Verification Strategies for Feature-Oriented Software Product Lines

Elias Kuiter, Alexander Knüppel, Tabea Bordis, Tobias Runge, Ina Schaefer

Otto-von-Guericke-University Magdeburg, Technische Universität Braunschweig

VaMoS 2022
February 24–25 | Florence, Italy

Motivation

Motivation

- **Safety** + **security** of configurable software matters
  $\Longrightarrow$ automotive, aircraft, ...

Motivation

- **Safety** + **security** of configurable software matters
  $\Longrightarrow$ automotive, aircraft, ...
- **Modular design** aids with separation of concerns
  $\Longrightarrow$ e.g., with FOP
  $\Longrightarrow$ cannot prevent bugs

Motivation

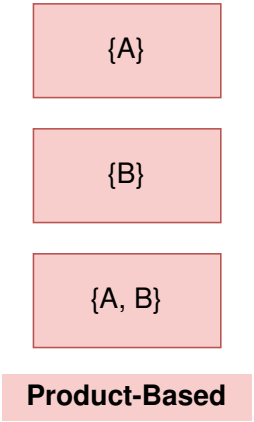
- **Safety + security** of configurable software matters
  $\Longrightarrow$ automotive, aircraft, ...
- **Modular design** aids with separation of concerns
  $\Longrightarrow$ e.g., with FOP
  $\Longrightarrow$ cannot prevent bugs
- **Deductive verification** creates formal proofs of correctness
  $\Longrightarrow$ e.g., with KeY
  $\Longrightarrow$ can show the presence of bugs

Motivation

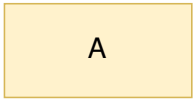
- **Safety + security** of configurable software matters
  $\Longrightarrow$ automotive, aircraft, ...
- **Modular design** aids with separation of concerns
  $\Longrightarrow$ e.g., with FOP
  $\Longrightarrow$ cannot prevent bugs
- **Deductive verification** creates formal proofs of correctness
  $\Longrightarrow$ e.g., with KeY
  $\Longrightarrow$ can show the presence of bugs
- **Verification of feature-oriented SPLs**

