \overline{C}) \quad \mathit{refine}(\Phi, \Phi.C) \quad \overline{M}\ \text{OK} \dashv \Phi.C}{\text{refines class C } \{ \overline{C\ f}; \overline{M} \}\ \text{OK} \dashv \Phi}$$

Product Line Type System – Formally

- Feature Featherweight Java
  [Apel et al., GPCE'08], [Apel et al., J. ASE'10]

- Proved theorems of product line type system
  - **Soundness**: all programs of a well-typed product line are well-typed
  - **Completeness**: for a set of well-typed programs there is a well-typed product line

- Prototype implementation in Haskell

What Do We Mean by Correctness?

- Syntactic correctness 
- Type correctness 
- Behavioral correctness

Behavioral Correctness

CorrectSum

```
…
int sum(int a, int b) {
  return a + b;
}
…
```

WrongSum

```
…
int sum(int a, int b) {
  return a - b;
}
…
```

Feature Model

$$UseSum \Rightarrow CorrectSum \oplus WrongSum$$

UseSum

```
… int c = sum(a, b); …
```

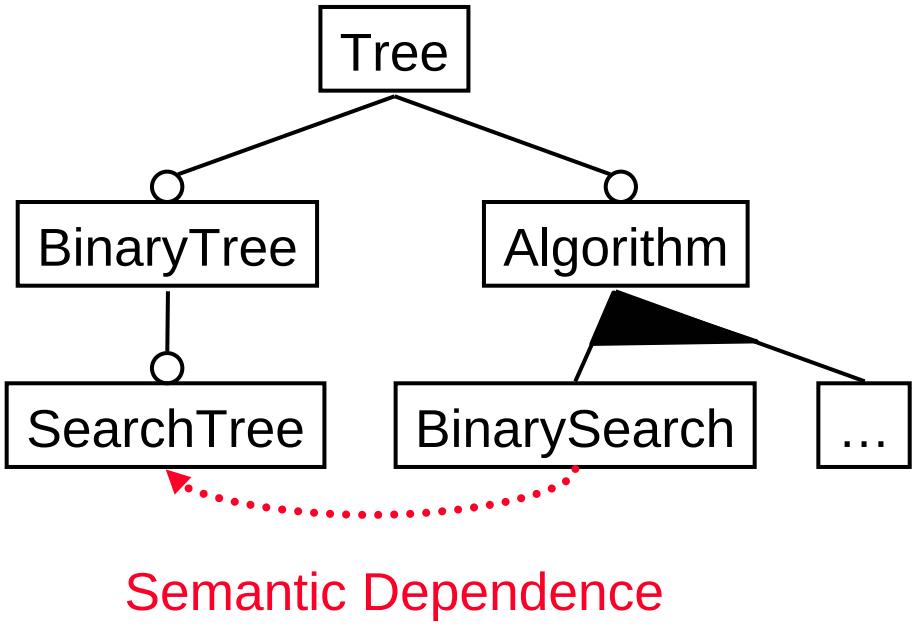
Potential Causes of Behavioral Errors

- Semantic dependence
- Feature interaction

Semantic Dependence

- A feature depends on the presence of another feature without referring to that feature

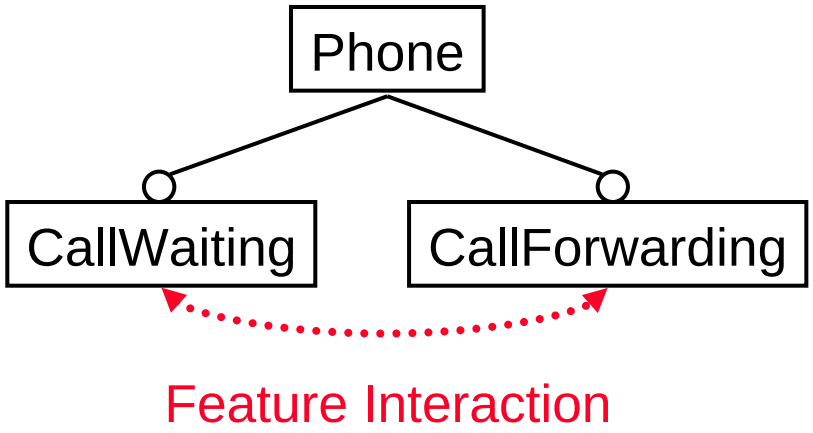
  ➔ there may be no syntactic relationship



Feature Interaction

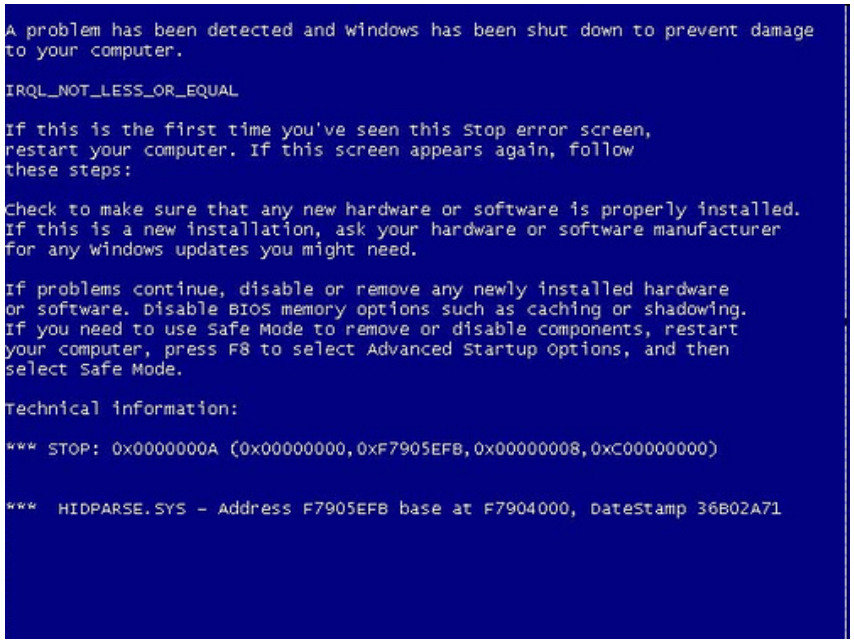
- Two features may work correct in isolation but exhibit erroneous behavior in combination

  ➔ there may be no syntactic relationships



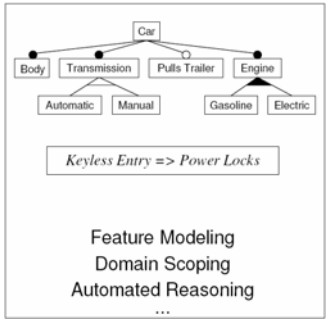
What is the Problem?

- Undocumented dependences and interactions
  - Type system reports no errors
  - Compiler produces a runnable program
  - Errors may arise only after a long operation time

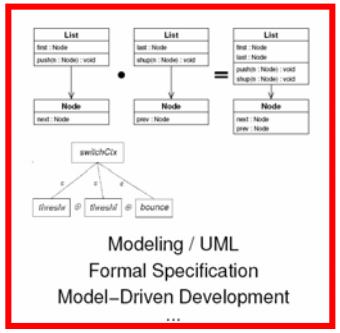


Method of Resolution

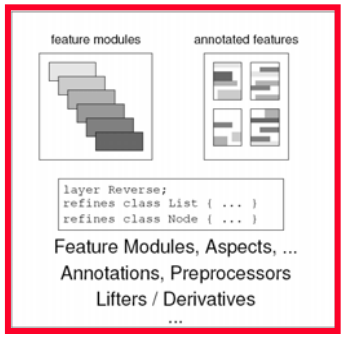
- Feature-oriented specification and design
- Feature-oriented verification



Domain Analysis



Domain Design & Specification



Implementation

Product Derivation

Feature-Oriented Design with FeatureAlloy

- Based on Alloy [Jackson, TOSEM 2002]
  - Textual, general purpose modeling language

```
module Graph
one sig Graph {
  nodes : set Node
} {
  Node in nodes
}
sig Node {
  inEdges : set Edge, outEdges : set Edge, edges = inEdges + outEdges
}
sig Edge {
  src : one Node, dest : one Node
}
fact prevNext {
  all n: Node, e: Edge |
    (n in e.src <=> e in n.outEdges) && (n in e.dest <=> e in n.inEdges)
}
```

Feature-Oriented Design with FeatureAlloy

- Based on Alloy [Jackson, TOSEM 2002]
  - Textual, general purpose modeling language
  - Automatic reasoning facilities ➔ Alloy Analyzer

