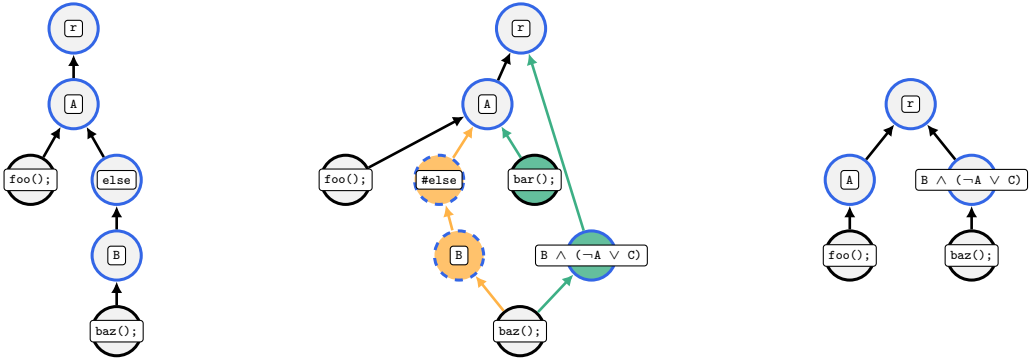


Classifying Edits to Variability in Source Code

Paul Bittner, Christof Tinnes, Alexander Schultheiß, Sören Viegener, Timo Kehrer, and Thomas Thüm | Nov 14, 2022

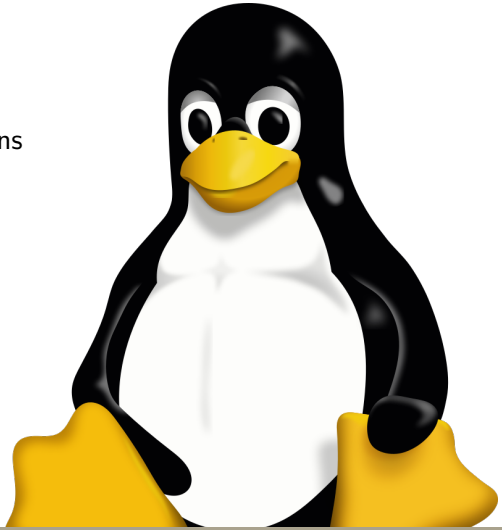
universität ulm

Software Comprises Massive Evolving Variability

$\geq$ **12,000**
configuration options

$\geq$ $\mathbf{10^{5000}}$
different variants

[2016]

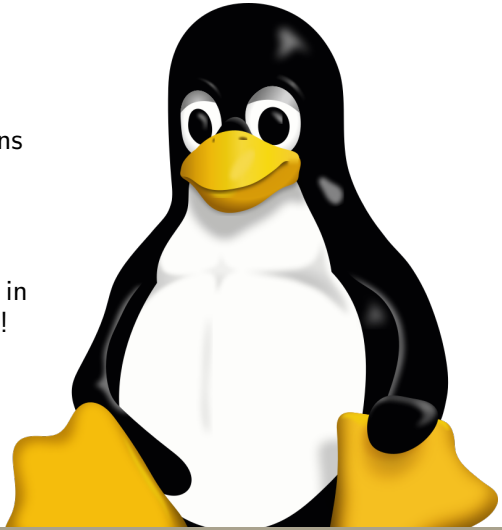


Software Comprises Massive Evolving Variability

$\geq$ **12,000**
configuration options

$\geq$ $\mathbf{10^{5000}}$
different variants
Only $\sim\mathbf{10^{80}}$ atoms in observable universe!

