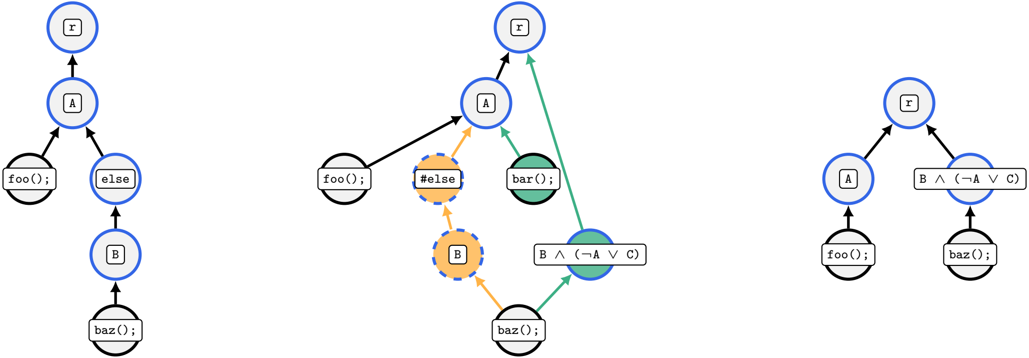


Classifying Edits to Variability in Source Code

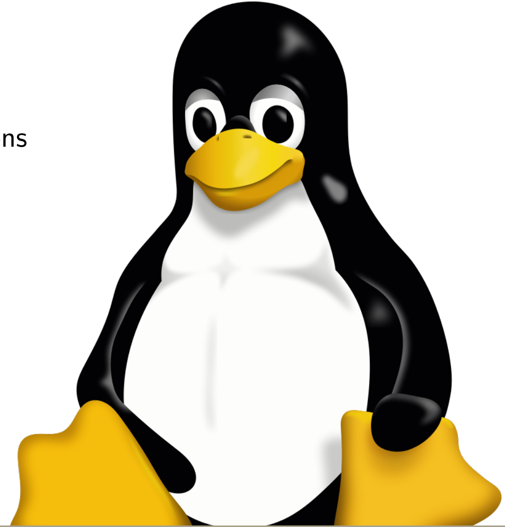
Paul Bittner, Christof Tinnes, Alexander Schultheiß, Sören Viegener, Timo Kehrer, and Thomas Thüm | Nov 14, 2022

Software Comprises Massive Evolving Variability

$\geq$ **4,000**
configuration options

$\geq$ **10^{725}**
different variants

[2007]



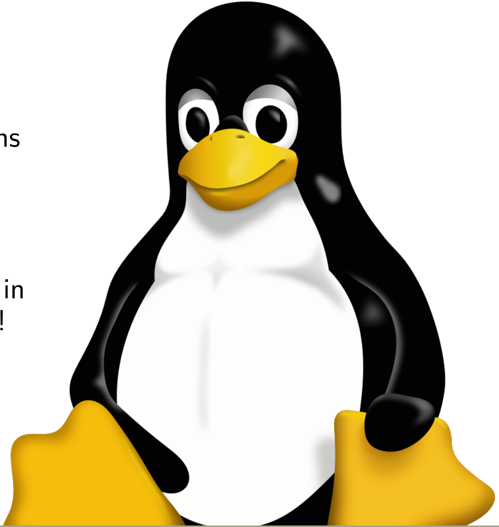
Software Comprises Massive Evolving Variability

$\geq$ **4,000**
configuration options

$\geq$ $\mathbf{10^{725}}$
different variants

Only $\sim\mathbf{10^{80}}$ atoms in observable universe!

[2007]



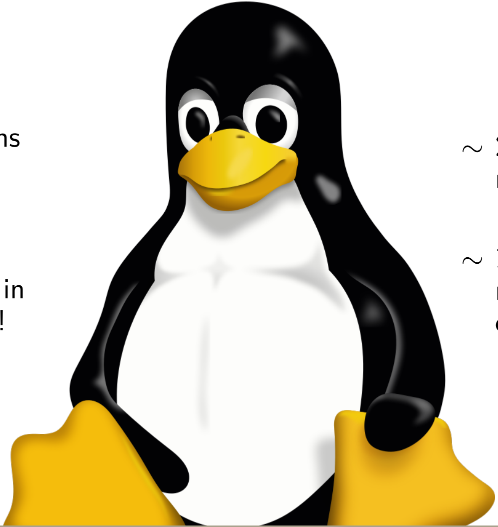
Software Comprises Massive Evolving Variability

$\geq$ **4,000**
configuration options

$\geq$ $\mathbf{10^{725}}$
different variants

Only $\sim\mathbf{10^{80}}$ atoms in observable universe!

[2007]



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

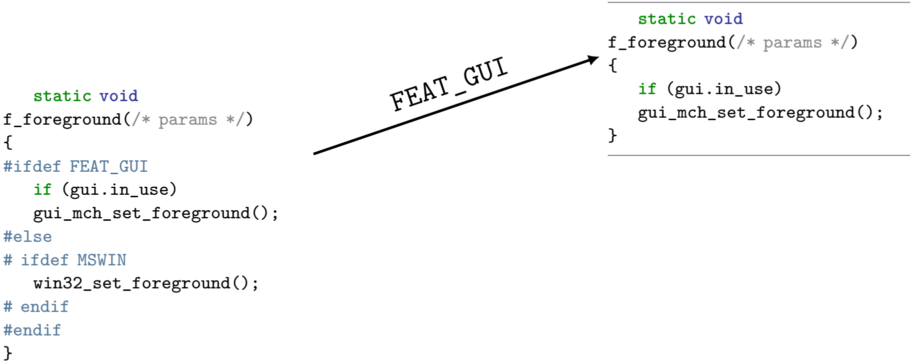
$\sim$ **15**
new configuration options / week