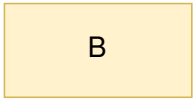
Problem: Scalable Verification

Problem: Scalable Verification

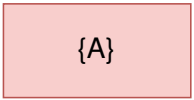


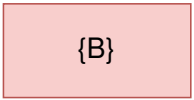
Problem: Scalable Verification

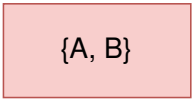




Feature-Based

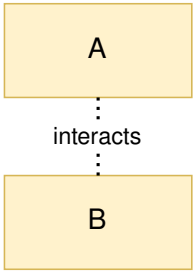




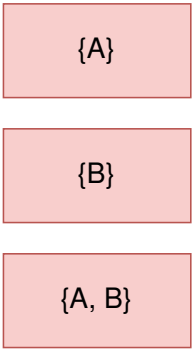


Product-Based

Problem: Scalable Verification

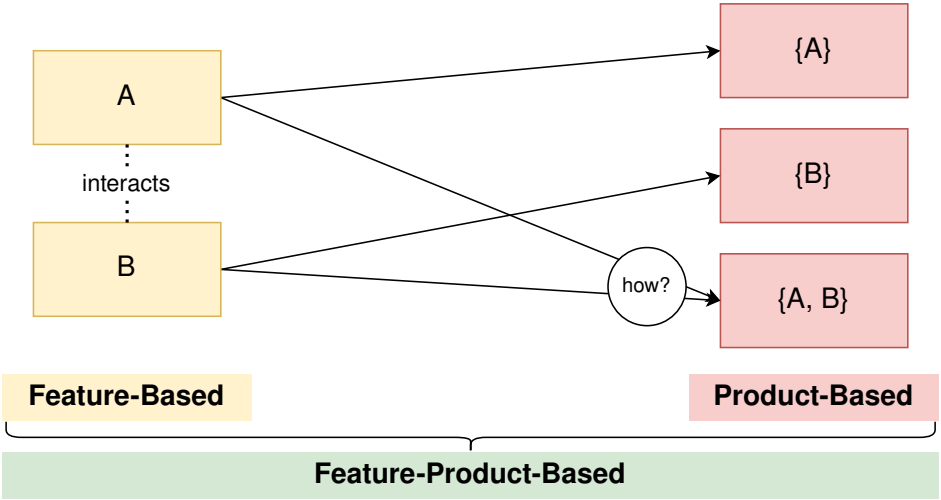


Feature-Based

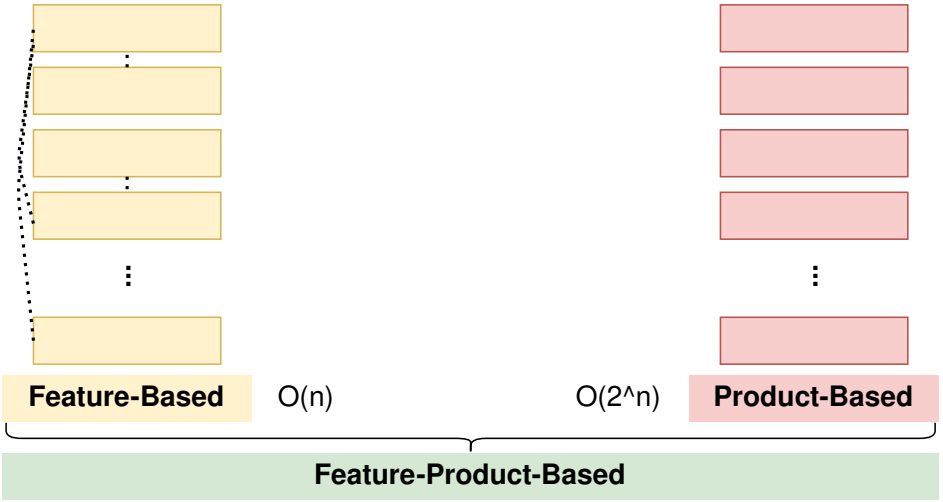


Product-Based

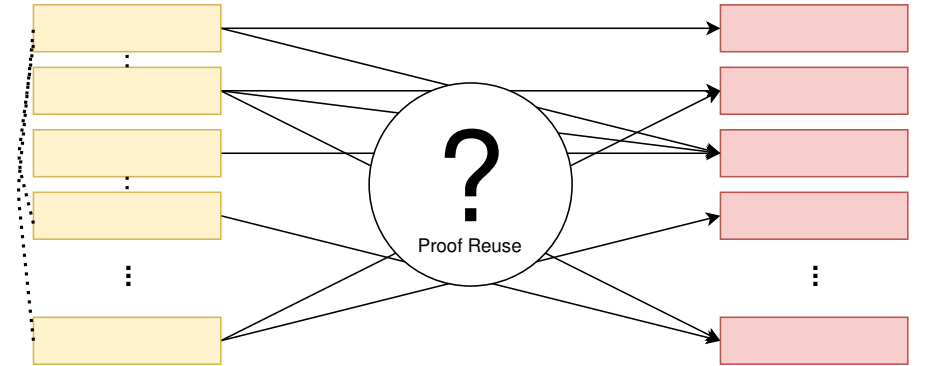
Problem: Scalable Verification



Problem: Scalable Verification



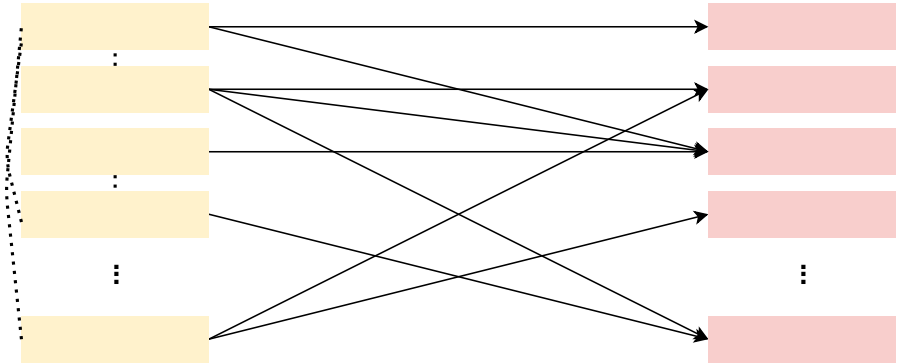
Problem: Scalable Verification



Two Opposing Solutions

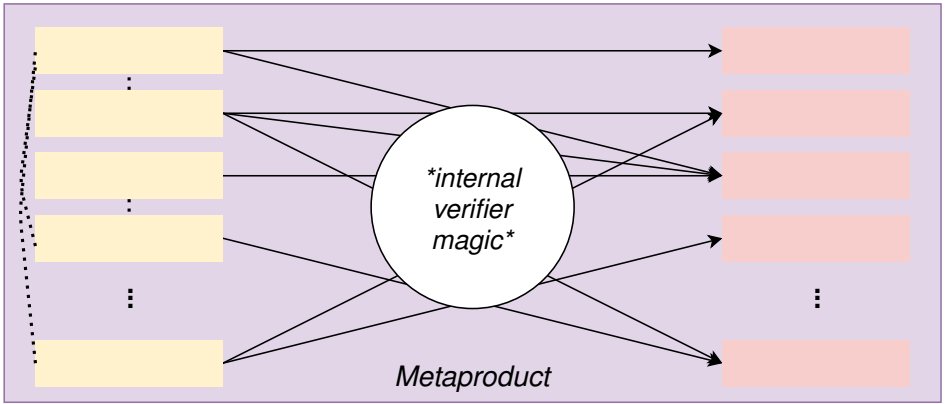
Two Opposing Solutions

Solution 1



Two Opposing Solutions

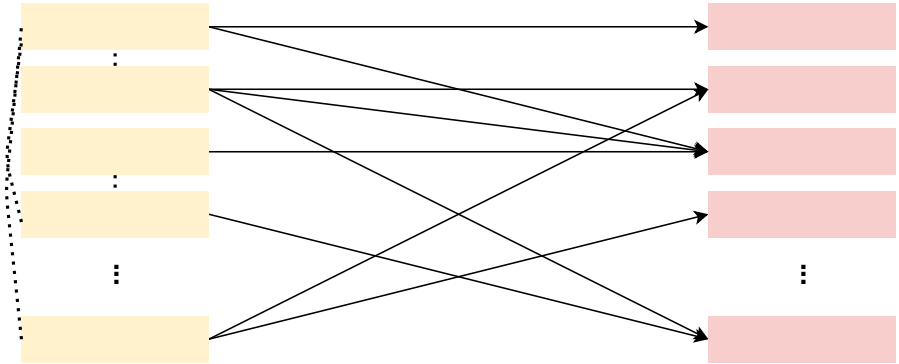
Solution 1



Family-Based

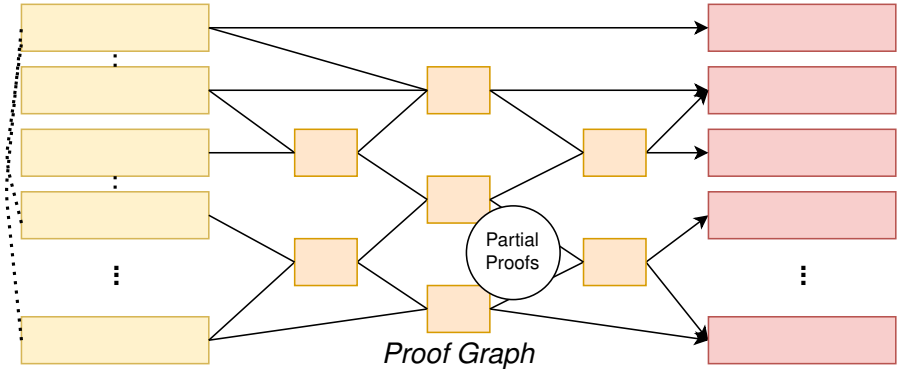
Two Opposing Solutions

Solution 2



Two Opposing Solutions

Solution 2



Feature-Family-Based

Contributions

Contributions

- Formalize **proof graphs and plans**

  $\Longrightarrow$ A novel technique for verification of FOP-based SPLs

Contributions

- Formalize **proof graphs and plans**
  $\Longrightarrow$ A novel technique for verification of FOP-based SPLs
- Express existing **verification strategies** as proof plans
  $\Longrightarrow$ (Feature-)(Family-)(Product-)Based

Contributions

- Formalize **proof graphs and plans**
  $\Longrightarrow$ A novel technique for verification of FOP-based SPLs
- Express existing **verification strategies** as proof plans
  $\Longrightarrow$ (Feature-)(Family-)(Product-)Based
- Propose possible **applications**