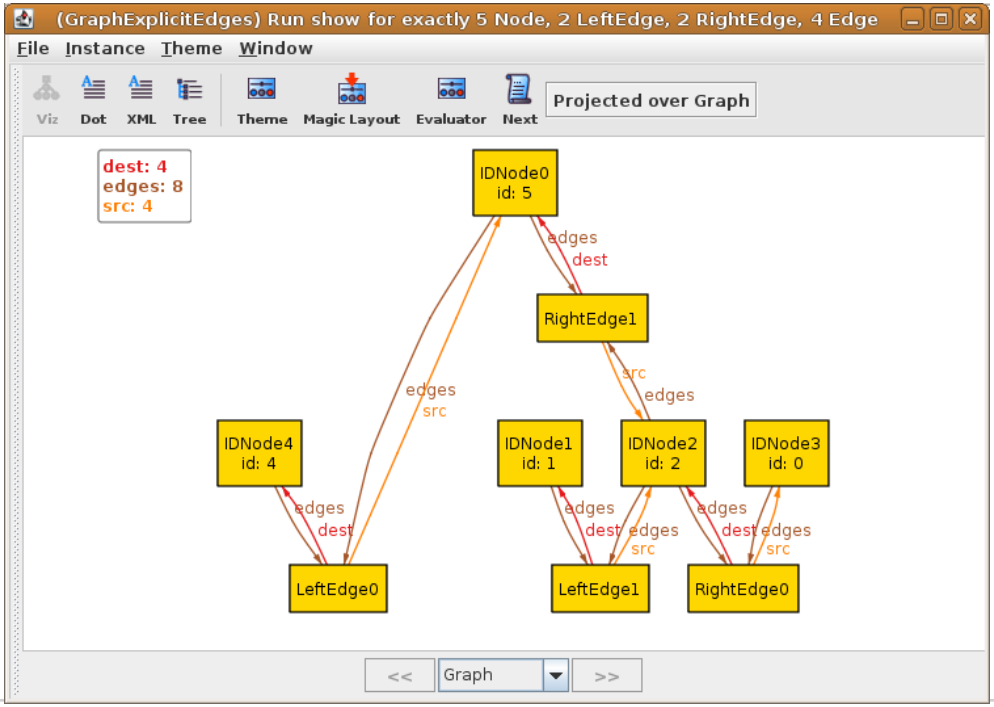
```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }

pred noDoubleEdges {
  all e, e': Edge | e != e' => e.(src + dest) != e'.(src + dest)
}
run noDoubleEdges for 5
```

```
assert hasNoDoubleEdges {
  !noDoubleEdges
}
check hasNoDoubleEdges for 5
```

Feature-Oriented Design with FeatureAlloy

- Based on Alloy [Jackson, TOSEM 2002]
  - Textual, general purpose modeling language
  - Automatic reasoning facilities ➔ Alloy Analyzer
  - Visualization facilities ➔ Alloy Analyzer



Model Features in Separate Units – Graph Example

Graph

```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }
```

Tree

```
refines module Graph
pred isConnected {
  some n: Node |
    (Graph.nodes = n) || (Graph.nodes = n.^(edges.(src + dest)))
}
pred noCycles {
  all n: Node | n not in (n.^(outEdges.dest) + n.^(inEdges.src))
}
pred loneParent {
  all n: Node | lone n.inEdges
}
fact isTree {
  noDoubleEdges && isConnected && noCycles && loneParent
}
```

Feature Composition

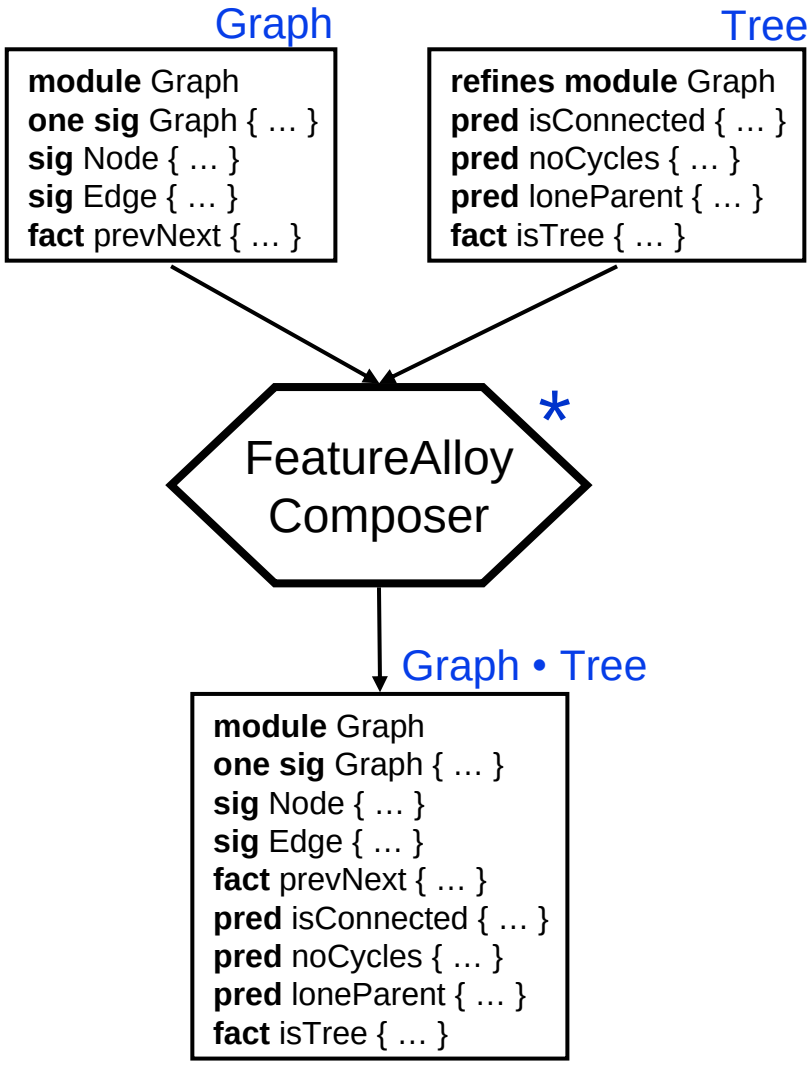
Graph

```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }
```

Tree

```
refines module Graph
pred isConnected { … }
pred noCycles { … }
pred loneParent { … }
fact isTree { … }
```

Feature Composition



* Plug-in of FeatureHouse

Feature Composition

Graph

```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }
```

Tree

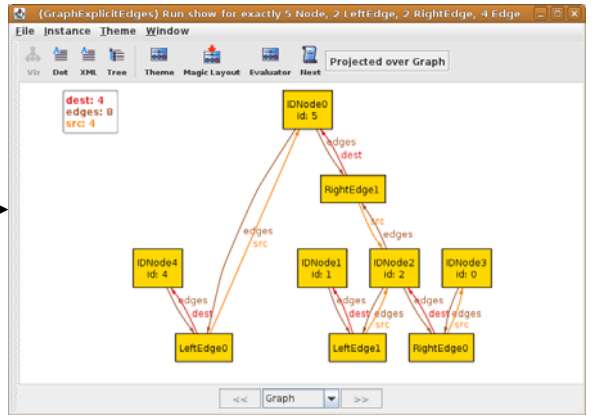
```
refines module Graph
pred isConnected { … }
pred noCycles { … }
pred loneParent { … }
fact isTree { … }
```

FeatureAlloy Composer

Graph • Tree

```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }
pred isConnected { … }
pred noCycles { … }
pred loneParent { … }
fact isTree { … }
```

Alloy Analyzer



Semantic Dependences – Graph Example

Graph

```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }
```

BinaryTree

```
refines module Graph
fact binaryTree {
  all n: Node | #n.outEdges =< 2
}
```

Tree

```
refines module Graph
pred isConnected { … }
pred noCycles { … }
pred loneParent { … }
fact isTree { … }
```