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

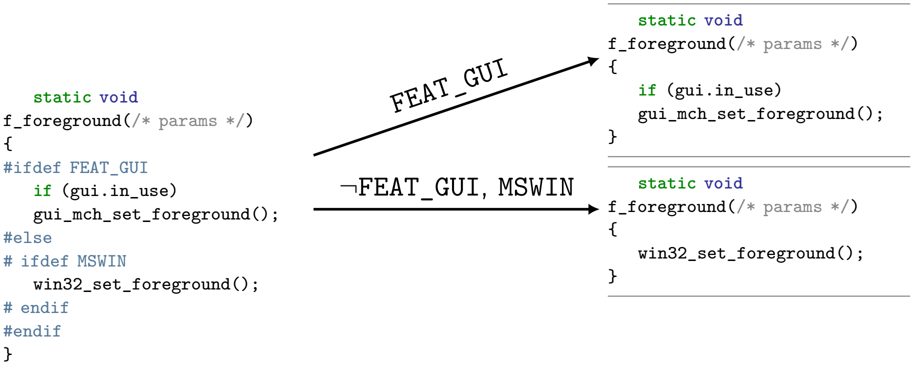


```
   static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
   if (gui.in_use)
   gui_mch_set_foreground();
#else
# ifdef MSWIN
   win32_set_foreground();
# endif
#endif
}
```

```
   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();
}
```

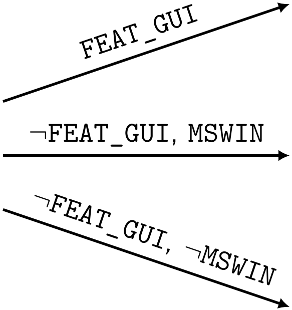
¬FEAT_GUI, MSWIN

```
   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

¬FEAT_GUI, MSWIN

¬FEAT_GUI, ¬MSWIN

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

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

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

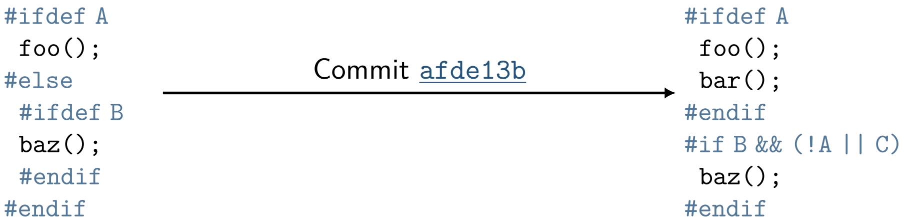
Vim Commit ab4cece.

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



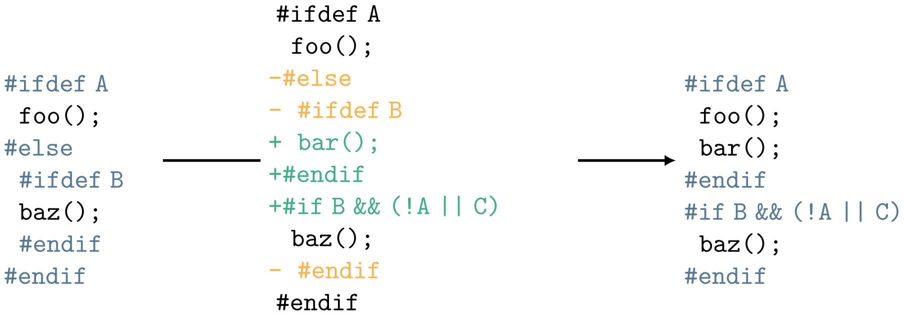
```
#ifdef A
 foo();
#else
 #ifdef B
 baz();
 #endif
#endif
```

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
```

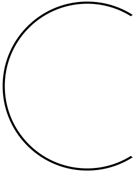
```
 #ifdef A
  foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
  baz();
-  #endif
 #endif
```

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

Example simplified from Vim commit afde13b.

Related Work on Edit Classification is . . .

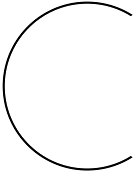
incomplete



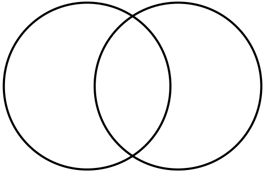
[Stănciulescu et al., 2016]
[Borba et al., 2012]
[Al-Hajjaji et al., 2016]
[Passos et al., 2016]

Related Work on Edit Classification is . . .

incomplete **or** **ambiguous**



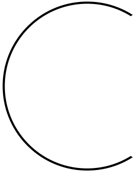
[Stănciulescu et al., 2016]
[Borba et al., 2012]
[Al-Hajjaji et al., 2016]
[Passos et al., 2016]



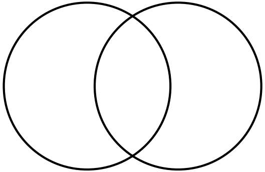
[Ji et al., 2015]
[Stănciulescu et al., 2016]

Related Work on Edit Classification is . . .

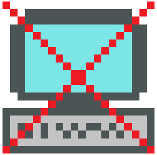
incomplete **or** **ambiguous** **or** **not automatable**



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

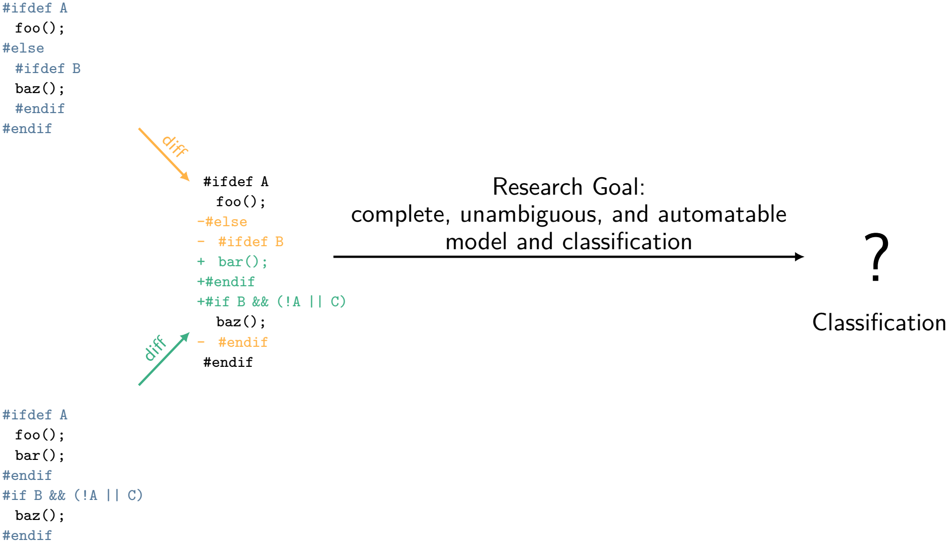


[Ji et al., 2015]
[Stănciulescu et al., 2016]



[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
```

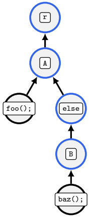
Research Goal:
complete, unambiguous, and automatable model and classification

?

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
```

diff

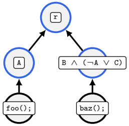
Research Goal:
complete, unambiguous, and automatable
model and classification

?