[2016]

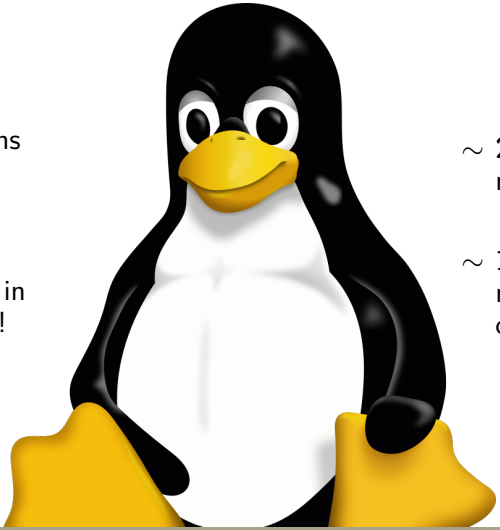


Software Comprises Massive Evolving Variability

$\geq$ **12,000**
configuration options

$\geq$ $\mathbf{10^{5000}}$
different variants
Only $\sim\mathbf{10^{80}}$ atoms in observable universe!

[2016]



$\sim$ **21,000**
new LOC / week

$\sim$ **15**
new configuration options / week

Variability via C Preprocessor

```
   static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
   if (gui.in_use)
   gui_mch_set_foreground();
#else
# ifdef MSWIN
   win32_set_foreground();
# endif
#endif
}
```

Vim Commit ab4cece.

Variability via C Preprocessor

```c
   static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
   if (gui.in_use)
   gui_mch_set_foreground();
#else
# ifdef MSWIN
   win32_set_foreground();
# endif
#endif
}
```

FEAT_GUI

```c
   static void
f_foreground(/* params */)
{
   if (gui.in_use)
   gui_mch_set_foreground();
}
```

Vim Commit ab4cece.

Variability via C Preprocessor

```
   static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
   if (gui.in_use)
   gui_mch_set_foreground();
#else
# ifdef MSWIN
   win32_set_foreground();
# endif
#endif
}
```

FEAT_GUI →

```
   static void
f_foreground(/* params */)
{
   if (gui.in_use)
   gui_mch_set_foreground();
}
```

¬FEAT_GUI, MSWIN →

```
   static void
f_foreground(/* params */)
{
   win32_set_foreground();
}
```

Vim Commit ab4cece.

Variability via C Preprocessor

```
   static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
   if (gui.in_use)
   gui_mch_set_foreground();
#else
# ifdef MSWIN
   win32_set_foreground();
# endif
#endif
}
```

FEAT_GUI →

```
   static void
f_foreground(/* params */)
{
   if (gui.in_use)
   gui_mch_set_foreground();
}
```

¬FEAT_GUI, MSWIN →

```
   static void
f_foreground(/* params */)
{
   win32_set_foreground();
}
```

¬FEAT_GUI, ¬MSWIN →

```
   static void
f_foreground(/* params */)
{
}
```

Vim Commit ab4cece.

Edits to Variability via C Preprocessor

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

Example simplified from Vim commit afde13b.

Edits to Variability via C Preprocessor

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

Commit afde13b →

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

Example simplified from Vim commit afde13b.

Edits to Variability via C Preprocessor

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

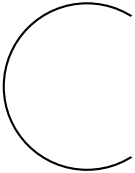
```
 #ifdef A
  foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
  baz();
-  #endif
 #endif
```

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

Example simplified from Vim commit afde13b.

Related Work on Edit Classification is . . .

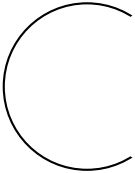
incomplete



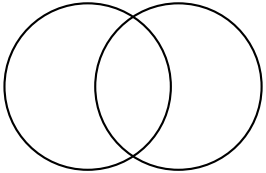
[Stănciulescu et al., 2016]
[Borba et al., 2012]
[Al-Hajjaji et al., 2016]
[Passos et al., 2016]

Related Work on Edit Classification is . . .

incomplete **or** **ambiguous**



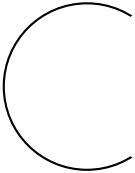
[Stănciulescu et al., 2016]
[Borba et al., 2012]
[Al-Hajjaji et al., 2016]
[Passos et al., 2016]



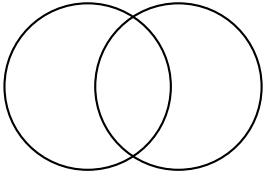
[Ji et al., 2015]
[Stănciulescu et al., 2016]

Related Work on Edit Classification is . . .

