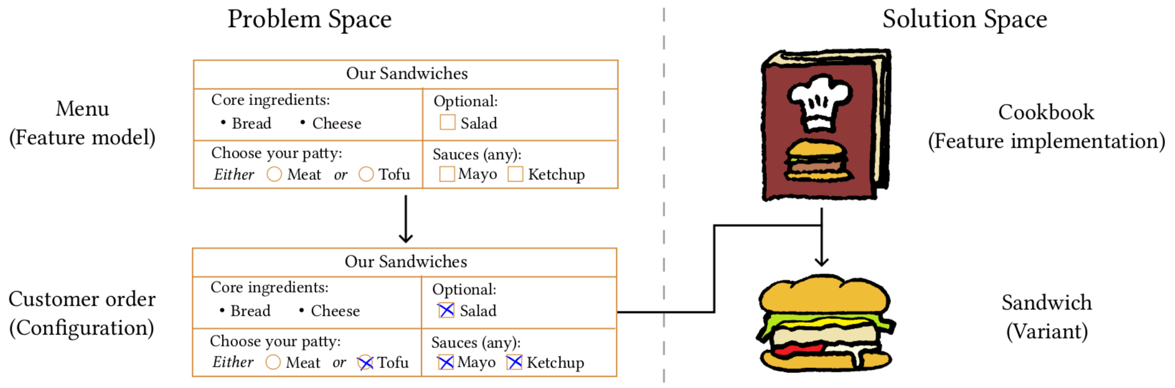


Our Sandwiches

Core ingredients: • Bread • Cheese	Optional: ☐ Salad
Choose your patty: *Either* ◯ Meat *or* ◯ Tofu	Sauces (any): ☐ Mayo ☐ Ketchup

Our Sandwiches

Core ingredients: • Bread • Cheese	Optional: [✓ Salad]
Choose your patty: *Either* ◯ Meat *or* [✓ Tofu]	Sauces (any): [✓ Mayo] [✓ Ketchup]

Theses Topics

Paul Bittner, Sebastian Krieter | April 18, 2024

I am. . .

- Paul Bittner
- a PhD student at SE Group at Uni Paderborn
- a Bachelor+Master from TU Braunschweig, began PhD at Ulm University

I work on . . .

formal languages for software variability,

evolution of configurable software,

synchronizing diverging branches and forks,

interactive theorem-proving.

I am. . .

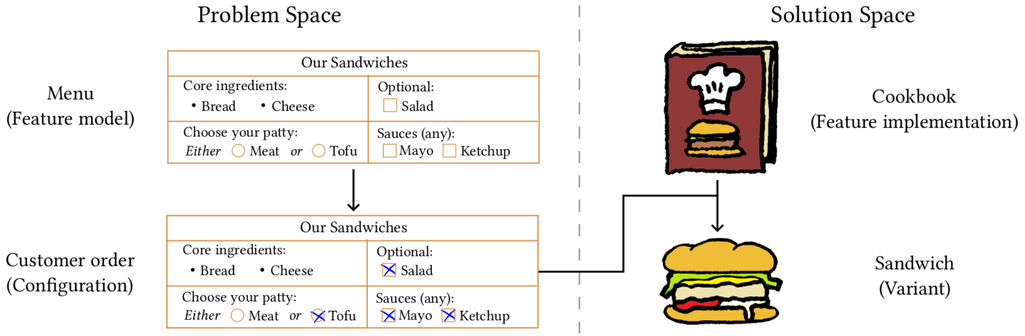
- Sebastian Krieter
- Postdoc at SE Group at Uni Paderborn
- Studied in Magdeburg , Worked at Harz University of Applied Sciences and Ulm University

I work on . . .

feature-model analysis and software product line testing

sampling of configurable systems

Engineering Highly-Configurable Systems at the Example of Sandwiches



Our Sandwiches	
Core ingredients: • Bread • Cheese	Optional: ☐ Salad
Choose your patty: *Either* ○ Meat *or* ○ Tofu	Sauces (any): ☐ Mayo ☐ Ketchup