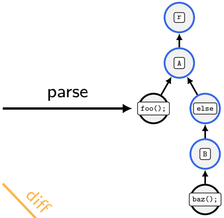
Classification

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

parse



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



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
```

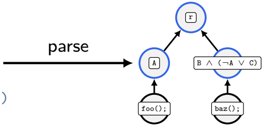
diff

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

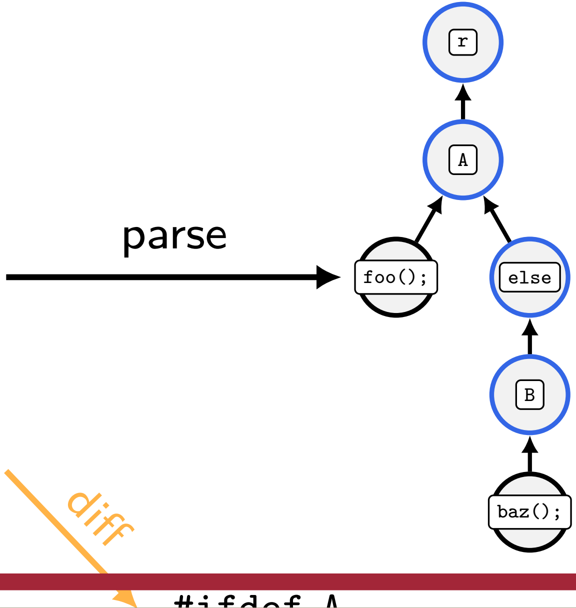
?

Classification

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



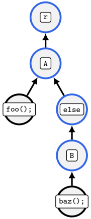
```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```



```
#ifdef A
```

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

parse



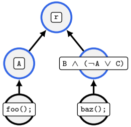
diff

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

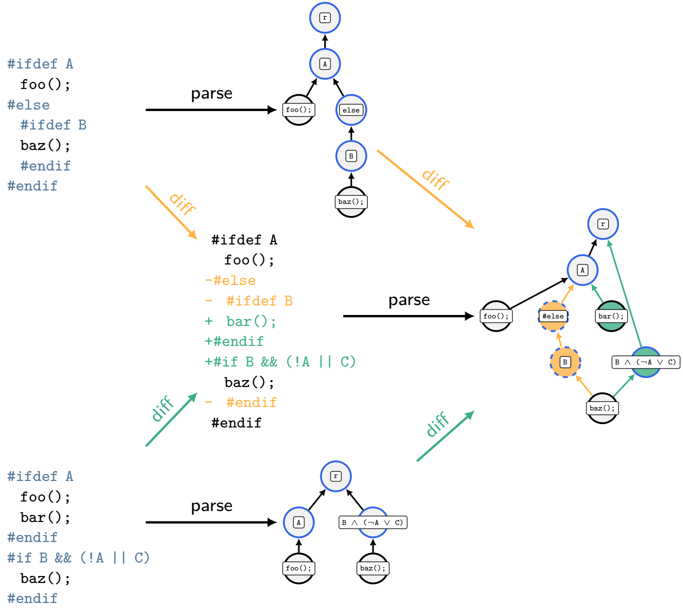
diff

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

parse



Edits to variability become edits to Variation Trees

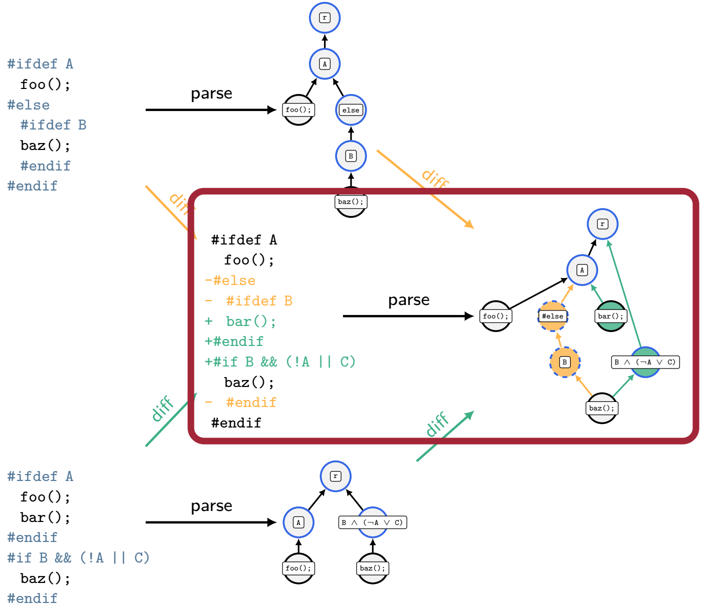


```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```

```
 #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
```

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



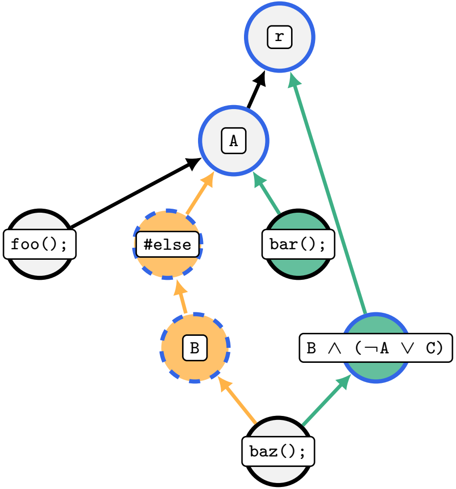
```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```