  $\Longrightarrow$ Post-Hoc Verification, Evolution Scenarios, Lazy Exploration

Contributions

- Formalize **proof graphs and plans**
  $\Longrightarrow$ A novel technique for verification of FOP-based SPLs
- Express existing **verification strategies** as proof plans
  $\Longrightarrow$ (Feature-)(Family-)(Product-)Based
- Propose possible **applications**
  $\Longrightarrow$ Post-Hoc Verification, Evolution Scenarios, Lazy Exploration
- **Implement** proof plans in KeY

Contributions

- Formalize **proof graphs and plans**
  $\Longrightarrow$ A novel technique for verification of FOP-based SPLs
- Express existing **verification strategies** as proof plans
  $\Longrightarrow$ (Feature-)(Family-)(Product-)Based
- Propose possible **applications**
  $\Longrightarrow$ Post-Hoc Verification, Evolution Scenarios, Lazy Exploration
- **Implement** proof plans in KeY
- **Evaluate** proof plans on a case study

Contributions

- Formalize **proof graphs and plans**
  $\Longrightarrow$ A novel technique for verification of FOP-based SPLs
- Express existing **verification strategies** as proof plans
  $\Longrightarrow$ (Feature-)(Family-)(Product-)Based
- Propose possible **applications**
  $\Longrightarrow$ Post-Hoc Verification, Evolution Scenarios, Lazy Exploration
- **Implement** proof plans in KeY
- **Evaluate** proof plans on a case study
- **Goals:**
  $\Longrightarrow$ Align proof reuse with software reuse by small, modular proofs
  $\Longrightarrow$ Position proof plans as manipulatable, first-class objects

Proof Reuse with Partial Proofs

Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

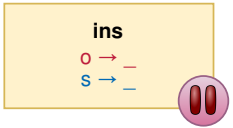
$\{A \text{ is sorted} \wedge x \in A\}$

Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$

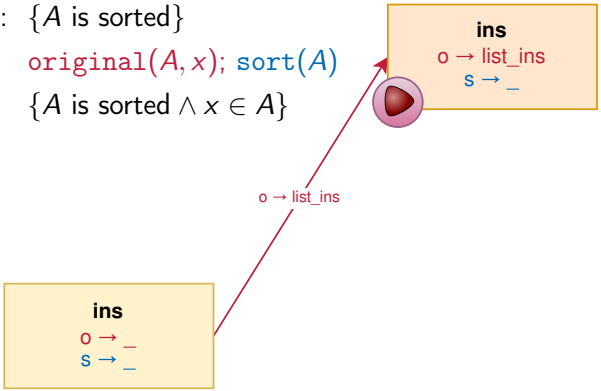


Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$

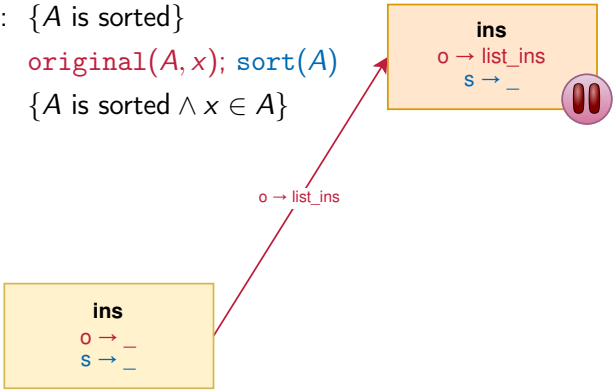


Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$

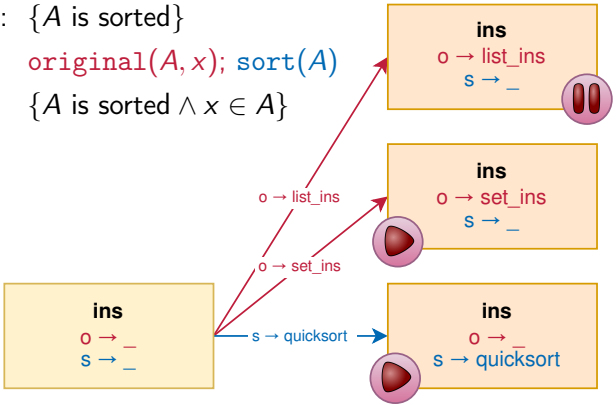


Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$

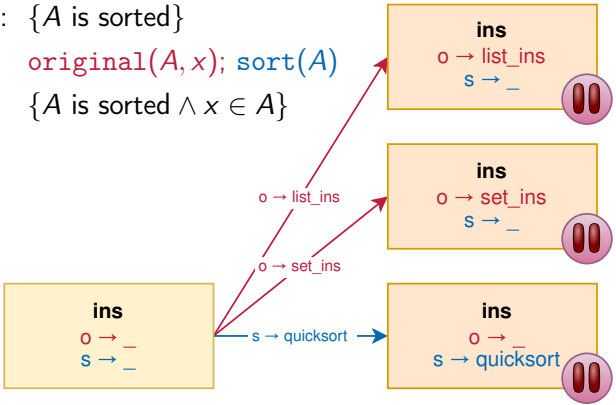


Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$

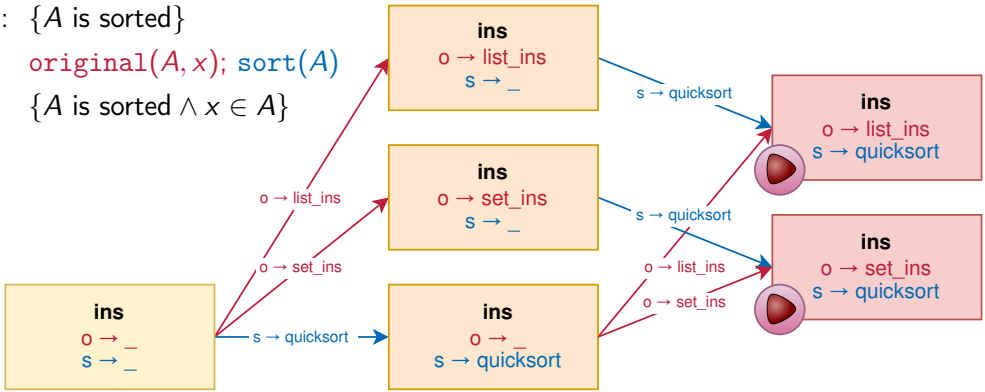


Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x);\ \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$

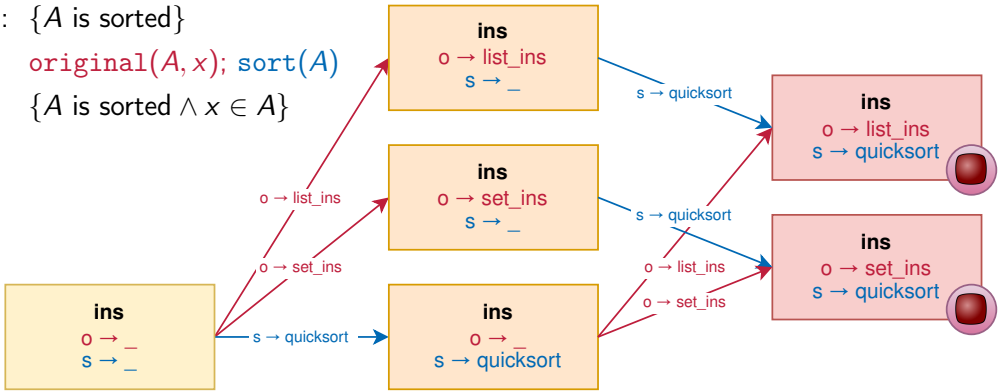


Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x);\ \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$