incomplete or **ambiguous** or **not automatable**



[Stănciulescu et al., 2016]
[Borba et al., 2012]
[Al-Hajjaji et al., 2016]
[Passos et al., 2016]



[Ji et al., 2015]
[Stănciulescu et al., 2016]

[Ji et al., 2015]

[Borba et al., 2012]

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

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
```

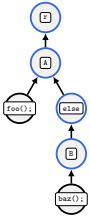
Research Goal:
complete, unambiguous, and automatable
model and classification

?

Classification

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

parse



First, we introduce *Variation Trees* as a model for variability in source code.

diff

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

Research Goal:
complete, unambiguous, and automatable model and classification

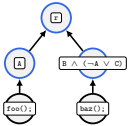
?

Classification

diff

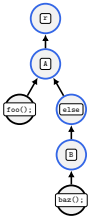
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



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

parse



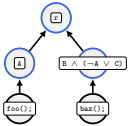
diff

```
  #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



First, we introduce *Variation Trees* as a model for variability in source code.

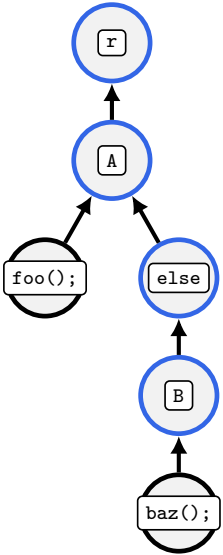
Research Goal:
complete, unambiguous, and automatable model and classification

?

Classification

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

parse

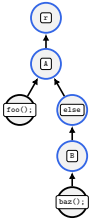


diff

```
#ifdef A
```

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

parse



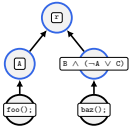
diff

```
  #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
```

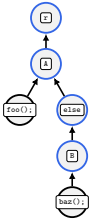
parse



Edits to variability become edits to Variation Trees

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

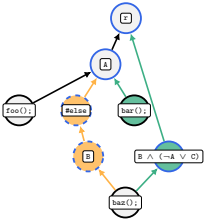
parse



diff

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

parse

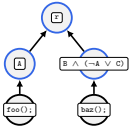


diff

diff

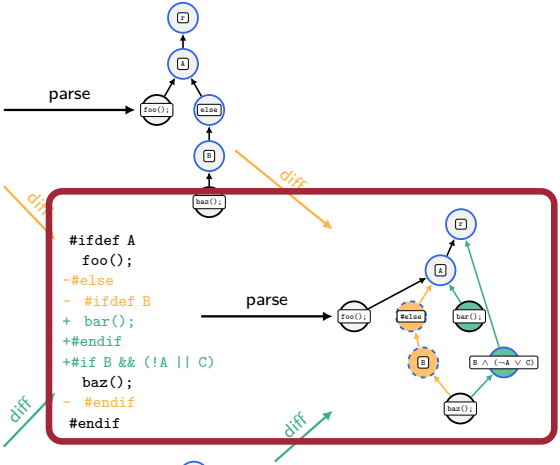
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



Edits to variability become edits to Variation Trees, for which we introduce *Variation Diffs*.