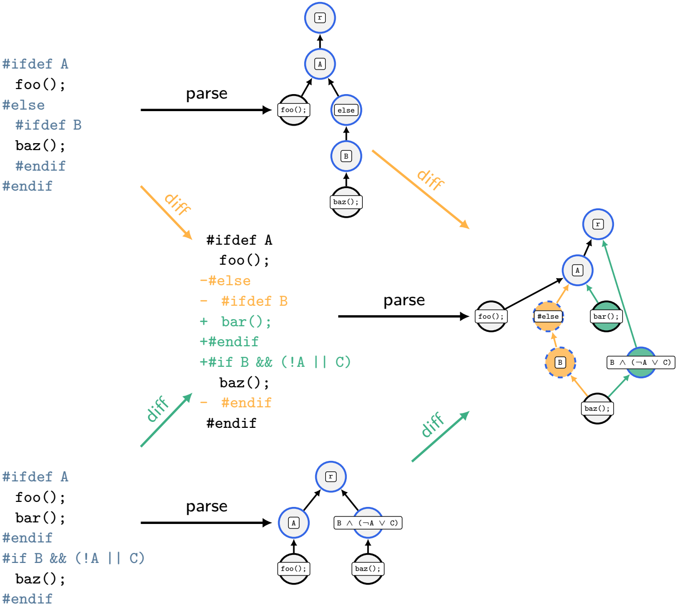
```
#ifdef A
  foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
  baz();
-  #endif
 #endif
```

```
#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
```





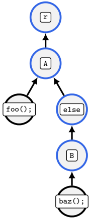
```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```

```
 #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

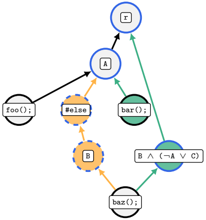


diff

diff

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

parse



classify

Classifying Edits to Variability in Source Code

reduces to

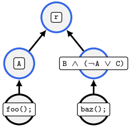
Classifying Structures in Variation Diffs

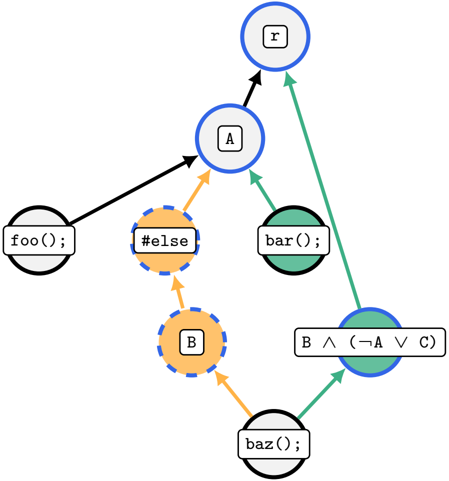
diff

diff

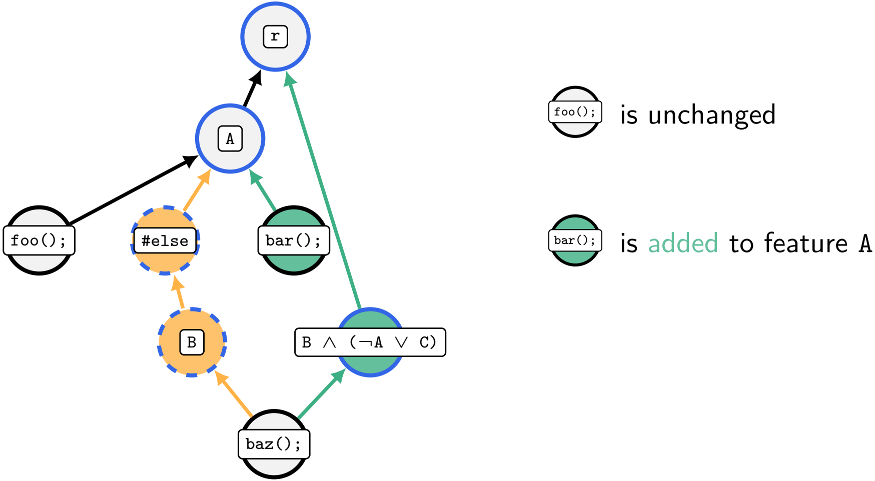
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

parse



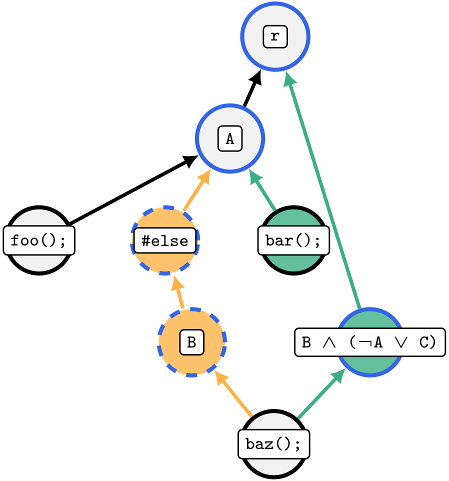


foo(); is unchanged



foo(); is unchanged

bar(); is added to feature A



`foo();` is unchanged

`bar();` is added to feature `A`

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

Classification $:=$ Set of Classes

Classification $\coloneqq$ Set of Classes

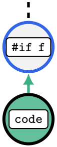
$$\mathit{class} : \boxed{\texttt{code}} \to \{\mathit{true}, \mathit{false}\}$$

Classification $:=$ Set of Classes

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

Example:

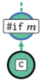
$$\mathit{AddToPC}(\boxed{\texttt{code}}) := \mathsf{added}(\boxed{\texttt{code}}) \land \neg\, \mathsf{added}(p_{\mathsf{a}}(\boxed{\texttt{code}}))$$



We define 9 classes.