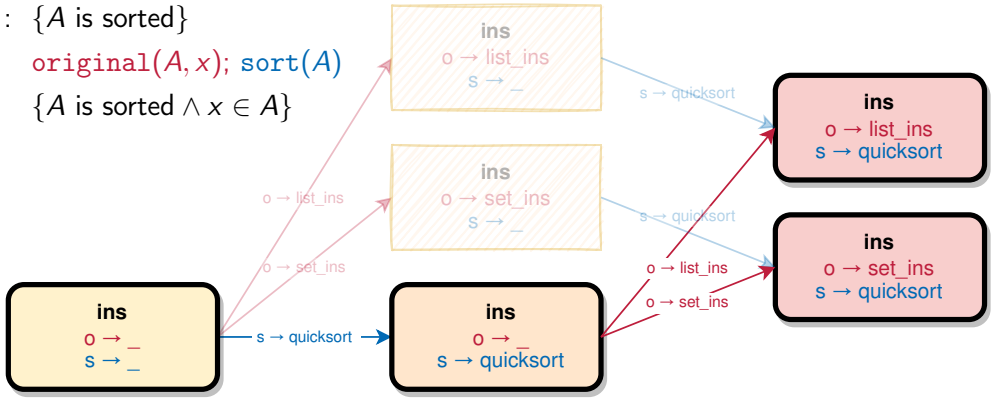


Proof Reuse with Partial Proofs

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x)$; $\texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$



Proof Plan with Reuse

An SPL and its Proof Graph

An SPL and its Proof Graph

	List	Set	Ord
ins	●	○	○
find	●	×	○
sort	×	×	○

$\{List\}$

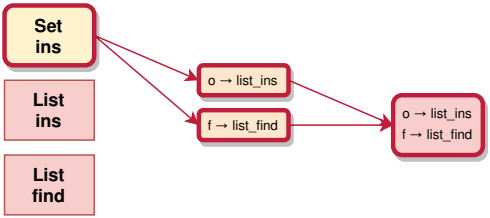
An SPL and its Proof Graph

	List	Set	Ord
ins	●	○	○
find	●	×	○
sort	×	×	○

$\{List\}$

	List	Set	Ord
ins	●	●	○
find	●	×	○
sort	×	×	○

$\{List, Set\}$



An SPL and its Proof Graph

	List	Set	Ord
ins	●	○	○
find	●	×	○
sort	×	×	○

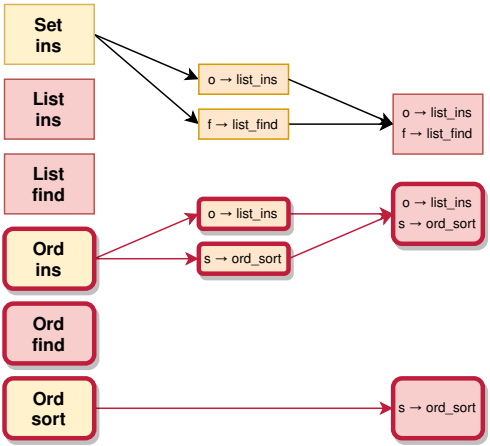
$\{List\}$

	List	Set	Ord
ins	●	●	○
find	●	×	○
sort	×	×	○

$\{List, Set\}$

	List	Set	Ord
ins	●	○	●
find	●	×	●
sort	×	×	●

$\{List, Ord\}$



An SPL and its Proof Graph

	List	Set	Ord
ins	●	○	○
find	●	×	○
sort	×	×	○

$\{List\}$

	List	Set	Ord
ins	●	●	○
find	●	×	○
sort	×	×	○

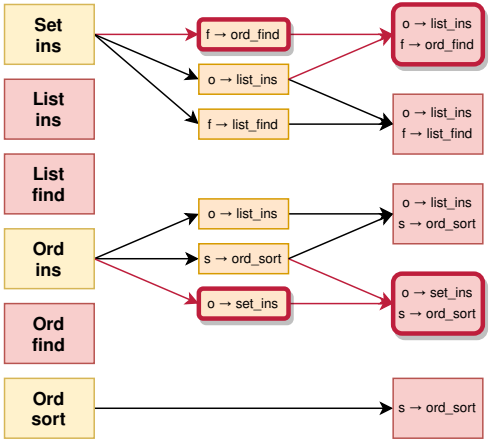
$\{List, Set\}$

	List	Set	Ord
ins	●	○	●
find	●	×	●
sort	×	×	●

$\{List, Ord\}$

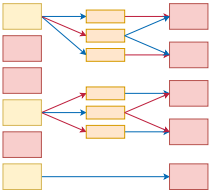
	List	Set	Ord
ins	●	●	●
find	●	×	●
sort	×	×	●