Semantic Dependence

Semantic Dependences – Graph Example

Graph

```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }
```

Tree

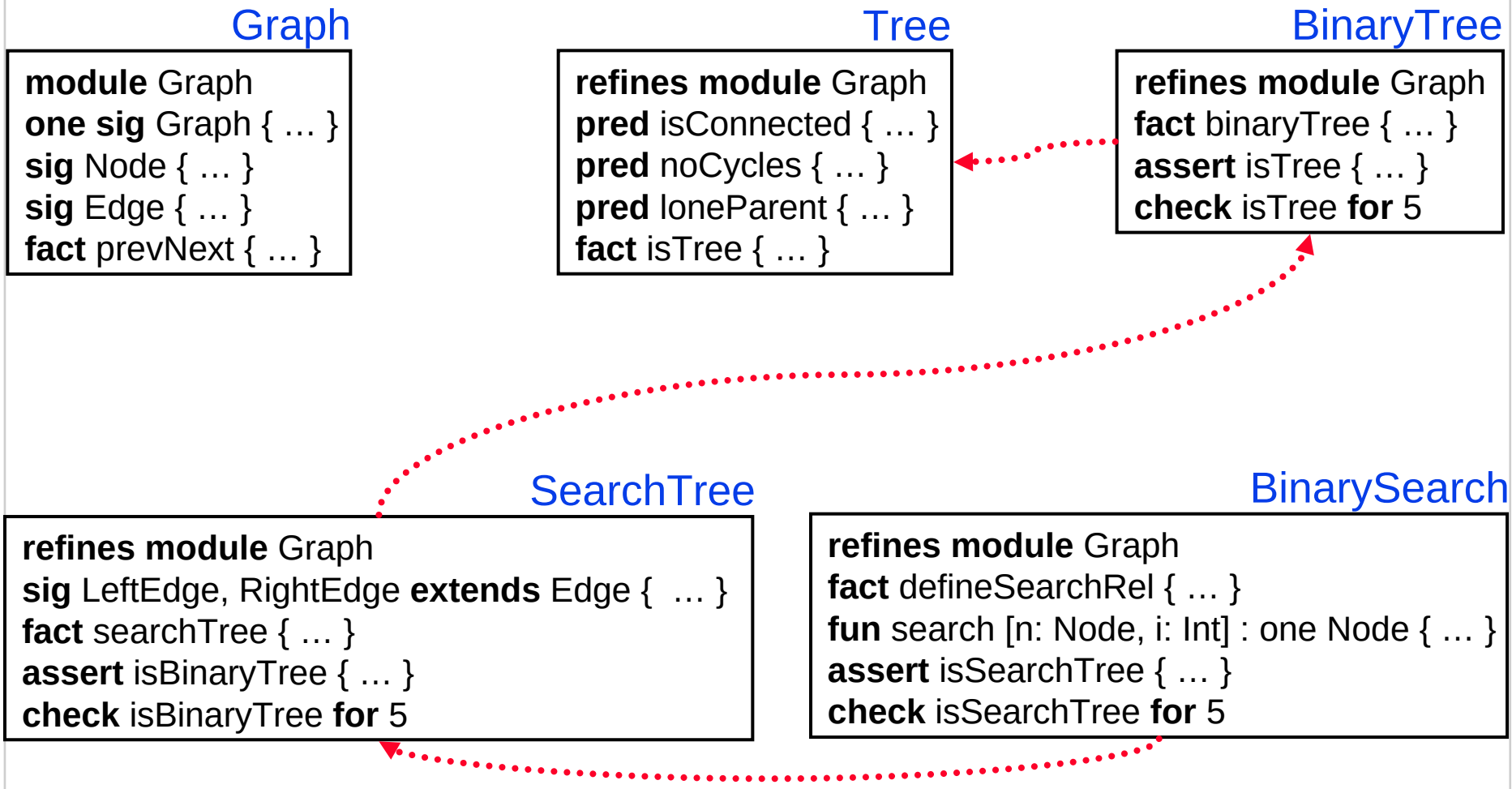
```
refines module Graph
pred isConnected { … }
pred noCycles { … }
pred loneParent { … }
fact isTree { … }
```

BinaryTree

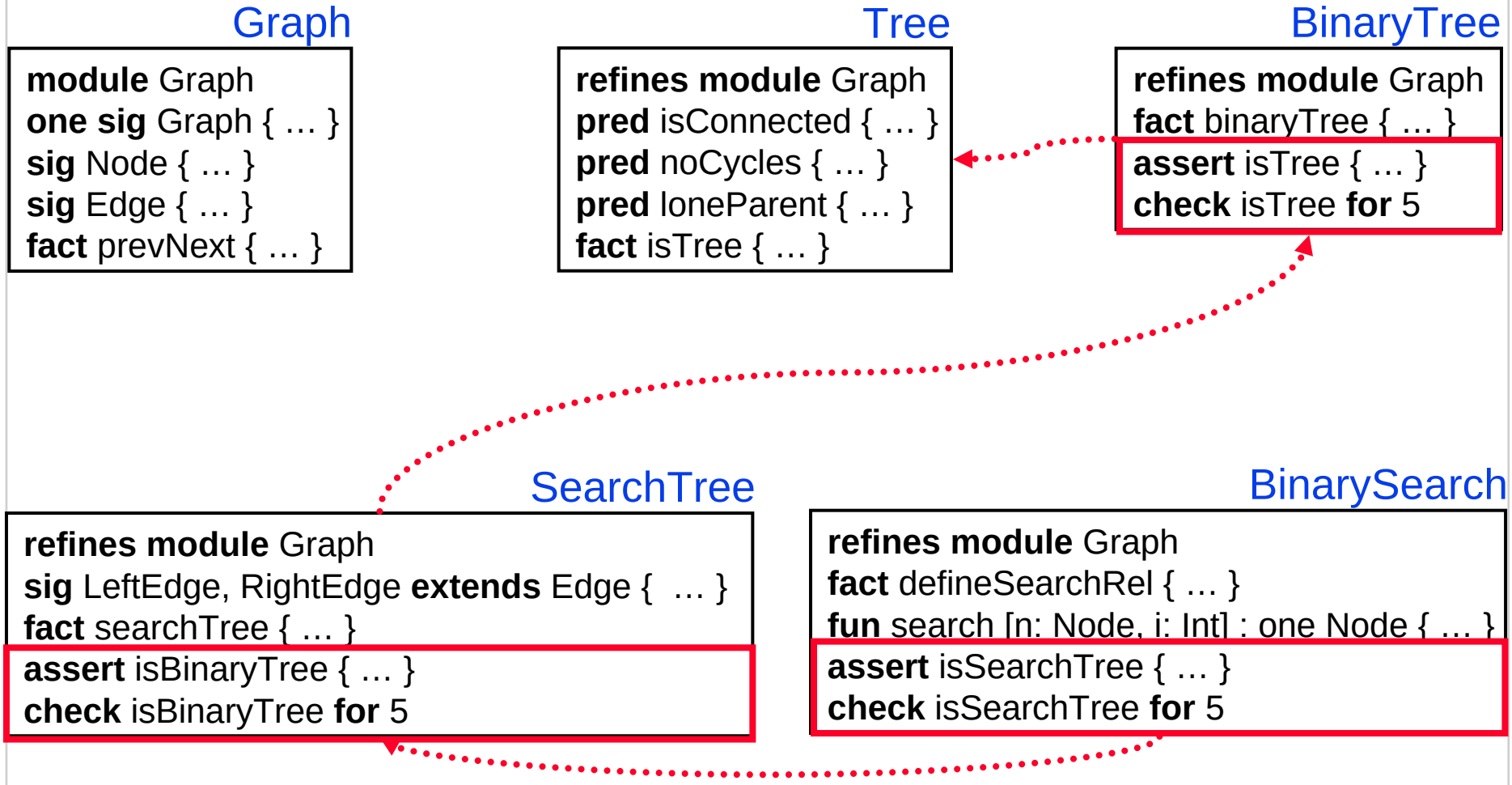
```
refines module Graph
fact binaryTree {
  all n: Node | #n.outEdges =< 2
}
assert isTree {
  all e, e': Edge | e != e' => e.(src + dest) != e'.(src + dest)
  some n: Node |
    (Graph.nodes = n) || (Graph.nodes = n.^(edges.(src + dest)))
  all n: Node | n not in (n.^(outEdges.dest) + n.^(inEdges.src))
  all n: Node | lone n.inEdges
}
check isTree for 5
```