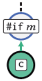
$\mathit{AddWithMapping}(c) :=$
$\mathrm{added}(c) \wedge \mathrm{added}(M_a(c))$

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



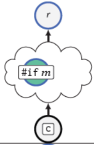
$\mathit{AddToPC}(c) :=$
$\mathrm{added}(c) \wedge \neg\,\mathrm{added}(M_a(c))$

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



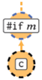
$\mathit{Specialization}(c) := \mathrm{unchanged}(c)$
$\wedge\neg(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))$
$\wedge\ (\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))$

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



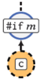
$\mathit{RemWithMapping}(c) :=$
$\mathrm{removed}(c) \wedge \mathrm{removed}(M_b(c))$

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



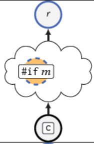
$\mathit{RemFromPC}(c) :=$
$\mathrm{removed}(c) \wedge \neg\,\mathrm{removed}(M_b(c))$

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



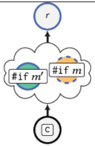
$\mathit{Generalization}(c) := \mathrm{unchanged}(c)$
$\wedge\ (\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))$
$\wedge\neg(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))$

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



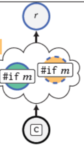
$\mathit{Reconfiguration}(c) := \mathrm{unchanged}(c)$
$\wedge\neg(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))$
$\wedge\neg(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))$

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

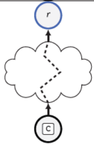


$\mathit{Refactoring}(c) := \mathrm{unchanged}(c)$
$\wedge(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))$
$\wedge(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))$
$\wedge(\mathit{path}_b(c) \neq \mathit{path}_a(c))$

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



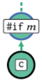
$\mathit{Untouched}(c) := \mathrm{unchanged}(c)$
$\wedge(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))$
$\wedge(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))$
$\wedge(\mathit{path}_b(c) = \mathit{path}_a(c))$



We define 9 classes.

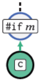
$$\mathit{AddWithMapping}(c) := \mathrm{added}(c) \land \mathrm{added}(M_a(c))$$

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



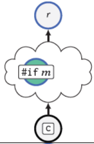
$$\mathit{AddToPC}(c) := \mathrm{added}(c) \land \neg\,\mathrm{added}(M_a(c))$$

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



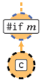
$$\begin{aligned}\mathit{Specialization}(c) := {} & \mathrm{unchanged}(c)\\ & \land\neg(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))\\ & \land\ (\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))\end{aligned}$$

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



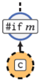
$$\mathit{RemWithMapping}(c) := \mathrm{removed}(c) \land \mathrm{removed}(M_b(c))$$

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



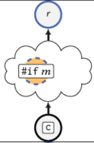
$$\mathit{RemFromPC}(c) := \mathrm{removed}(c) \land \neg\,\mathrm{removed}(M_b(c))$$

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



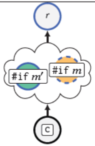
$$\begin{aligned}\mathit{Generalization}(c) := {} & \mathrm{unchanged}(c)\\ & \land\ (\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))\\ & \land\neg(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))\end{aligned}$$

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



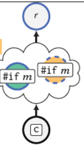
$$\begin{aligned}\mathit{Reconfiguration}(c) := {} & \mathrm{unchanged}(c)\\ & \land\neg(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))\\ & \land\neg(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))\end{aligned}$$

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

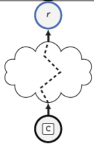


$$\begin{aligned}\mathit{Refactoring}(c) := {} & \mathrm{unchanged}(c)\\ & \land(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))\\ & \land(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))\\ & \land(\mathit{path}_b(c) \neq \mathit{path}_a(c))\end{aligned}$$

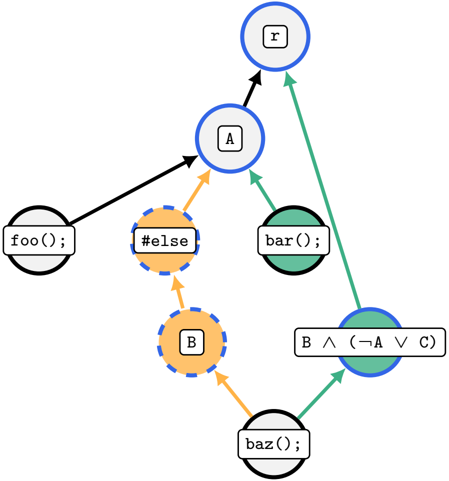
```
- #if A || (B && !A)
+ #if A || B
  c
  #endif
```



$$\begin{aligned}\mathit{Untouched}(c) := {} & \mathrm{unchanged}(c)\\ & \land(\mathrm{PC}_b(c) \models \mathrm{PC}_a(c))\\ & \land(\mathrm{PC}_a(c) \models \mathrm{PC}_b(c))\\ & \land(\mathit{path}_b(c) = \mathit{path}_a(c))\end{aligned}$$



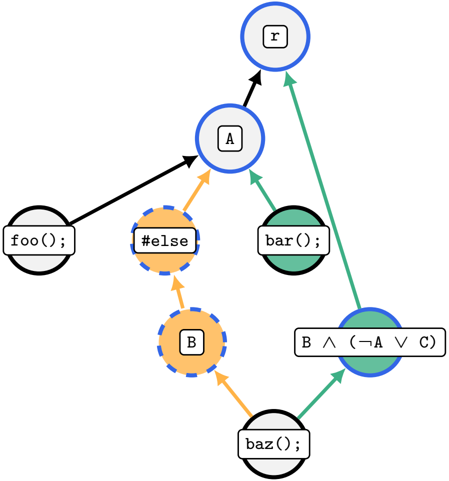
Custom classifications possible.



foo(); is unchanged

bar(); is added to feature A

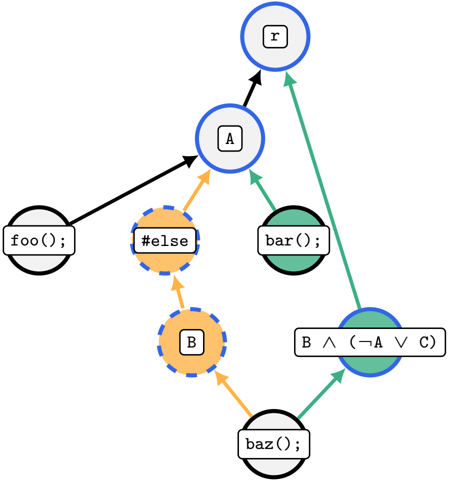
baz(); is moved
from $B \land \ \neg A$
to $B \land (\neg A \lor C)$



foo(); is unchanged

AddToPC(bar();) = *true*

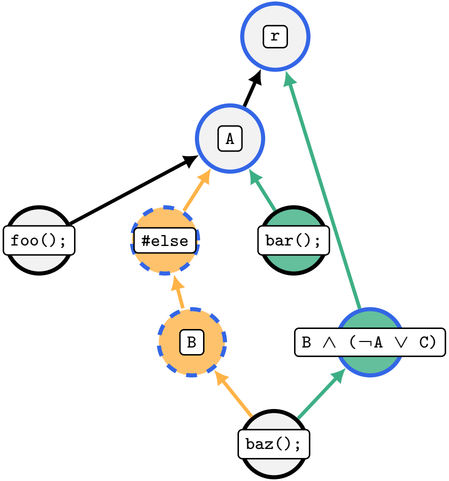
baz(); is moved
from $B \wedge \neg A$
to $B \wedge (\neg A \vee C)$