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

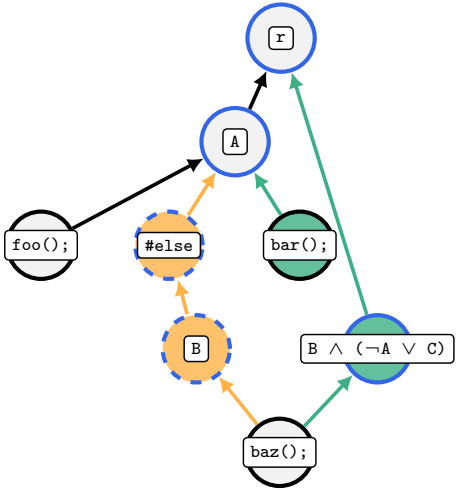



```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

Edits to variability become edits to Variation Trees, for which we introduce *Variation Diffs*.

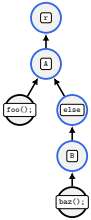
```
 #ifdef A
   foo();
-#else
-   #ifdef B
+   bar();
+#endif
+#if B && (!A || C)
   baz();
-   #endif
 #endif
```

parse



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

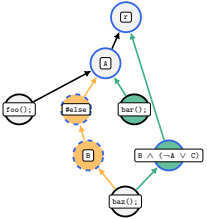
parse



diff

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

parse



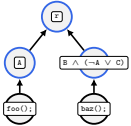
diff

diff

diff

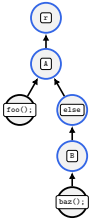
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



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

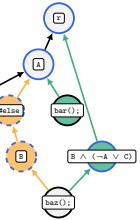
parse



diff

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

parse



classify

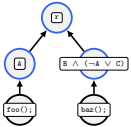
diff

diff

diff

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

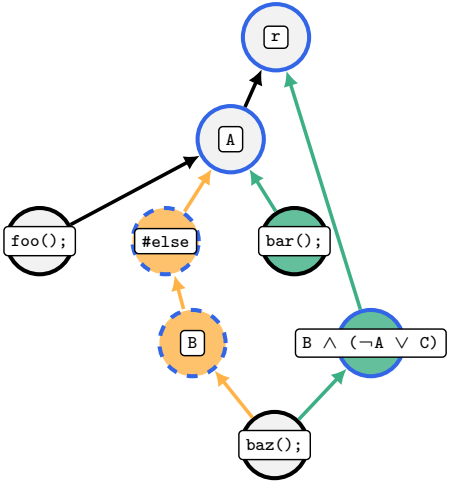
parse



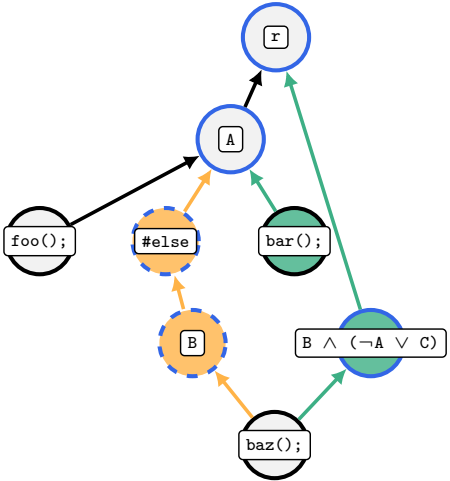
Classifying Edits to Variability in Source Code

reduces to

Classifying Structures in Variation Diffs

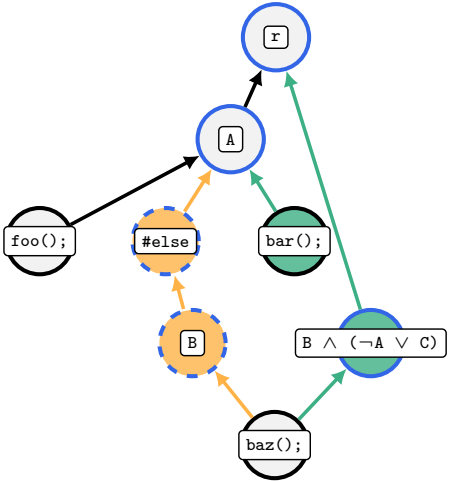


is unchanged



is unchanged

is added to feature A



is unchanged

is added to feature A

is moved
from $B \wedge \neg A$
to $B \wedge (\neg A \vee C)$

Classification $:=$ Set of Classes

Classification := Set of Classes

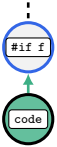
$class : \boxed{\text{code}} \rightarrow \{true, false\}$

Classification := Set of Classes

$class : \textcircled{\boxed{\texttt{code}}} \rightarrow \{true, false\}$

Example:

$$AddToPC(\textcircled{\boxed{\texttt{code}}}) := \text{added}(\textcircled{\boxed{\texttt{code}}}) \wedge \neg\, \text{added}(p_a(\textcircled{\boxed{\texttt{code}}}))$$



We define 9 classes.

AddWithMapping(c) := added(c) ∧ added$(M_a(c))$

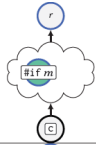
```
+ #if m
+   c
+ #endif
```

AddToPC(c) := added(c) ∧ ¬ added$(M_a(c))$