$\{List, Set, Ord\}$



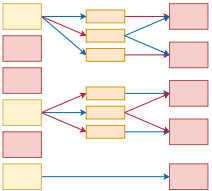
A Spectrum of Verification Strategies

A Spectrum of Verification Strategies

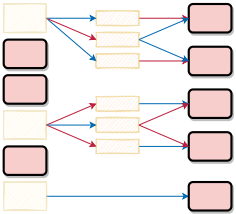


Proof Graph

A Spectrum of Verification Strategies

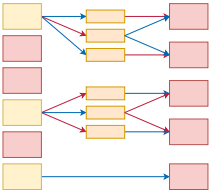


Proof Graph

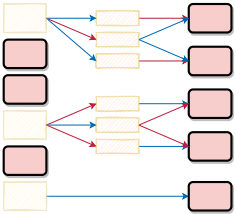


Product-Based

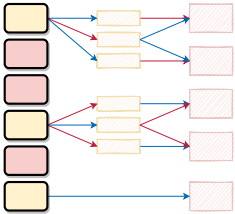
A Spectrum of Verification Strategies



Proof Graph

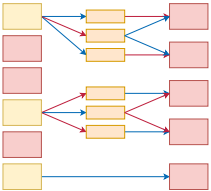


Product-Based

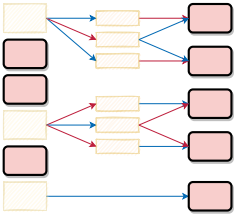


Feature-Based

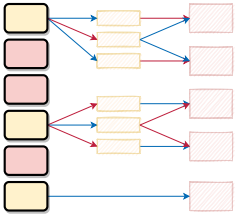
A Spectrum of Verification Strategies



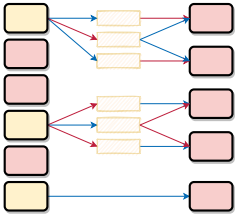
Proof Graph



Product-Based

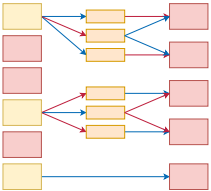


Feature-Based

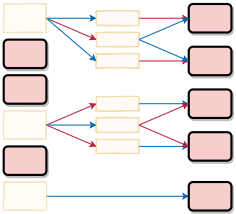


Feature-Product-Based

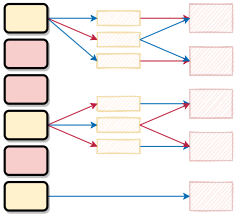
A Spectrum of Verification Strategies



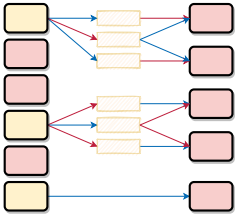
Proof Graph



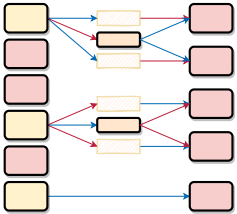
Product-Based



Feature-Based

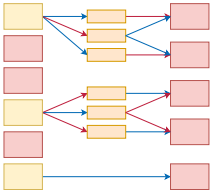


Feature-Product-Based



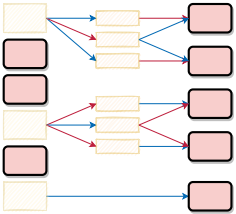
Feature-Family-Based*

A Spectrum of Verification Strategies

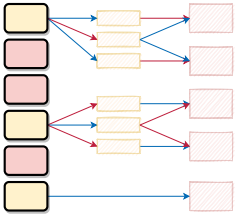


Proof Graph

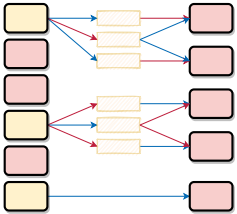
* *How to choose well?*



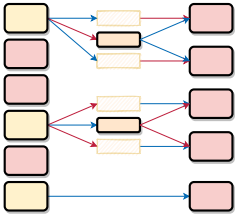
Product-Based



Feature-Based



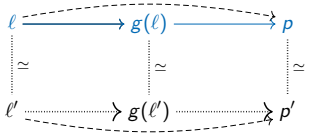
Feature-Product-Based



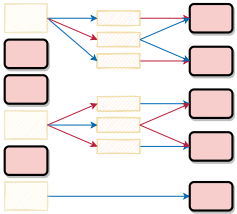
Feature-Family-Based*

A Spectrum of Verification Strategies

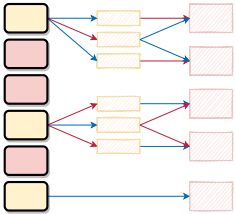
Applications



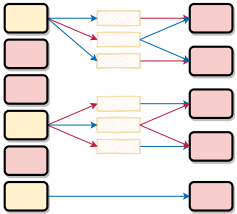
** How to choose well?*



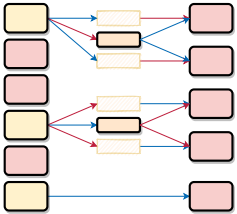
Product-Based



Feature-Based



Feature-Product-Based

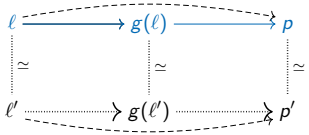


Feature-Family-Based*

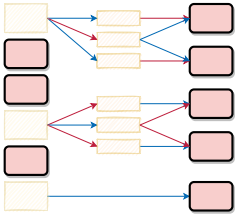
A Spectrum of Verification Strategies

Applications

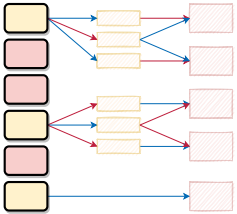
- Post-Hoc Verification



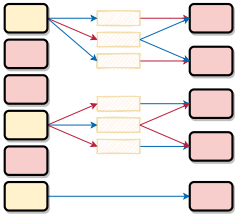
** How to choose well?*



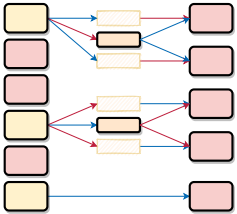
Product-Based



Feature-Based



Feature-Product-Based

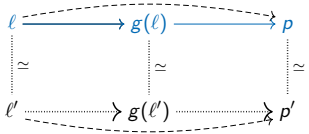


Feature-Family-Based*

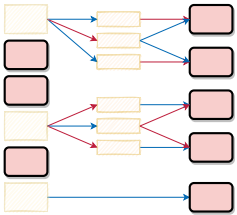
A Spectrum of Verification Strategies

Applications

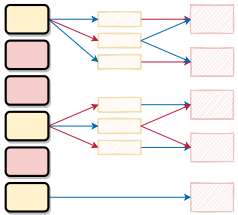
- Post-Hoc Verification
- Evolution Scenarios