Our Sandwiches	
Core ingredients: • Bread • Cheese	Optional: ☒ Salad
Choose your patty: *Either* ○ Meat *or* [✓ Tofu]	Sauces (any): ☒ Mayo ☒ Ketchup

Background: Presence Conditions

Presence Conditions	Source File
Log	`//#if Logging`
Log	`public class Logger {`
Log	`  public void log(String message) {`
$(Log \wedge EXT)$	`    //#if EXT`
$(Log \wedge EXT)$	`    print("FS␣is␣EXT");`
$(Log \wedge EXT)$	`    //#endif`
$(Log \wedge Mac) \vee (Log \wedge Win)$	`    //#if macOS || Windows`
$(Log \wedge Mac) \vee (Log \wedge Win)$	`    print("OS␣is␣not␣Linux");`
$(Log \wedge Mac) \vee (Log \wedge Win)$	`    //#endif`
Log	`  }`
Log	`}`
Log	`//#endif`

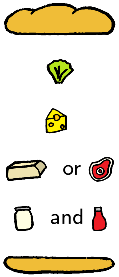
;

Formal Languages for Variability — Example: Choice Calculus

$$
\begin{array}{rcll}
e & ::= & a\prec e, \ldots, e\succ & \textit{Object Structure} \\
  & | & D\langle e, \ldots, e\rangle & \textit{Choice}
\end{array}
$$

Formal Languages for Variability — Example: Choice Calculus

$$\begin{array}{lcll} e & ::= & a{\prec}e,\ldots,e{\succ} & \textit{Object Structure} \\ & | & D\langle e,\ldots,e\rangle & \textit{Choice} \end{array}$$



always

maybe

always

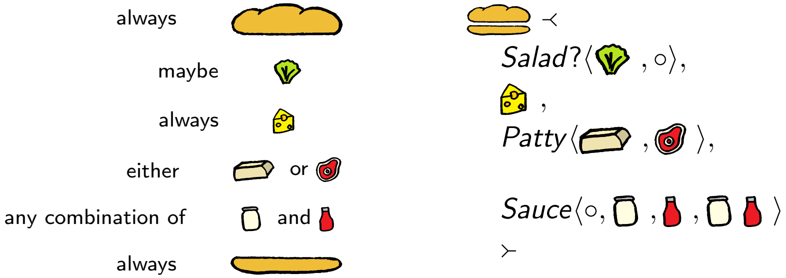
either … or

any combination of … and

always

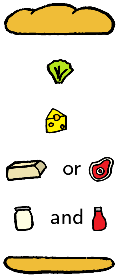
Formal Languages for Variability — Example: Choice Calculus

$$
\begin{array}{rcll}
e & ::= & a\prec e,\ldots,e\succ & \textit{Object Structure} \\
 & | & D\langle e,\ldots,e\rangle & \textit{Choice}
\end{array}
$$

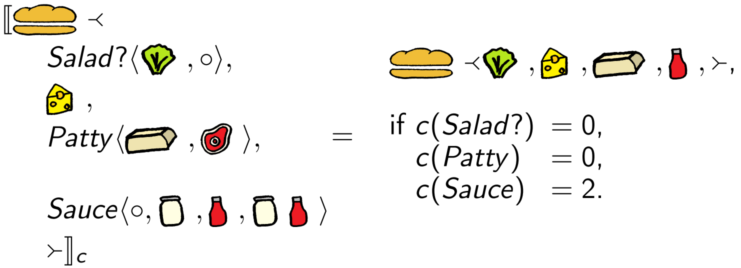


Formal Languages for Variability — Example: Choice Calculus

$$
\begin{array}{lcll}
e & ::= & a\prec e,\ldots,e\succ & \textit{Object Structure} \\
  & | & D\langle e,\ldots,e\rangle & \textit{Choice}
\end{array}
$$