```
  #if m
+   c
  #endif
```

Specialization(c) := unchanged(c)
∧¬(PC$_b(c)$ ⊨ PC$_a(c)$)
∧ (PC$_a(c)$ ⊨ PC$_b(c)$)

```
+ #if m
    c
+ #endif
```



RemWithMapping(c) := removed(c) ∧ removed$(M_b(c))$

```
- #if m
-   c
- #endif
```

RemFromPC(c) := removed(c) ∧ ¬ removed$(M_b(c))$

```
  #if m
-   c
  #endif
```

Generalization(c) := unchanged(c)
∧ (PC$_b(c)$ ⊨ PC$_a(c)$)
∧¬(PC$_a(c)$ ⊨ PC$_b(c)$)

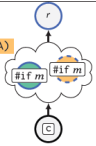
```
- #if m
    c
- #endif
```

Reconfiguration(c) := unchanged(c)
∧¬(PC$_b(c)$ ⊨ PC$_a(c)$)
∧¬(PC$_a(c)$ ⊨ PC$_b(c)$)

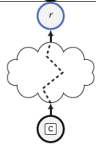
```
- #if m
+ #if m'
    c
  #endif
```

Refactoring(c) := unchanged(c)
∧(PC$_b(c)$ ⊨ PC$_a(c)$)
∧(PC$_a(c)$ ⊨ PC$_b(c)$)
∧($path_b(c) \neq path_a(c)$)

```
- #if A || (B && !A)
+ #if A || B
    c
  #endif
```



Untouched(c) := unchanged(c)
∧(PC$_b(c)$ ⊨ PC$_a(c)$)
∧(PC$_a(c)$ ⊨ PC$_b(c)$)
∧($path_b(c) = path_a(c)$)



We define 9 classes.

AddWithMapping(c) := added(c) ∧ added$(M_a(c))$

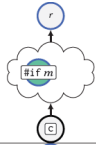
```
+ #if m
+   c
+ #endif
```

AddToPC(c) := added(c) ∧ ¬ added$(M_a(c))$

```
  #if m
+   c
  #endif
```

Specialization(c) := unchanged(c)
∧¬(PC$_b(c)$ ⊨ PC$_a(c)$)
∧ (PC$_a(c)$ ⊨ PC$_b(c)$)

```
+ #if m
  c
+ #endif
```



RemWithMapping(c) := removed(c) ∧ removed$(M_b(c))$