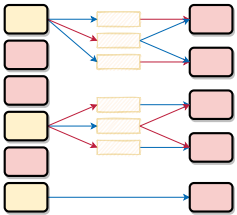
** How to choose well?*



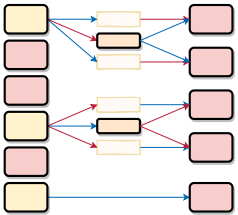
Product-Based



Feature-Based



Feature-Product-Based

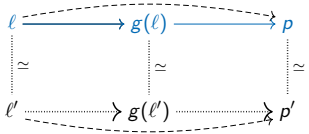


Feature-Family-Based*

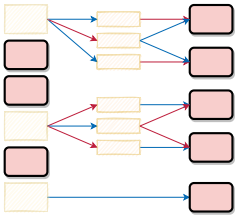
A Spectrum of Verification Strategies

Applications

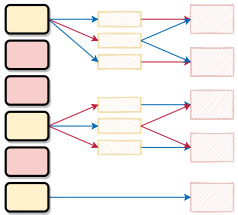
- Post-Hoc Verification
- Evolution Scenarios
- Lazy Exploration*



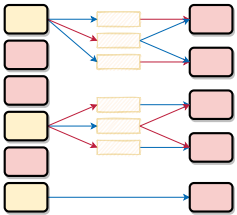
** How to choose well?*



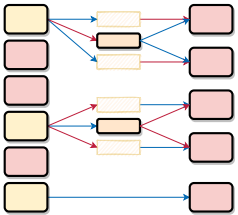
Product-Based



Feature-Based



Feature-Product-Based



Feature-Family-Based*

Evaluation

[1] https://github.com/ekuiter/KeYPl

Evaluation

- **Subject**: *List* case study[1]
  $\Longrightarrow$ Java + JML + FeatureHouse, 5 features, 13 methods

[1] `https://github.com/ekuiter/KeYPl`

Evaluation

- **Subject**: *List* case study[1]
  $\Longrightarrow$ Java + JML + FeatureHouse, 5 features, 13 methods
- **Application**: Post-hoc verification

[1] `https://github.com/ekuiter/KeYPl`

Evaluation

- **Subject**: *List* case study[1]
  $\Longrightarrow$ Java + JML + FeatureHouse, 5 features, 13 methods
- **Application**: Post-hoc verification
- **Verification system**: KeY 2.8.0 + *KeYPl* prototype[1]

[1] `https://github.com/ekuiter/KeYPl`

Evaluation

- **Subject**: *List* case study[1]
  $\Longrightarrow$ Java + JML + FeatureHouse, 5 features, 13 methods
- **Application**: Post-hoc verification
- **Verification system**: KeY 2.8.0 + *KeYPl* prototype[1]
- **Partial proof mechanism**: Abstract `@model` methods

[1] `https://github.com/ekuiter/KeYPl`

Evaluation

- **Subject**: *List* case study[1]
  $\Longrightarrow$ Java + JML + FeatureHouse, 5 features, 13 methods
- **Application**: Post-hoc verification
- **Verification system**: KeY 2.8.0 + *KeYPl* prototype[1]
- **Partial proof mechanism**: Abstract `@model` methods

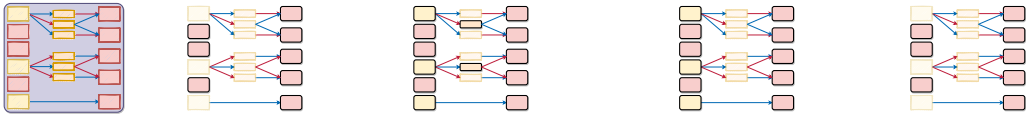
Verification strategy	*Time [s]*
Family-Based	20.0
Product-Based (~~dup~~)	23.7
Feature-Family-Based	26.2
Feature-Product-Based	28.8
Product-Based (dup)	157.7

[1] `https://github.com/ekuiter/KeYPl`

Evaluation

- **Subject**: *List* case study[1]
  $\Longrightarrow$ Java + JML + FeatureHouse, 5 features, 13 methods
- **Application**: Post-hoc verification
- **Verification system**: KeY 2.8.0 + *KeYPl* prototype[1]
- **Partial proof mechanism**: Abstract `@model` methods

Verification strategy	*Time [s]*
Family-Based	20.0
Product-Based (~~dup~~)	23.7
Feature-Family-Based	26.2
Feature-Product-Based	28.8
Product-Based (dup)	157.7