always	
maybe	
always	
either	or
any combination of	and
always	



$[\![\prec$

$Salad?\langle\ ,\circ\rangle,$

$,$

$Patty\langle\ ,\ \rangle,$

$Sauce\langle\circ,\ ,\ ,\ ,\ \rangle$

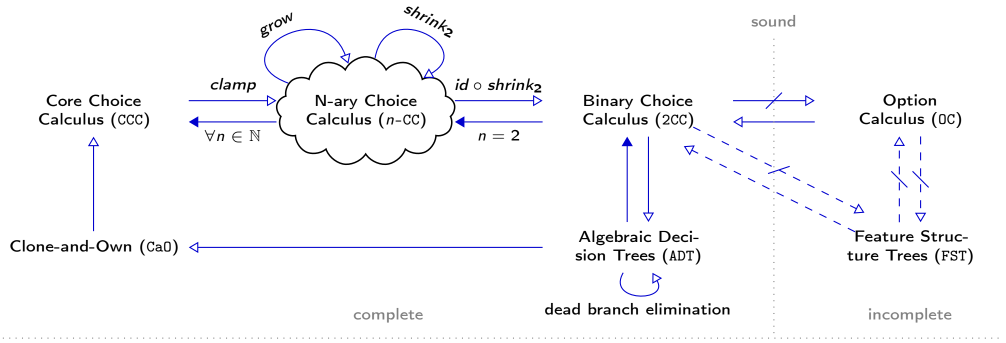
$\succ]\!]_c$

$=$

$\prec\ ,\ ,\ ,\ ,\succ,$

$$
\begin{array}{lll}
\text{if } c(Salad?) & = & 0, \\
c(Patty) & = & 0, \\
c(Sauce) & = & 2.
\end{array}
$$

Comparing the Expressive Power of Variability Languages


sound
grow
$shrink_2$
clamp
$id \circ shrink_2$
Core Choice Calculus (CCC)
N-ary Choice Calculus (n-CC)
Binary Choice Calculus (2CC)
Option Calculus (OC)
$\forall n \in \mathbb{N}$
$n = 2$
Clone-and-Own (CaO)
Algebraic Decision Trees (ADT)
Feature Structure Trees (FST)
dead branch elimination
complete
incomplete

Topic: On the Expressive Power of Variation Trees

Context

There are many formal languages for static software variability. These languages formalize and specify implementations such as C preprocessor, feature-oriented, aspect-oriented, delta-oriented programming. We can compare the expressive power of such languages by means of compilers between languages and correctness proofs.

Problem

Variation Trees are a graph-based language used to study the evolution of configurable software. Basic properties of variation trees such as soundness, completeness, and expressiveness have not yet been explored.

Task

Formalize variation trees in Agda, and prove their (in)completeness, (un)soundness, and relate their expressive power to the other languages, we have already formalized in our library.

Useful but not Mandatory Experience

interest in formal methods or compilers, experience in functional programming (e.g., Haskell or Agda)

Further Reading

- *Programming Language Foundations in Agda*
- *The Choice Calculus – A Formal Language of Variation, Walkingshaw, Dissertation*
- *Classifying Edits to Variability in Source Code, Bittner et al., ESEC/FSE'22*

M.Sc.

Topic: Formalizing Levels of Expressiveness

Context

See previous slide + As an absolute measure for expressiveness, we introduced *completeness*. A language is complete, if it can describe any set of variants. We detected different levels of expressiveness of languages in an initial study.

Problem

Besides completeness, there is no absolute measure for other expressiveness levels. These levels are not yet clearly understood and formalized as distinct properties.

Task

Formalize language properties besides completeness in Agda. Investigate relations between your properties and which existing (incomplete) languages satisfy these properties.