```
- #if m
-   c
- #endif
```

RemFromPC(c) := removed(c) ∧ ¬ removed$(M_b(c))$

```
  #if m
-   c
  #endif
```

Generalization(c) := unchanged(c)
∧ (PC$_b(c)$ ⊨ PC$_a(c)$)
∧¬(PC$_a(c)$ ⊨ PC$_b(c)$)

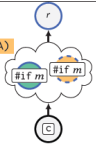
```
- #if m
  c
- #endif
```

Reconfiguration(c) := unchanged(c)
∧¬(PC$_b(c)$ ⊨ PC$_a(c)$)
∧¬(PC$_a(c)$ ⊨ PC$_b(c)$)

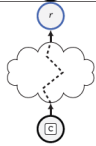
```
- #if m
+ #if m'
  c
  #endif
```

Refactoring(c) := unchanged(c)
∧(PC$_b(c)$ ⊨ PC$_a(c)$)
∧(PC$_a(c)$ ⊨ PC$_b(c)$)
∧($path_b(c) \neq path_a(c)$)

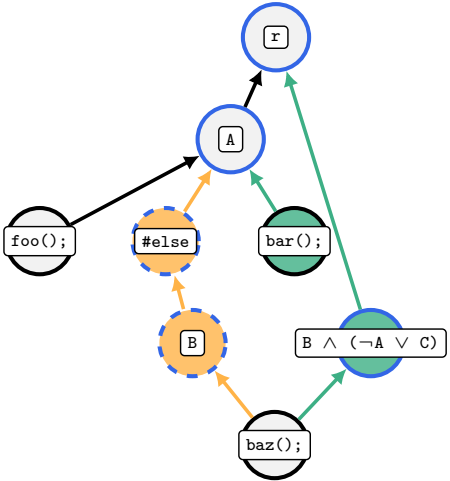
```
- #if A || (B && !A)
+ #if A || B
  c
  #endif
```



Untouched(c) := unchanged(c)
∧(PC$_b(c)$ ⊨ PC$_a(c)$)
∧(PC$_a(c)$ ⊨ PC$_b(c)$)
∧($path_b(c) = path_a(c)$)



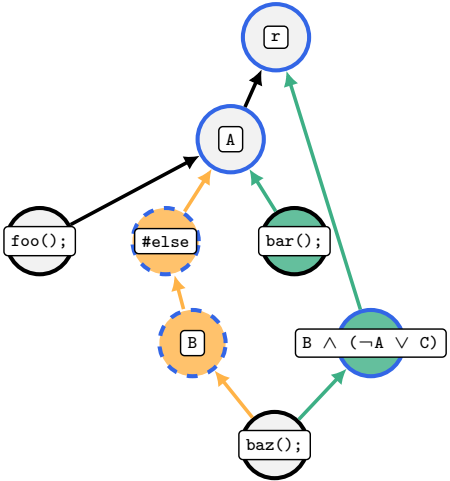
Custom classifications possible.



is unchanged

is added to feature A

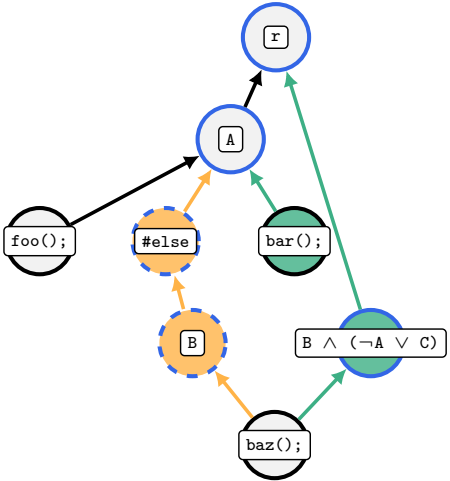
is moved
from $B \wedge \neg A$
to $B \wedge (\neg A \vee C)$



is unchanged

$AddToPC($  $) = true$

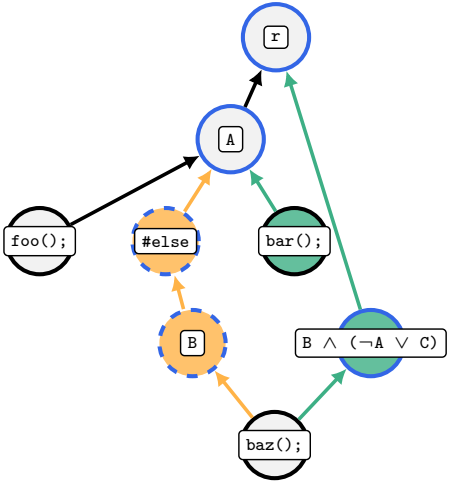
is moved
from $B \wedge \neg A$
to $B \wedge (\neg A \vee C)$



Untouched(`foo();`) = *true*

AddToPC(`bar();`) = *true*

`baz();` is moved
from $B \wedge \neg A$
to $B \wedge (\neg A \vee C)$



Untouched(foo();) = *true*

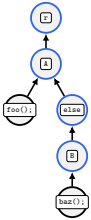
AddToPC(bar();) = *true*

Generalization(baz();) = *true*

because $B \wedge \neg A \models B \wedge (\neg A \vee C)$

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

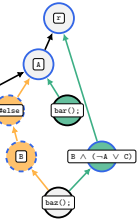
parse



diff

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

parse



classify

Untouched(foo();) = *true*

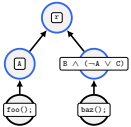
AddToPC(bar();) = *true*

Generalization(baz();) = *true*

diff

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

parse

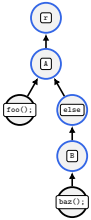


diff

Analytical Evaluation

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

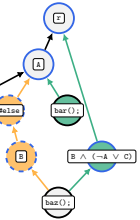
parse



diff

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

parse



classify

$Untouched($ foo(); $) = true$

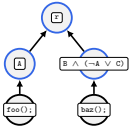
$AddToPC($ bar(); $) = true$

$Generalization($ baz(); $) = true$

diff

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

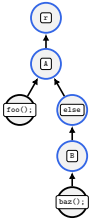
parse



Analytical Evaluation

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

parse



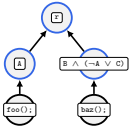
diff

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

diff

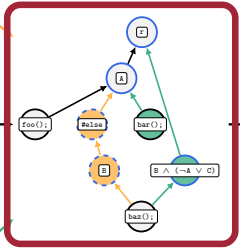
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



diff

parse



classify

Untouched(foo();) = *true*

AddToPC(bar();) = *true*

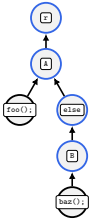
Generalization(baz();) = *true*

We prove completeness ◯ and soundness. □

Analytical Evaluation

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

parse



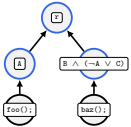
diff