Semantic Dependence

Semantic Dependences – Graph Example



Semantic Dependences – Graph Example



Detect Unresolved Dependences

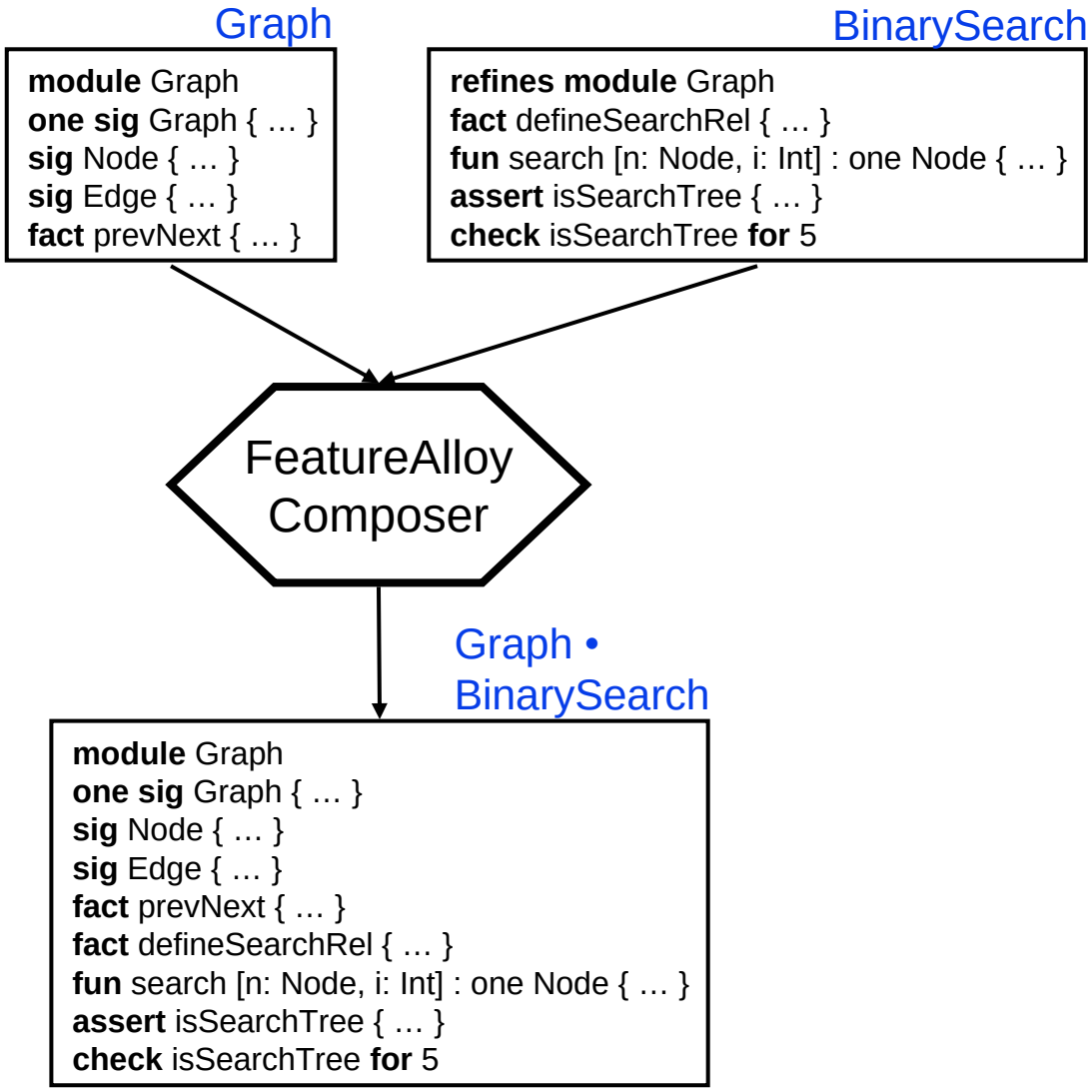
Graph

```
module Graph
one sig Graph { … }
sig Node { … }
sig Edge { … }
fact prevNext { … }
```

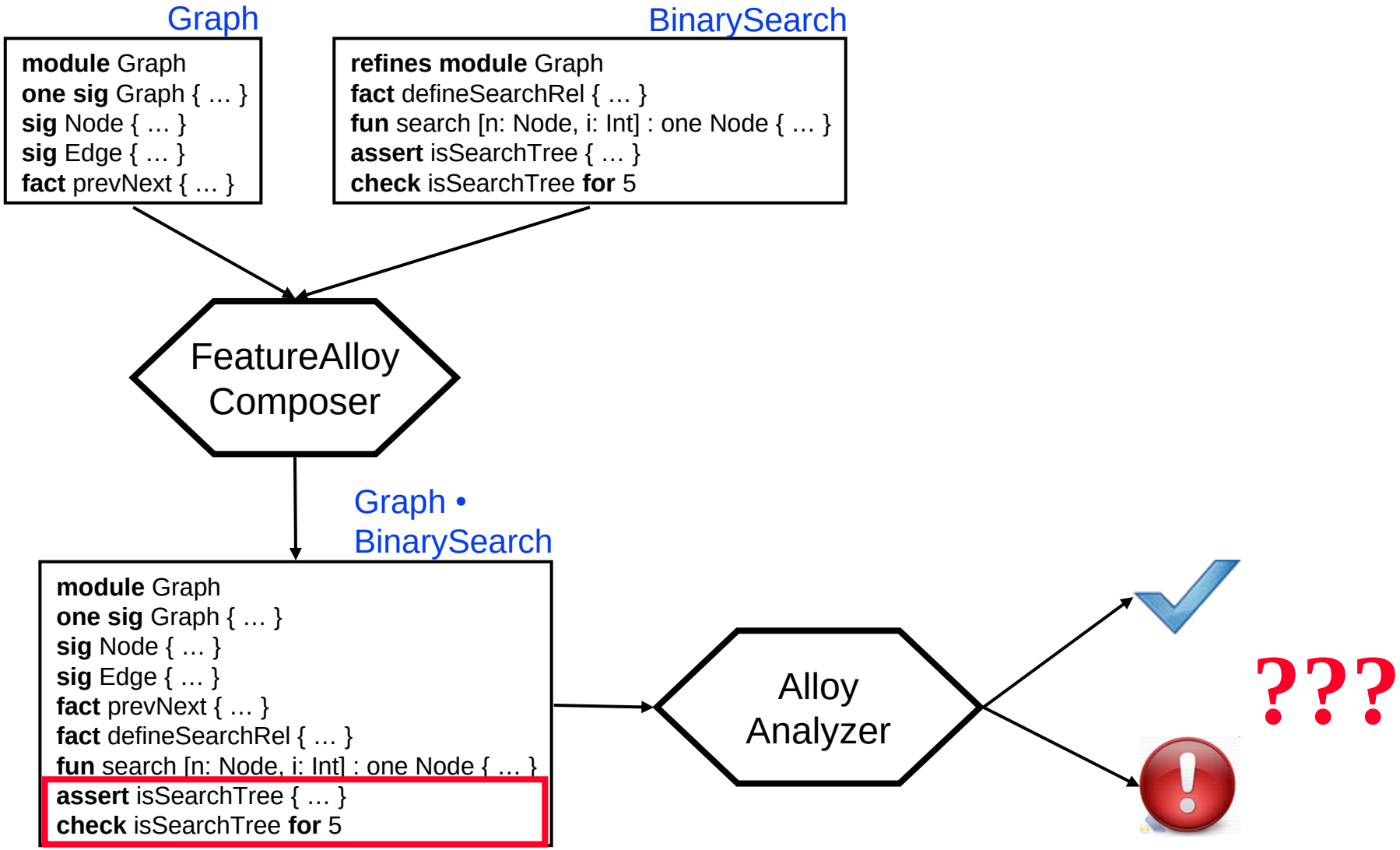
BinarySearch

```
refines module Graph
fact defineSearchRel { … }
fun search [n: Node, i: Int] : one Node { … }
assert isSearchTree { … }
check isSearchTree for 5
```