Useful but not Mandatory Experience

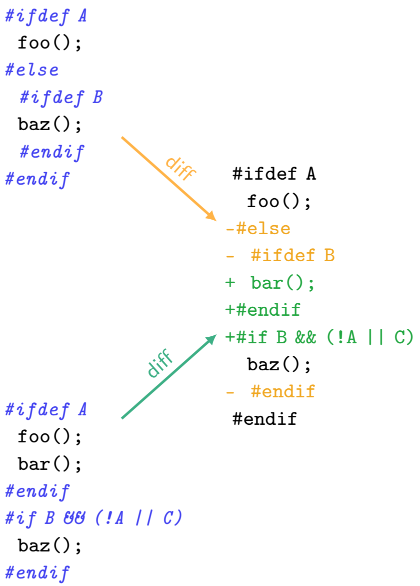
interest in formal methods and type theory
experience in functional programming (e.g., Haskell or Agda)

Further Reading

- see previous slide
- contact us for a preprint with more concrete information

```
#ifdef A
 foo();
#else
 #ifdef B
 baz();
 #endif
#endif
```

```
#ifdef A
 foo();
 bar();
#endif
#if B && (!A || C)
 baz();
#endif
```



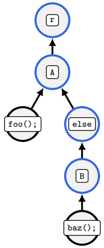
```
#ifdef A
 foo();
#else
 #ifdef B
 baz();
 #endif
#endif
```

```
 #ifdef A
  foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
  baz();
-  #endif
 #endif
```

```
#ifdef A
 foo();
 bar();
#endif
#if B && (!A || C)
 baz();
#endif
```

```
#ifdef A
 foo();
#else
  #ifdef B
 baz();
  #endif
#endif
```

parse



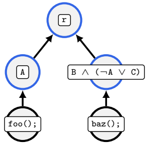
diff

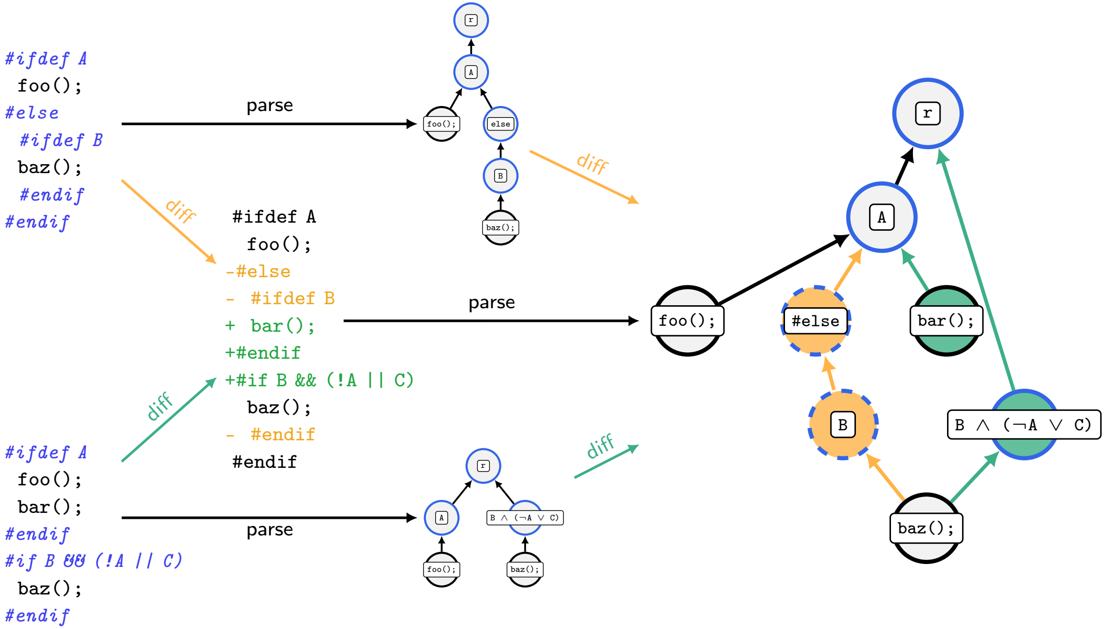
```
 #ifdef A
   foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
   baz();
-  #endif
 #endif
```

diff

```
#ifdef A
 foo();
 bar();
#endif
#if B && (!A || C)
 baz();
#endif
```

parse





```
#ifdef A
 foo();
#else
 #ifdef B
 baz();
 #endif
#endif
```

```
#ifdef A
  foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
  baz();
-  #endif
 #endif
```

```
#ifdef A
 foo();
 bar();
#endif
#if B && (!A || C)
 baz();
#endif
```

Topic: Unparsing Variation Trees and Diffs

Context

Variation trees and variation diffs constitute abstract syntax for variational software and changes to that software (i.e., patches and diffs). This abstract syntax enables powerful analyses and transformations.