```
  #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
```

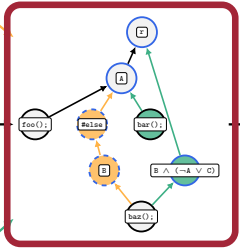
parse



diff

diff

parse



We prove completeness ◯ and soundness. □

classify

Untouched(foo();) = *true*

AddToPC(bar();) = *true*

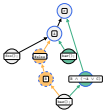
Generalization(baz();) = *true*

We prove completeness ◯ and unambiguity ◯◯. □

From Theory to Practice

44 open-source system histories (incl. 🐧)

Parse all Patches to



Success → Classify (RG_2) → Count Class Occurrences (RG_3)

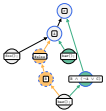
Failure → Inspect Failure (RG_1)

From Theory to Practice

44 open-source system histories (incl. 🐧)

1.7 million commits
45 million edits

Parse all Patches to



Success → Classify (RG_2) → Count Class Occurrences (RG_3)

Failure → Inspect Failure (RG_1)

From Theory to Practice

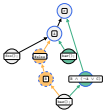
Measure Runtime (RG_4)

git

44 open-source system histories (incl. 🐧)

1.7 million commits
45 million edits

Parse all Patches to



Success

Failure

Classify (RG_2)

Count Class Occurrences (RG_3)

Inspect Failure (RG_1)

From Theory to Practice

RG_1 Variation Diffs Validate completeness of variation diffs.

From Theory to Practice

RG_1 Variation Diffs Validate completeness of variation diffs.

Result All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ $\Rightarrow$ ◯

From Theory to Practice

RG_1 Variation Diffs	Validate completeness of variation diffs.
Result	All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ ⇒ ◯
RG_2 Classification	Validate completeness and unambiguity of classification.

From Theory to Practice

RG_1	**Variation Diffs**	Validate completeness of variation diffs.
	Result	All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ $\Rightarrow$ ◯
RG_2	**Classification**	Validate completeness and unambiguity of classification.
	Result	All edits were assigned exactly one class. ✓ $\Rightarrow$ ◯ $\wedge$ ◯◯

From Theory to Practice

RG_1	**Variation Diffs**	Validate completeness of variation diffs.
	Result	All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ $\Rightarrow$ ◯
RG_2	**Classification**	Validate completeness and unambiguity of classification.
	Result	All edits were assigned exactly one class. ✓ $\Rightarrow$ ◯ $\wedge$ ◯◯
RG_3	**Relevancy**	Validate that our edit classes are relevant (i.e., all classes occur in practice).

From Theory to Practice

RG_1	**Variation Diffs**	Validate completeness of variation diffs.
	Result	All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ $\Rightarrow$ ◯
RG_2	**Classification**	Validate completeness and unambiguity of classification.
	Result	All edits were assigned exactly one class. ✓ $\Rightarrow$ ◯ $\wedge$ ◯◯
RG_3	**Relevancy**	Validate that our edit classes are relevant (i.e., all classes occur in practice).
	Result	All classes occur in practice (91,000 to 22 million occurrences). ✓

From Theory to Practice

RG_1	**Variation Diffs**	Validate completeness of variation diffs.
	Result	All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ $\Rightarrow$ ◯
RG_2	**Classification**	Validate completeness and unambiguity of classification.
	Result	All edits were assigned exactly one class. ✓ $\Rightarrow$ ◯ $\wedge$ ◯◯
RG_3	**Relevancy**	Validate that our edit classes are relevant (i.e., all classes occur in practice).