$\mathit{Untouched}($ foo(); $) = \mathit{true}$

$\mathit{AddToPC}($ bar(); $) = \mathit{true}$

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

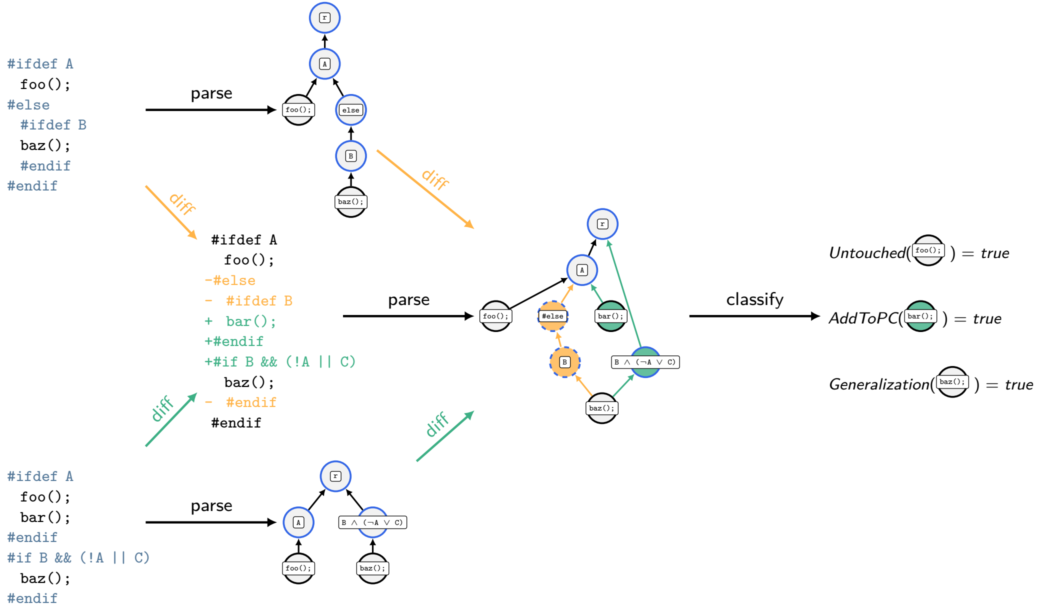


$\mathit{Untouched}(\texttt{foo();}) = \mathit{true}$

$\mathit{AddToPC}(\texttt{bar();}) = \mathit{true}$

$\mathit{Generalization}(\texttt{baz();}) = \mathit{true}$

because $B \land \neg A \models B \land (\neg A \lor C)$



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

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

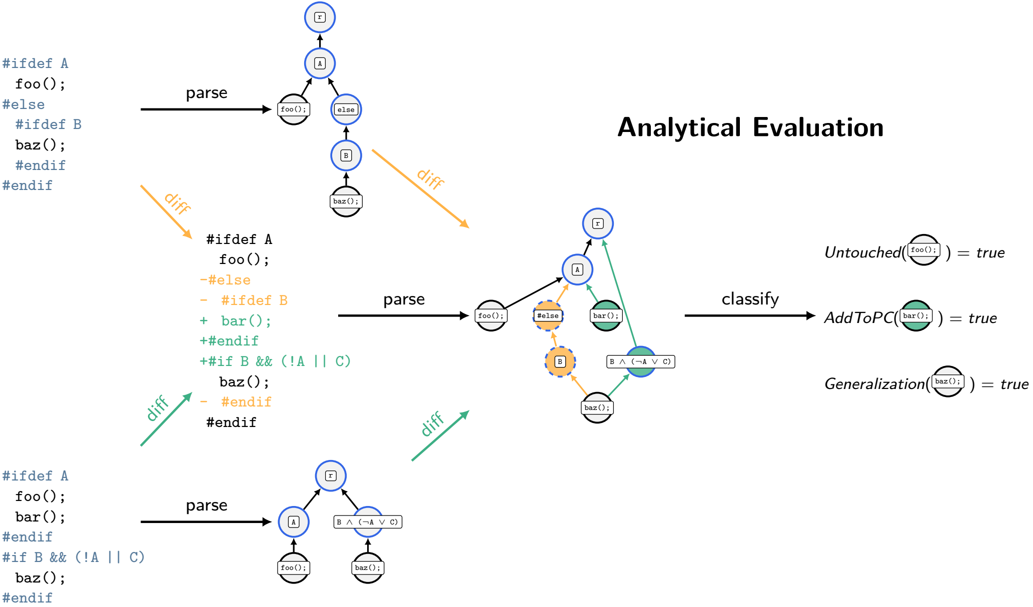
```
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif
```

Untouched(foo();) = *true*

AddToPC(bar();) = *true*

Generalization(baz();) = *true*

Analytical Evaluation

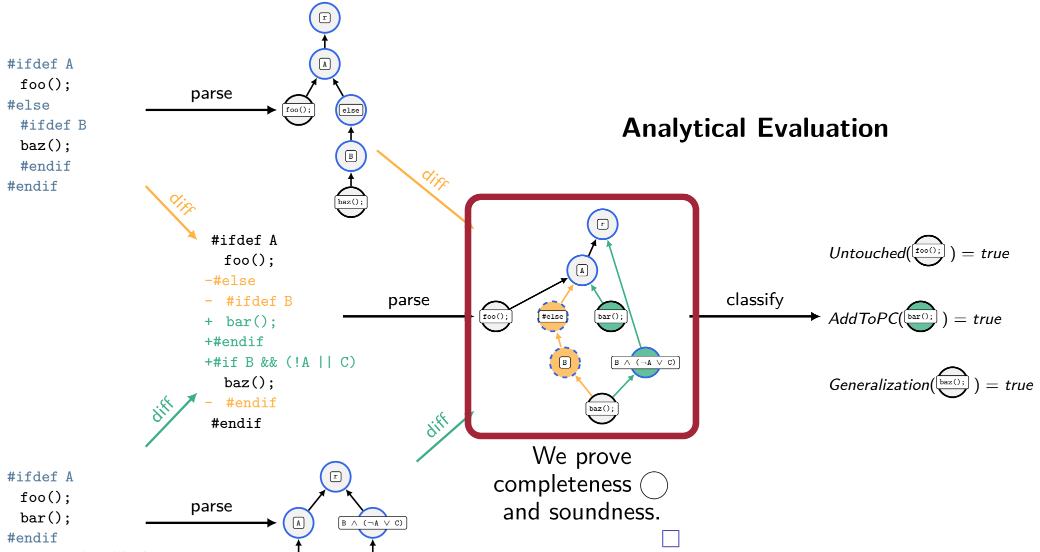


```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```