Problem

Some tasks require concrete syntax instead of abstract syntax (e.g., applying a modified variation diff as a patch). However, our tool DiffDetective cannot yet unparse variation trees and diffs after potential modification to their initial text representation.

Task

Implement unparsing of variation trees and diffs and develop strategies for encountered challenges such as ambiguities due to information loss. (Invert the horizontal arrows in the previous image.) Test correctness and evaluate runtime performance empirically.

Useful but not Mandatory Experience

experience in OOP/Java, interest in compilers

Further Reading

- *DiffDetective website*
- *Classifying Edits to Variability in Source Code, Bittner et al., ESEC/FSE'22*
- *Variability-Aware Differencing with DiffDetective, Bittner et al., FSE'24*

Topic: Variability-Aware Patching

Context

Software variants might be developed in different branches or forks. With time, these branches might get out of sync.

Problem

Synchronizing certain changes (e.g., bugfixes) between variants can be challenging. Given a patch *p* to a variant *a*, applying *p* to another variant *b* might fail when *p* also changes code in *a* that is not in *b*, or when there is auxiliary code in *b* at the location of the patch.

Task

Develop + implement a patch mutation operator to adapt a patch on a source variant to a target variant. Identify when such an operator is correct. Evaluate runtime performance and patching quality empirically.
Hint: This topic partially subsumes the previous.

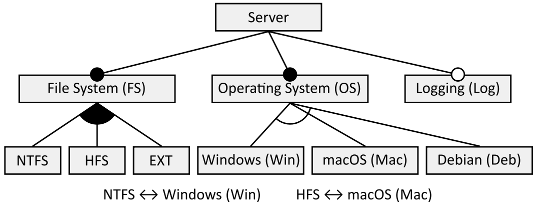
Useful but not Mandatory Experience

interest in formal methods and set theory, experience in OOP/Java

Further Reading

- see previous slide
- *Views on Edits to Variational Software, Bittner et al., SPLC'23*

Background: Feature-Model Sampling



$\Rightarrow$

$\{Server, FS, OS, HFS, Mac\}$
$\{Server, FS, OS, NTFS, EXT, Win\}$
$\{Server, FS, OS, EXT, Deb, Log\}$
...

- Create a representative list of configurations (e.g., for testing)
  - Random
  - Coverage criteria
  - ...

;

Topic: Incremental Partial T-Wise Sampling

Context

Full t-wise sample be may be too large to be tested completely. Typically, there are some fixed test capabilities per system (number of configurations that can be tested).

Problem

What configurations to choose when not all can be tested?

Task

Optimize the coverage achieved by a fixed number of configurations. Compare the results to existing solutions.

Useful but not Mandatory Experience

Knowledge about configurable software / software product lines, experience in Java

Further Reading

- *YASA: yet another sampling algorithm, Krieter et al., VaMoS'20*

Topic: Sampling with Evolving Presence Conditions

Context

Samples can be based on presence conditions of implementation artifacts. Like the code itself, presence conditions may also change when a product line evolves.

Problem

How often and to what degree do presence conditions change on average?

Task

Measure the rate at which presence conditions change during the history of multiple product lines. Compare the results and try to find correlations and trends.

Useful but not Mandatory Experience

Knowledge about configurable software / software product lines, experience in Java

Further Reading

- *T-Wise Presence Condition Coverage and Sampling for Configurable Systems , Krieter et al., arXiv 2022*
- *Variability-Aware Differencing with DiffDetective, Bittner et al., FSE'24*