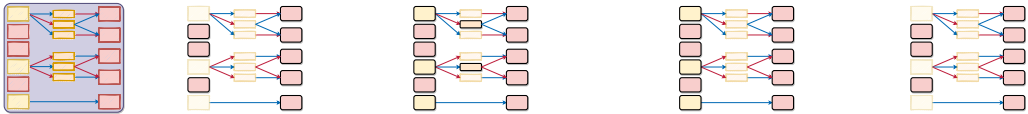
Family-Based > Product-Based (~~dup~~) > Feature-Family-Based > Feature-Product-Based ≫ Product-Based (dup)

[1] `https://github.com/ekuiter/KeYPl`

Evaluation

- **Subject**: *List* case study[1]
  $\Longrightarrow$ Java + JML + FeatureHouse, 5 features, 13 methods
- **Application**: Post-hoc verification
- **Verification system**: KeY 2.8.0 + *KeYPl* prototype[1]
- **Partial proof mechanism**: Abstract `@model` methods

Verification strategy	*Time [s]*
Family-Based	20.0
Product-Based (~~dup~~)	23.7
Feature-Family-Based	26.2
Feature-Product-Based	28.8
Product-Based (dup)	157.7

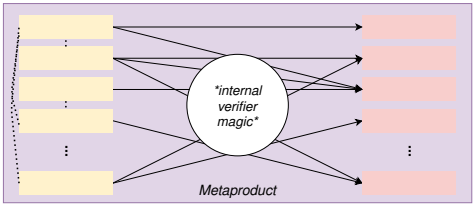


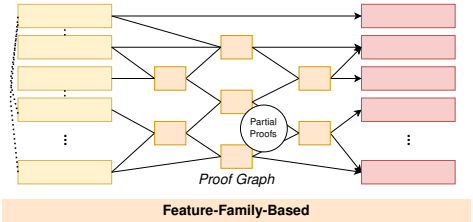
Family-Based > Product-Based (~~dup~~) > Feature-Family-Based > Feature-Product-Based ≫ Product-Based (dup)

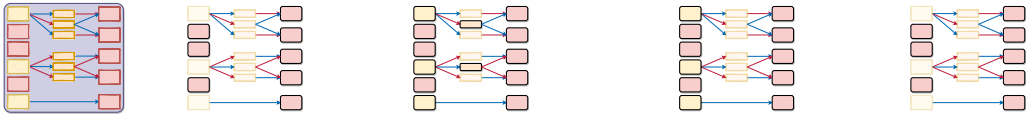
Threats to Validity?

[1] `https://github.com/ekuiter/KeYPl`

Conclusion







Family-Based > Product-Based > Feature-Family-Based > Feature-Product-Based ≫ Product-Based

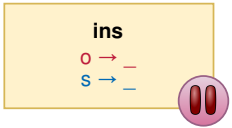
(~~dup~~) (dup)

Abstract Model Methods

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$



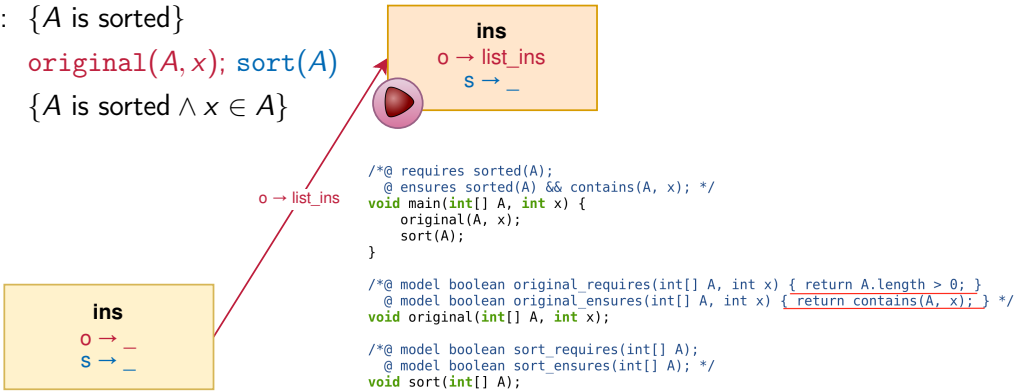

```
/*@ requires sorted(A);
  @ ensures sorted(A) && contains(A, x); */
void main(int[] A, int x) {
    original(A, x);
    sort(A);
}

/*@ model boolean original_requires(int[] A, int x);
  @ model boolean original_ensures(int[] A, int x); */
void original(int[] A, int x);

/*@ model boolean sort_requires(int[] A);
  @ model boolean sort_ensures(int[] A); */
void sort(int[] A);
```

Abstract Model Methods

$\texttt{ins}(A, x)$: $\{A \text{ is sorted}\}$

$\texttt{original}(A, x); \texttt{sort}(A)$

$\{A \text{ is sorted} \wedge x \in A\}$




```
/*@ requires sorted(A);
  @ ensures sorted(A) && contains(A, x); */
void main(int[] A, int x) {
    original(A, x);
    sort(A);
}

/*@ model boolean original_requires(int[] A, int x) { return A.length > 0; }
  @ model boolean original_ensures(int[] A, int x) { return contains(A, x); } */
void original(int[] A, int x);

/*@ model boolean sort_requires(int[] A);
  @ model boolean sort_ensures(int[] A); */
void sort(int[] A);
```