```
#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
```

Analytical Evaluation



```
#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
```

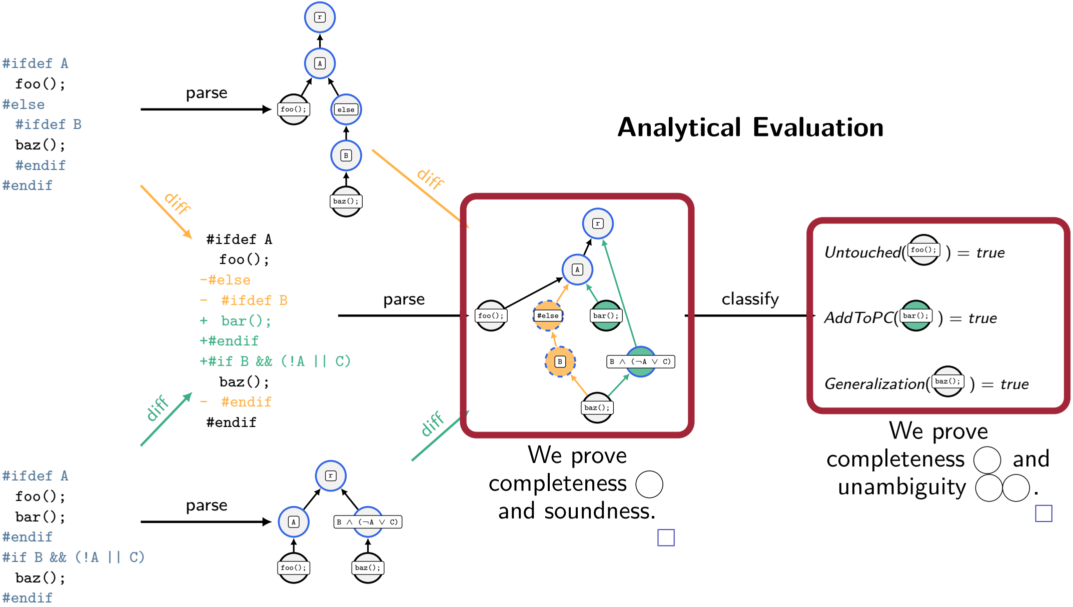
$\mathit{Untouched}(\text{foo();}) = \mathit{true}$

$\mathit{AddToPC}(\text{bar();}) = \mathit{true}$

$\mathit{Generalization}(\text{baz();}) = \mathit{true}$

We prove completeness ○ and soundness. □

Analytical Evaluation