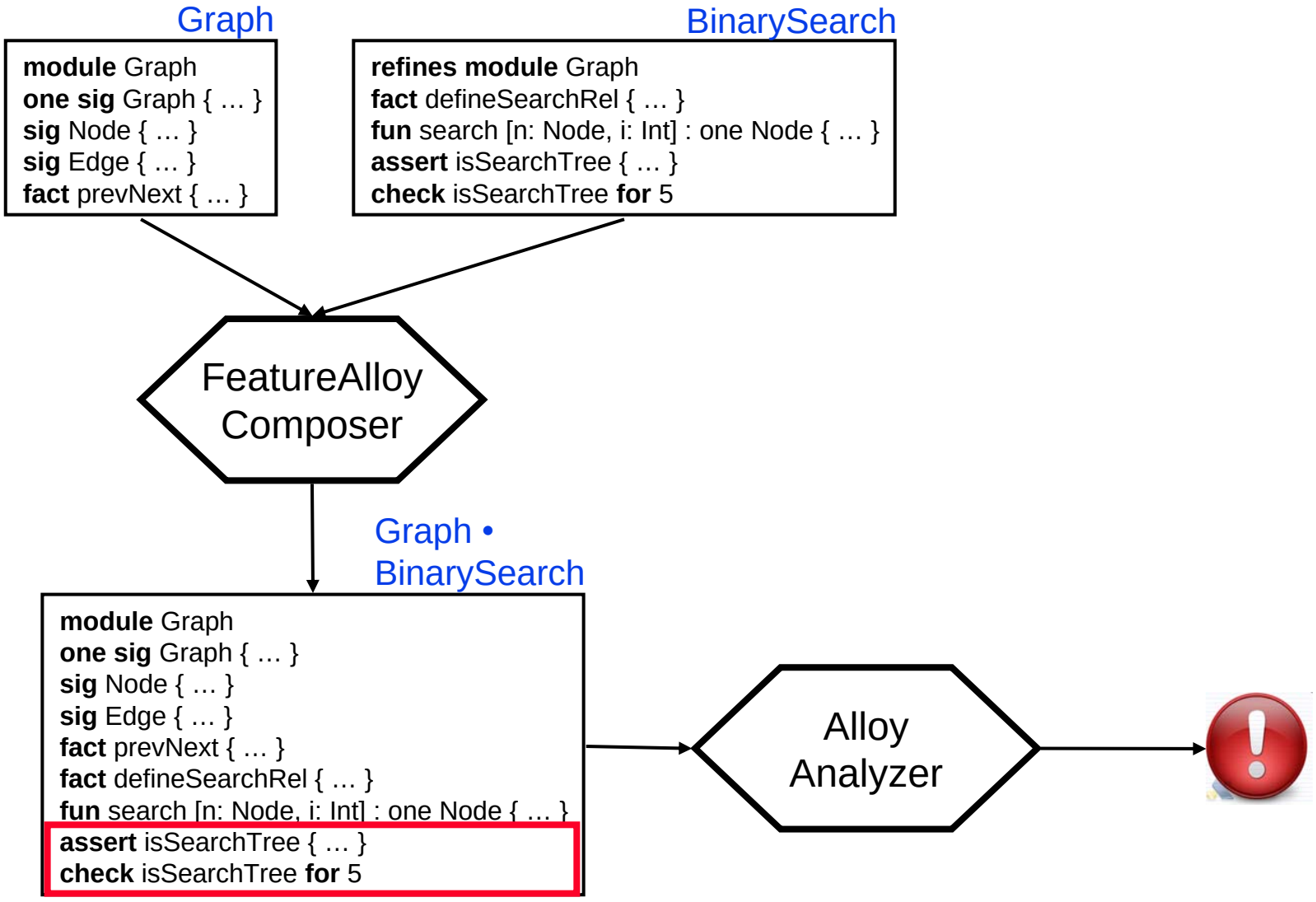
Detect Unresolved Dependences



Detect Unresolved Dependences



Detect Unresolved Dependences



Feature Interactions – Phone Example

BasicPhone

```
module Phone
sig Phone { ... }
abstract sig State {}
one sig Idle, Busy extends State {} ...
pred incomingCall [in: Call, disj p, p': Phone] { ... }
```

CallForwarding

```
refines module Phone
refines sig Phone {
  forward: lone Phone
}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isForwarded { ... }
check isForwarded for 5
```

CallWaiting

```
refines module Phone
refines sig Phone {
  waitingCall: set Call
} ...
one sig Suspended extends State {}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isSuspended { ... }
check isSuspended for 5
```

Feature Interactions – Phone Example

BasicPhone

```
module Phone
sig Phone { ... }
abstract sig State {}
one sig Idle, Busy extends State {} ...
pred incomingCall [in: Call, disj p, p': Phone] { ... }
```

CallForwarding

```
refines module Phone
refines sig Phone {
  forward: lone Phone
}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isForwarded { ... }
check isForwarded for 5
```

CallWaiting

```
refines module Phone
refines sig Phone {
  waitingCall: set Call
} ...
one sig Suspended extends State {}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isSuspended { ... }
check isSuspended for 5
```

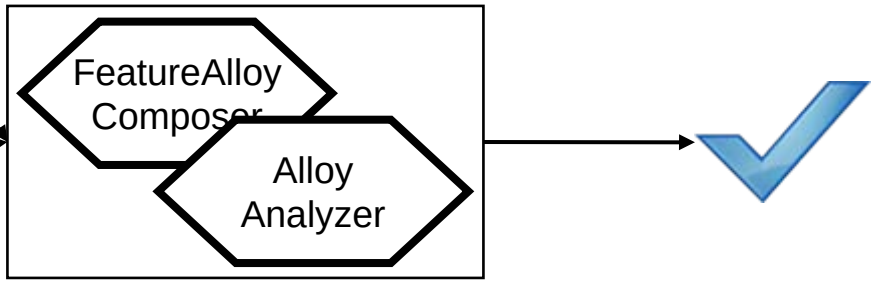
Feature Interactions – Phone Example

BasicPhone

```
module Phone
sig Phone { ... }
abstract sig State {}
one sig Idle, Busy extends State {} ...
pred incomingCall [in: Call, disj p, p': Phone] { ... }
```

CallForwarding

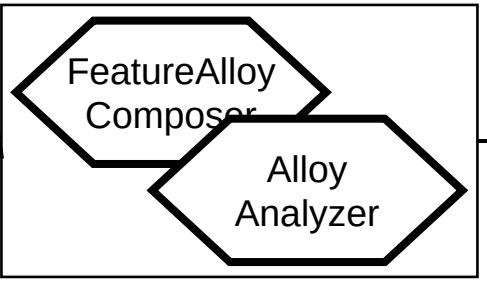
```
refines module Phone
refines sig Phone {
  forward: lone Phone
}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isForwarded { ... }
check isForwarded for 5
```



Feature Interactions – Phone Example

BasicPhone

```
module Phone
sig Phone { ... }
abstract sig State {}
one sig Idle, Busy extends State {} ...
pred incomingCall [in: Call, disj p, p': Phone] { ... }
```



CallWaiting

```
refines module Phone
refines sig Phone {
  waitingCall: set Call
} ...
one sig Suspended extends State {}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isSuspended { ... }
check isSuspended for 5
```

Feature Interactions – Phone Example

BasicPhone

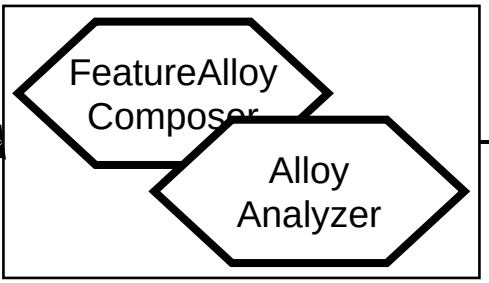
```
module Phone
sig Phone { ... }
abstract sig State {}
one sig Idle, Busy extends State {} ...
pred incomingCall [in: Call, disj p, p': Phone] { ... }
```

CallForwarding

```
refines module Phone
refines sig Phone {
  forward: lone Phone
}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isForwarded { ... }
check isForwarded for 5
```

CallWaiting

```
refines module Phone
refines sig Phone {
  waitingCall: set Call
} ...
one sig Suspended extends State {}
refines pred incomingCall [in: Call, disj p, p': Phone] { ... }
assert isSuspended { ... }
check isSuspended for 5
```



Tool Chain and Case Studies

- Composition with FeatureAlloy@FeatureHouse [Apel et al., ICSE'09]
- Analysis with Alloy Analyzer

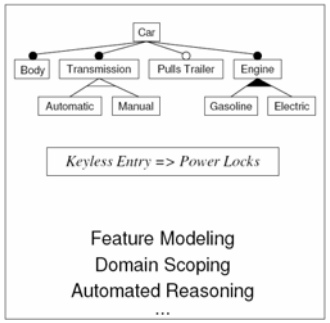
Case study	# Features	# Semantic dependences	# Feature interactions
Graph product line	7	4	0
Phone system	3	0	1
Content addressable network	8	1	2
POSIX file system	8	0	1

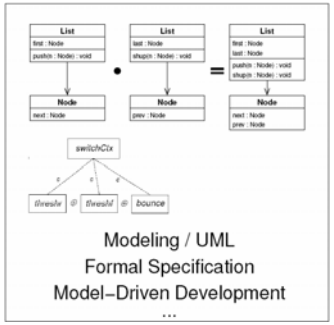
Feature-Oriented Verification

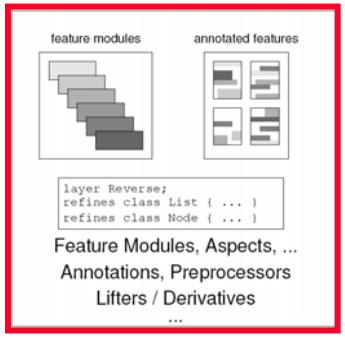
- Question:
  - If we have a correct model, …
  - …how do we know that the actual implementation is correct?