	Result	All classes occur in practice (91,000 to 22 million occurrences). ✓
RG_4	**Scalability**	Validate that edit classification can be automated and scales.

From Theory to Practice

RG_1	**Variation Diffs**	Validate completeness of variation diffs.
	Result	All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ ⇒ ◯
RG_2	**Classification**	Validate completeness and unambiguity of classification.
	Result	All edits were assigned exactly one class. ✓ ⇒ ◯ ∧ ◯◯
RG_3	**Relevancy**	Validate that our edit classes are relevant (i.e., all classes occur in practice).
	Result	All classes occur in practice (91,000 to 22 million occurrences). ✓
RG_4	**Scalability**	Validate that edit classification can be automated and scales.
	Result	99.89% of commits processed in < 1s with 7ms/commit as median. ✓ ⇒

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

diff

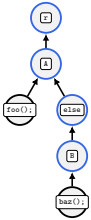
```
 #ifdef A
   foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
   baz();
-  #endif
 #endif
```

diff

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

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

parse



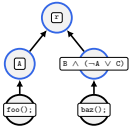
diff

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

diff

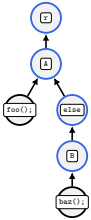
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



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

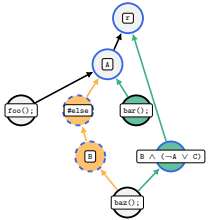
parse



diff

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

parse



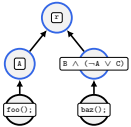
diff

diff

diff

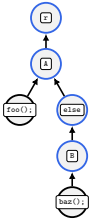
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



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

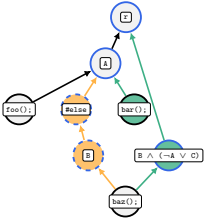
parse



diff

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

parse



classify

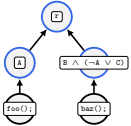
diff

diff

diff

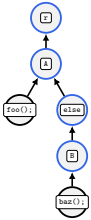
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



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

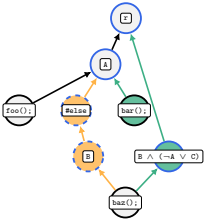
parse



diff

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

parse



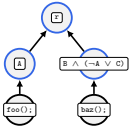
diff

diff

diff

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

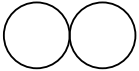
parse



classify

Complete

Unambiguous



Automated

Al-Hajjaji, M., Benduhn, F., Thüm, T., Leich, T., and Saake, G. (2016).
Mutation Operators for Preprocessor-Based Variability.
In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 81–88. ACM.

Borba, P., Teixeira, L., and Gheyi, R. (2012).
A Theory of Software Product Line Refinement.
Theoretical Computer Science, 455(0):2–30.

Ji, W., Berger, T., Antkiewicz, M., and Czarnecki, K. (2015).
Maintaining Feature Traceability with Embedded Annotations.
In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*, pages 61–70. ACM.

Passos, L., Teixeira, L., Dintzner, N., Apel, S., Wąsowski, A., Czarnecki, K., Borba, P., and Guo, J. (2016).
Coevolution of Variability Models and Related Software Artifacts.
Empirical Software Engineering (EMSE), 21(4).

Stănciulescu, S., Berger, T., Walkingshaw, E., and Wąsowski, A. (2016).
Concepts, Operations, and Feasibility of a Projection-Based Variation Control System.
In *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*, pages 323–333. IEEE.