```
#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
```

Untouched(foo();) = *true*

AddToPC(bar();) = *true*

Generalization(baz();) = *true*

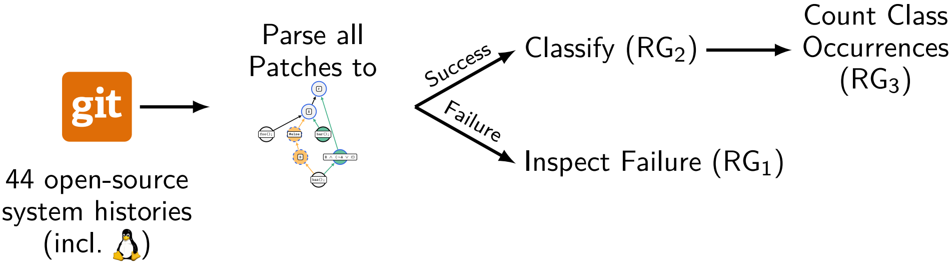
We prove completeness ○ and soundness. □

We prove completeness ○ and unambiguity ○○. □

From Theory to Practice

Artifacts Available acm V1.1

Artifacts Evaluated acm Reusable V1.1



44 open-source system histories (incl. 🐧)

Parse all Patches to

Success

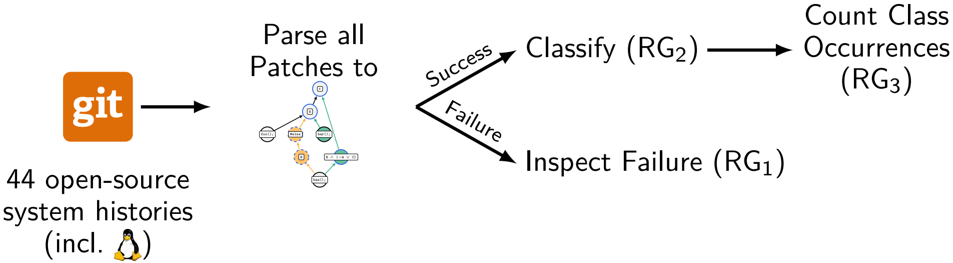
Failure

Classify (RG_2)

Count Class Occurrences (RG_3)

Inspect Failure (RG_1)

From Theory to Practice



44 open-source system histories (incl. 🐧)

Parse all Patches to

Success

Failure

Classify (RG_2)

Count Class Occurrences (RG_3)

Inspect Failure (RG_1)

1.7 million commits
45 million edits

From Theory to Practice

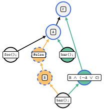
Measure Runtime (RG_4)

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%). ✓ ⇒ ○

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%). ✓ ⇒ ○

RG$_2$ Classification Validate completeness and unambiguity of classification.

Result All edits were assigned exactly one class. ✓ ⇒ ○ ∧ ○○

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%). ✓ ⇒ ○

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). ✓

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.

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.

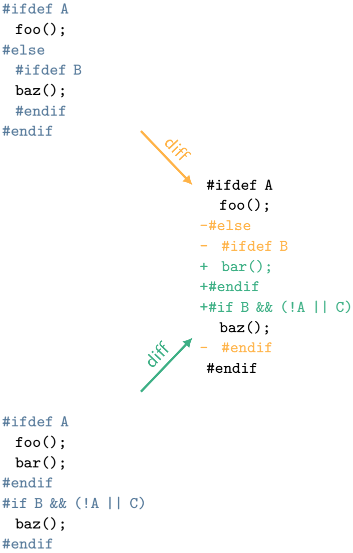
Result All edits were assigned exactly one class. ✓ ⇒ ○ ∧ ○○

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. ✓ ⇒ 💻



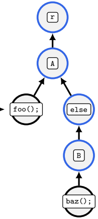
```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```

```
 #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



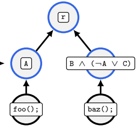
diff

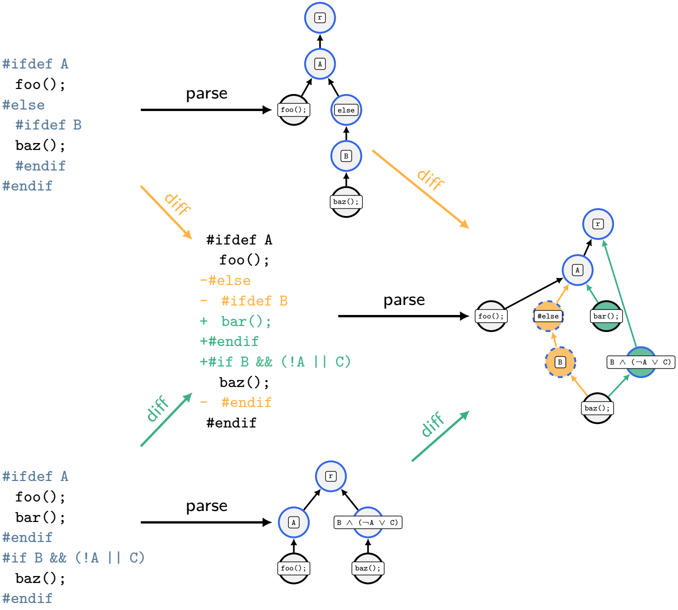
```
 #ifdef A
  foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
  baz();
-  #endif
 #endif
```

diff

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

parse

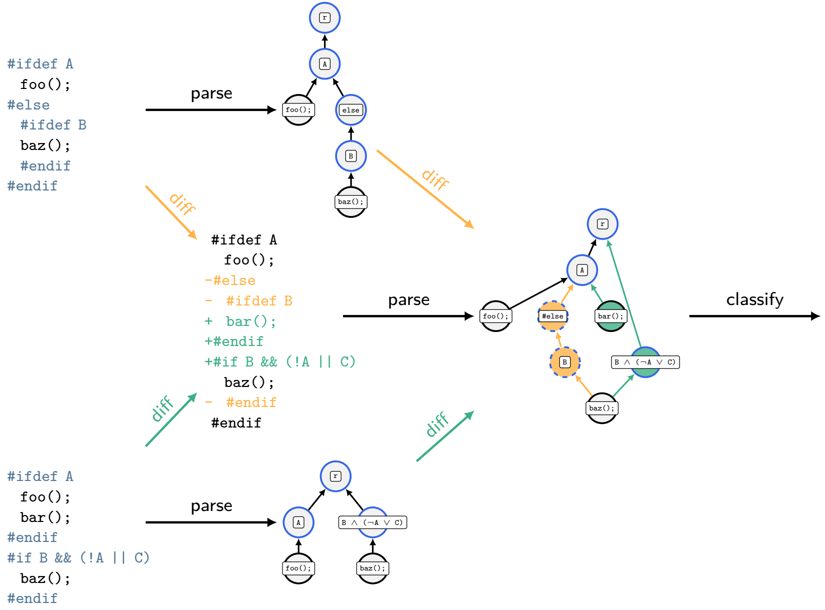




```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```

```
 #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
```



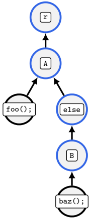
```
#ifdef A
 foo();
#else
 #ifdef B
 baz();
 #endif
#endif
```

```
 #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

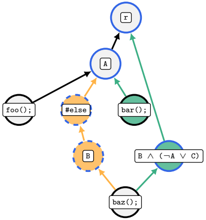


diff

diff

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

parse



classify

Complete

Unambiguous

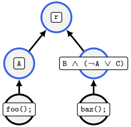
Automated

diff

diff

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

parse



Backup Slides

Use Cases

Test Case Evolution

Code Understanding

Incremental Analyses

Variation Control

Managed Clone-and-Own

Detect Unintended Changes

. . .

Software projects are heterogeneous

Source Code

Build Files

Other Resources

All identifiable *artifacts* within a software project might be subject to variability.

Variability comes in many forms

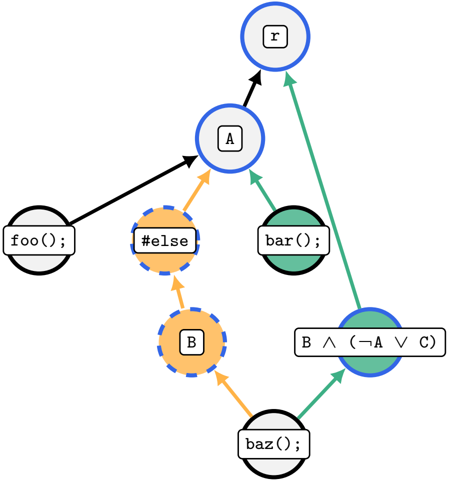
```
#ifdef A
  foo();
#else
  #ifdef B
  baz();
  #endif
#endif
```

Variability Annotations
(in different dialects)

Frameworks

Language Extensions

Commonality: Each artifact is identified with its configuration options.



We prove that variation diffs are

complete (i.e., any edit to a variation tree can be described as a variation diff) and

sound (i.e., any variation diff describes an edit to a variation tree)

□

giving us a complete and sound model for edits to variability.

- 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.