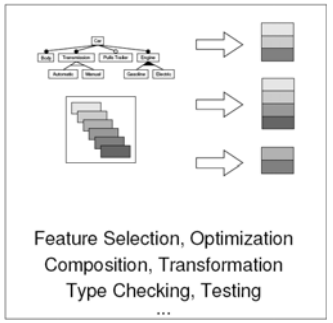
Feature-Oriented Verification

- Question:
  - If we have a correct model, …
  - …how do we know that the actual implementation is correct?
- Answer:
  - Feature-oriented verification at implementation level



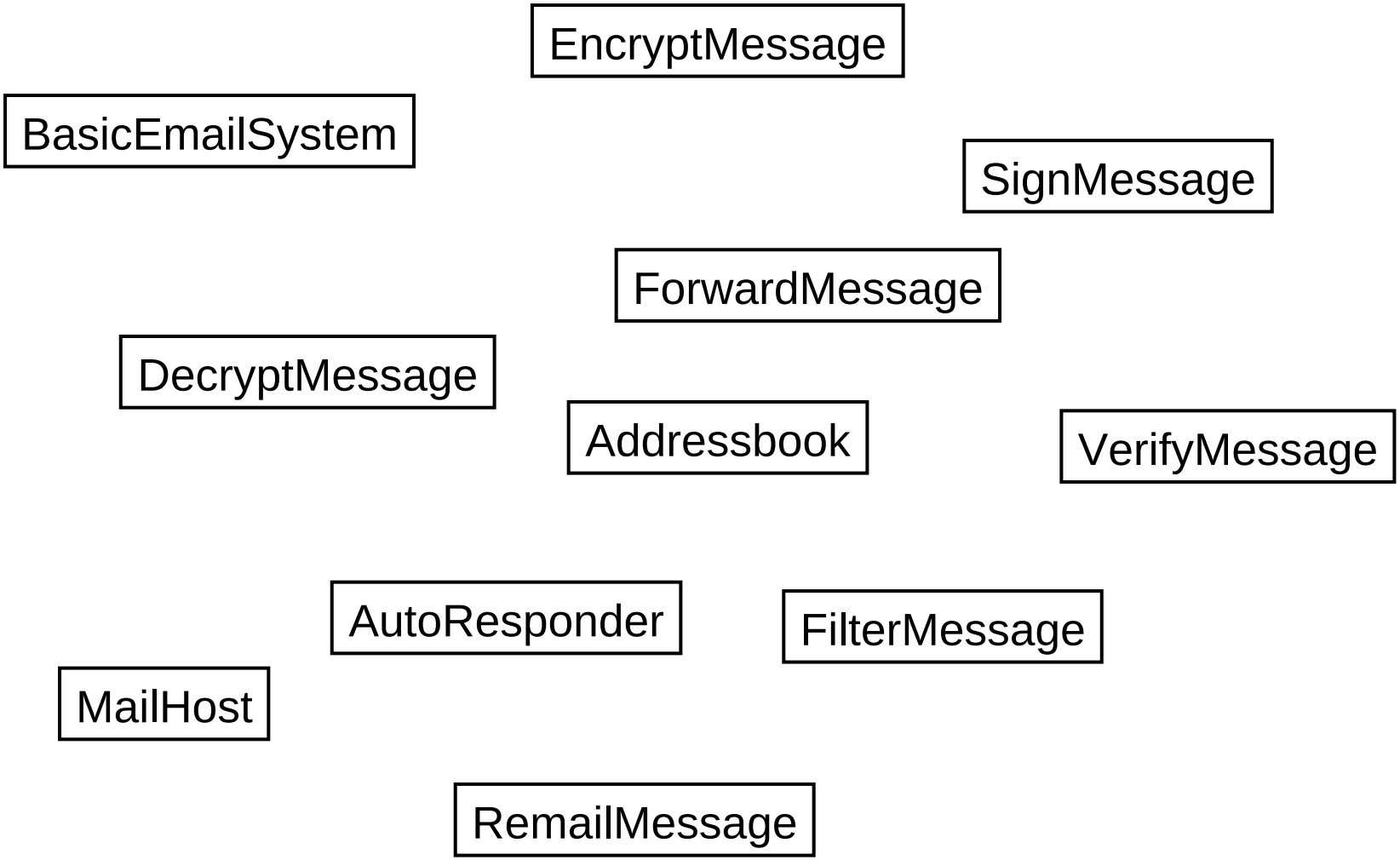






Domain Analysis | Domain Design & Specification | Implementation | Product Derivation

Feature-Oriented Verification – Email Example*

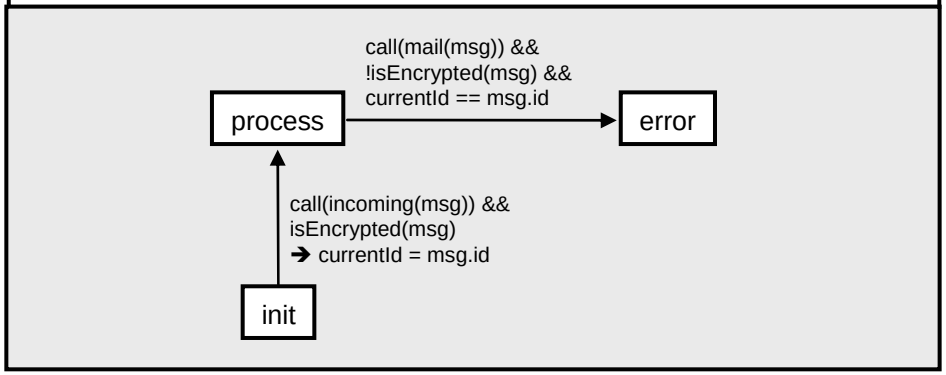


* [Hall, J. ASE 2005]

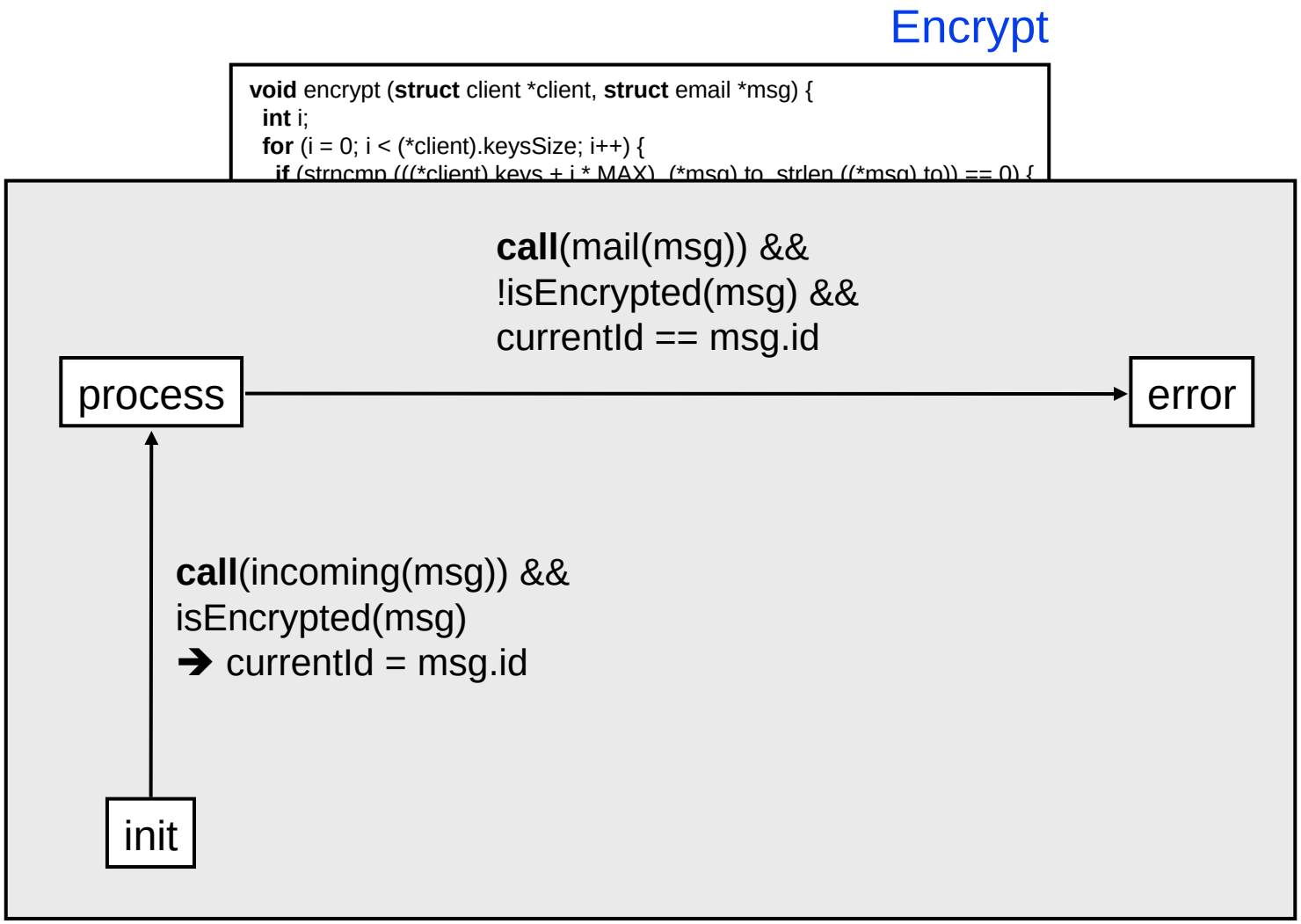
Feature-Oriented Verification – Email Example

Encrypt

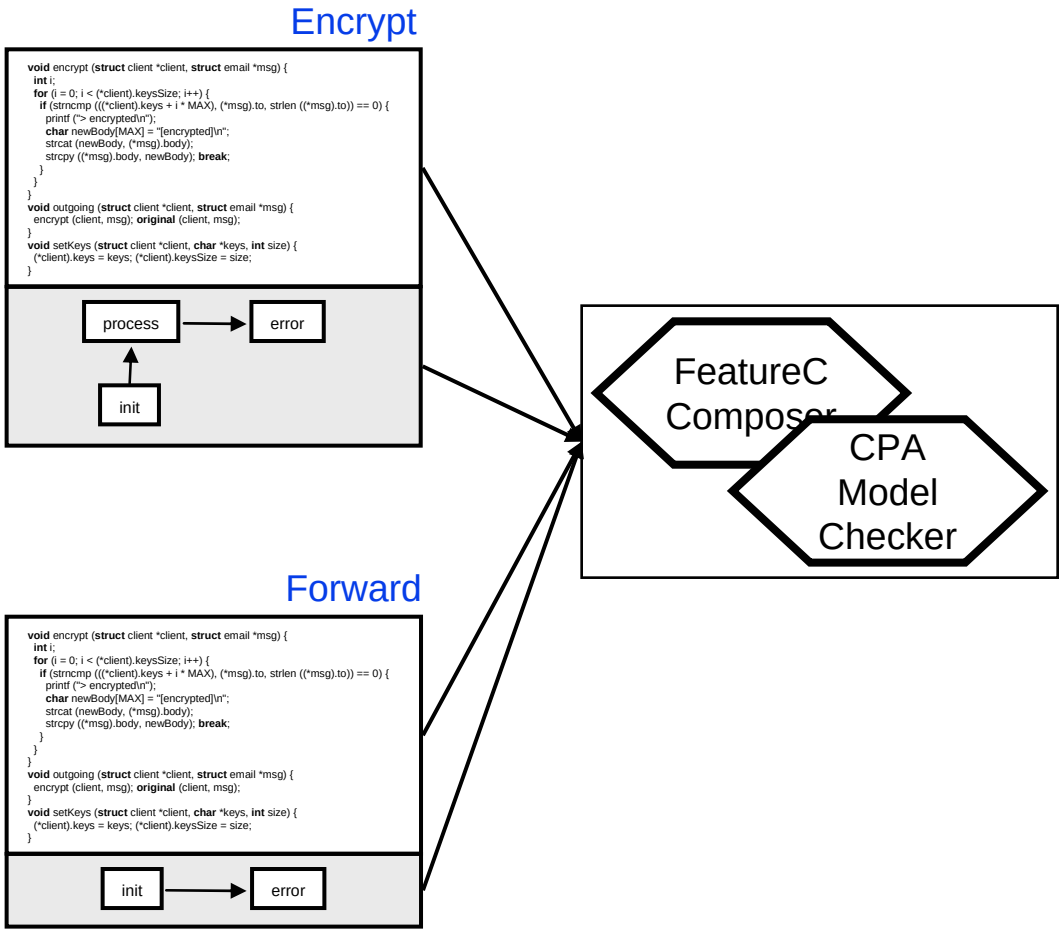
```
void encrypt (struct client *client, struct email *msg) {
  int i;
  for (i = 0; i < (*client).keysSize; i++) {
    if (strncmp (((*client).keys + i * MAX), (*msg).to, strlen ((*msg).to)) == 0) {
      printf ("> encrypted\n");
      char newBody[MAX] = "[encrypted]\n";
      strcat (newBody, (*msg).body);
      strcpy ((*msg).body, newBody); break;
    }
  }
}
void outgoing (struct client *client, struct email *msg) {
  encrypt (client, msg); original (client, msg);
}
void setKeys (struct client *client, char *keys, int size) {
  (*client).keys = keys; (*client).keysSize = size;
}
```



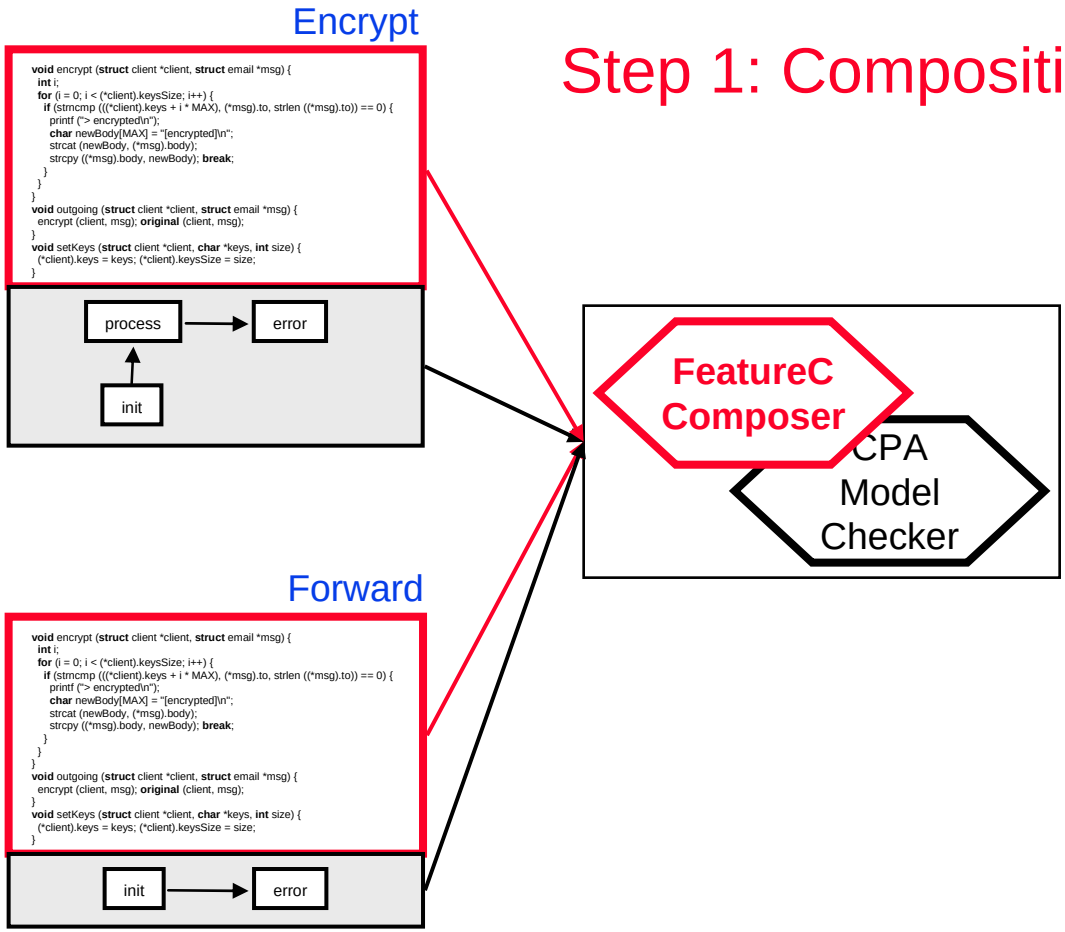
Feature-Oriented Verification – Email Example



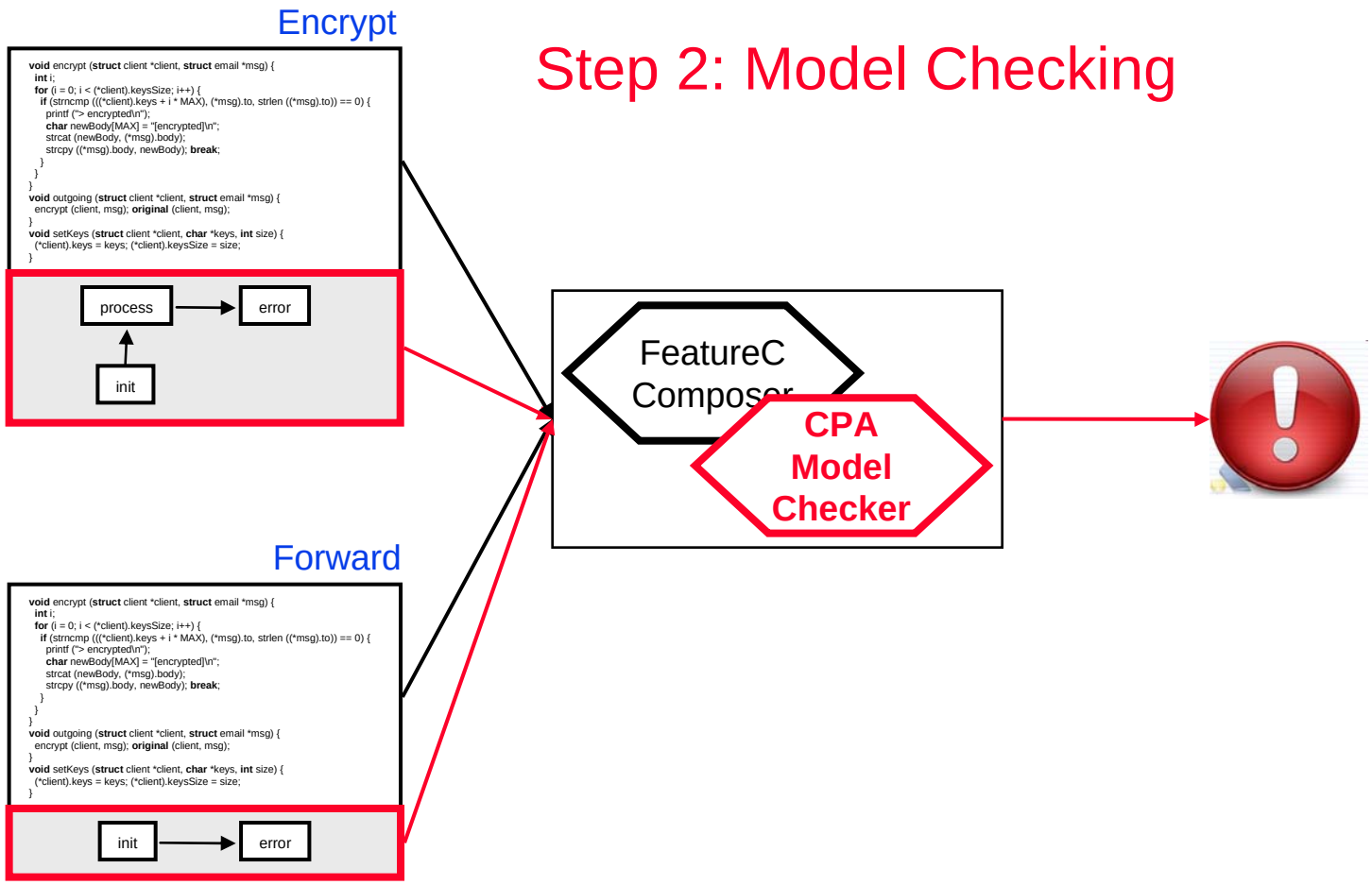
Feature-Oriented Verification – Overview



Feature-Oriented Verification – Overview



Feature-Oriented Verification – Overview

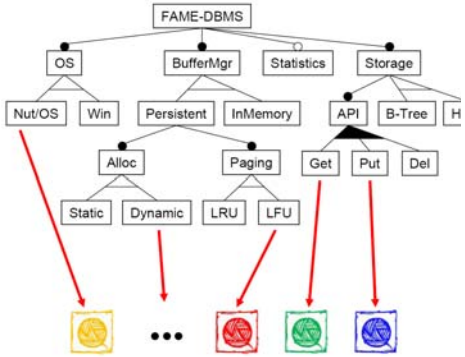


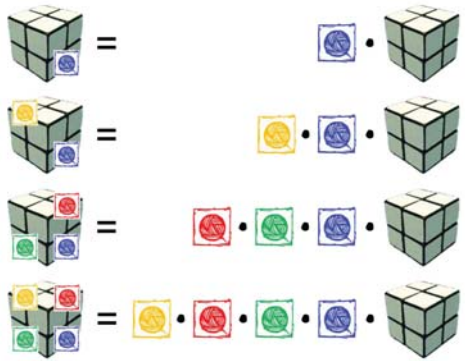
Tool Chain and Case Study

- Tool chain
  - Composition with FeatureC@FeatureHouse [Apel et al., ICSE'09]
  - Specification of observer automata (currently hand-coded)
  - Model checking with CPA Checker (successor of BLAST [Beyer et al., STTT 2007])
- Case study: Email System [Hall, J. ASE 2005]
  - 10 features (7 implemented)
  - 26 interactions (10 can be detected modularly)

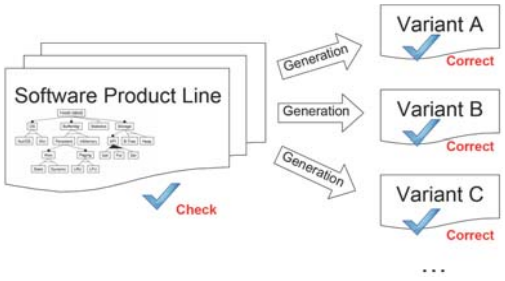
Conclusion

- Feature-oriented software development

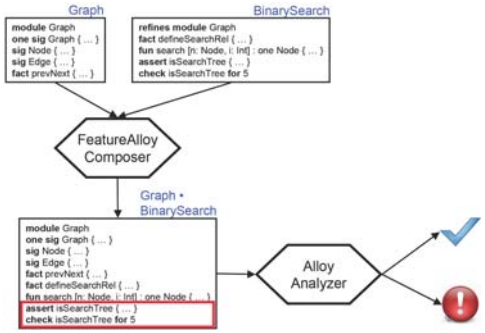




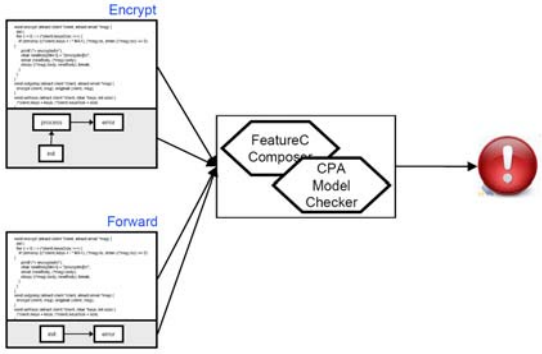
- Safe generation of software products



Product line type system



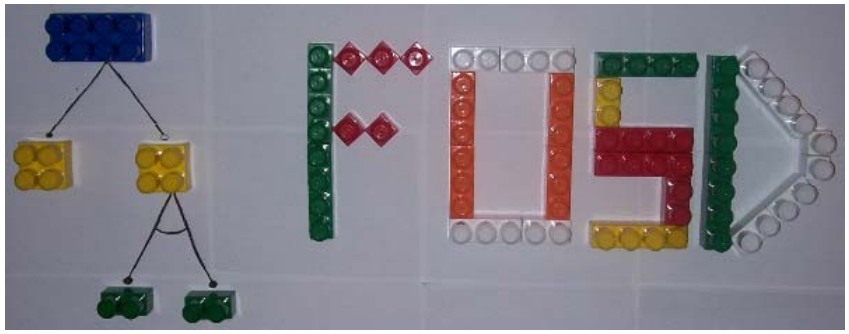
Feature-oriented specification and design



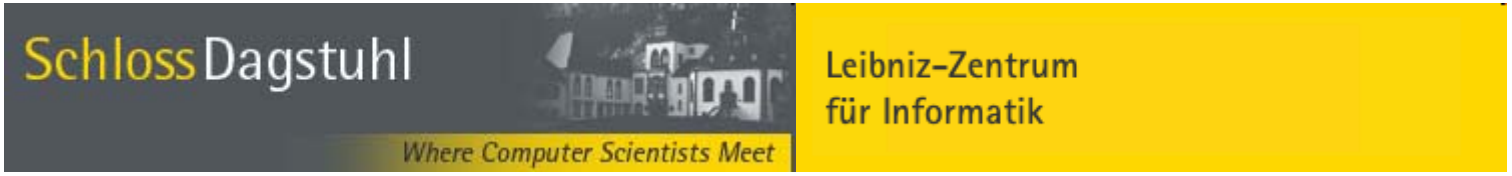
Feature-oriented verification

FOSD Events

- 2nd Workshop on Feature-Oriented Software Development, in conjunction with GPCE and SLE 2010



- Dagstuhl Seminar: Feature-Oriented Software Development (No. 11021). January 9th - 14th, 2011.



Questions?

- Contact information
  - Sven Apel, University of Passau, Germany
  - http://www.infosun.fim.uni-passau.de/cl/staff/apel/
  - apel@uni-passau.de
- FeatureHouse
  - http://www.fosd.de/fh
- Related publications
  - Sven Apel and Christian Kästner. **An Overview of Feature-Oriented Software Development**. *Journal of Object Technology (JOT)*, 8(5):49–84, July 2009.
  - Sven Apel, Christian Kästner, and Christian Lengauer. **FeatureHouse: Language-Independent, Automated Software Composition**. In *Proc. ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 221–231. IEEE, May 2009.
  - Sven Apel, Christian Kästner, Armin Größlinger, and Christian Lengauer. **Type Safety for Feature-Oriented Product Lines**. *Automated Software Engineering – An International Journal*. 2010.

Detected Interactions in the Email Example

- AddressBook vs. EncryptMessage
- SignMessage vs. VerifySignature
- SignMessage vs. ForwardMessage
- EncryptMessage vs. DecryptMessage
- EncryptMessage vs. VerifySignature
- EncryptMessage vs. AutoResponder
- EncryptMessage vs. AutoResponder
- EncryptMessage vs. ForwardMessage
- DecryptMessage vs. AutoResponder
- VerifySignature vs. ForwardMessage
- …