Identifying change patterns in software history

Jason Dagit | Galois, Inc

Motivation

Tools to detect changes exist.

For example, traditional line-based diff:

- Pro: diff is very general and programming language agnostic
- Con: diff is not structurally aware:

```
if( foo ){
  bar;
}
```

```
if( foo )
{
  bar;
}
```

We need tools for interpreting changes.

galois

Motivation

Common looping pattern with loop counter initialized to zero:

```
for (□ = 0; □ < □; □) {
    □
}
```

We also want to see how source code *changes*.

galois

Example from Clojure: Related edits

Our tool found these related edits:

PersistentArrayMap.java

```
 public Object kvreduce(IFn f, Object init){
     for(int i=0;i < array.length;i+=2){
         init = f.invoke(init, array[i], array[i+1]);
-            if(RT.isReduced(init))
-                return ((IDeref)init).deref();
         }
     return init;
 }
```

PersistentHashMap.java

```
 public Object kvreduce(IFn f, Object init){
-    for(INode node : array){
-        if(node != null){
+    for(INode node : array)
+        {
+        if(node != null)
             init = node.kvreduce(f,init);
-                if(RT.isReduced(init))
-                    return ((IDeref)init).deref();
-                }
-            }
+        }
     return init;
 }
```

galois

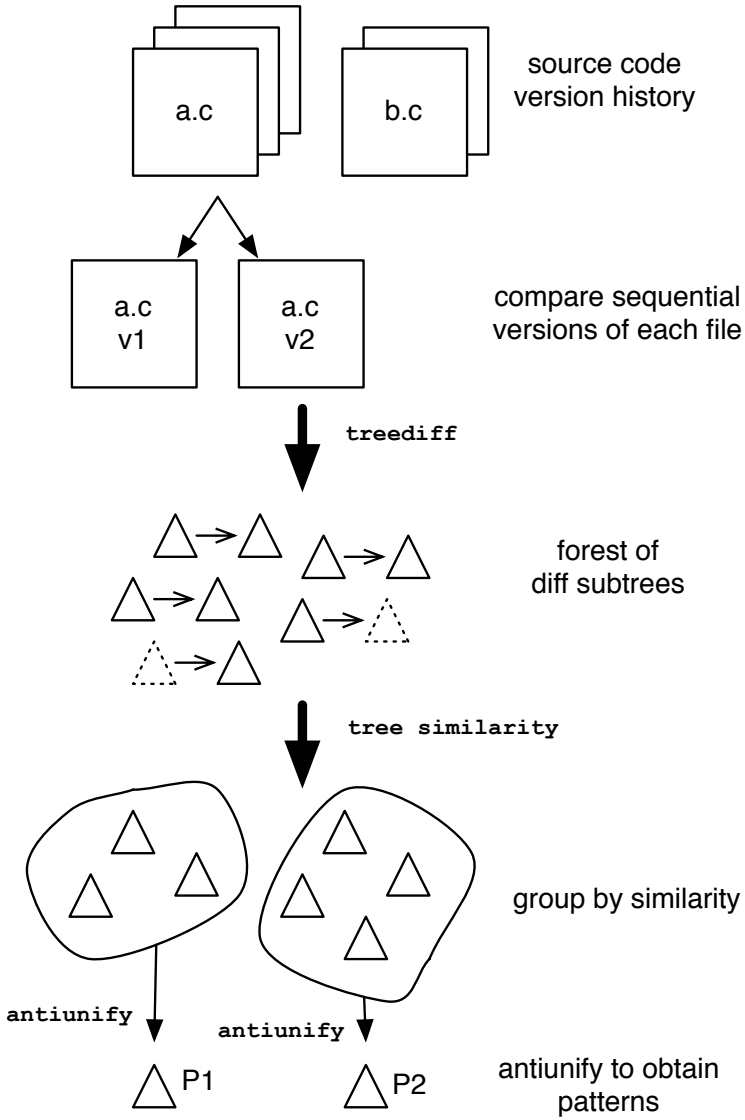
Approach

Key Idea: We can find *structural patterns* by generalizing *sufficiently similar* difference trees.

- Difference trees computed using structural diff of AST
- Similarity is measured using a tree edit distance score
- Generalization is accomplished through *antiunification*

galois

Workflow



galois

```
i++;
```

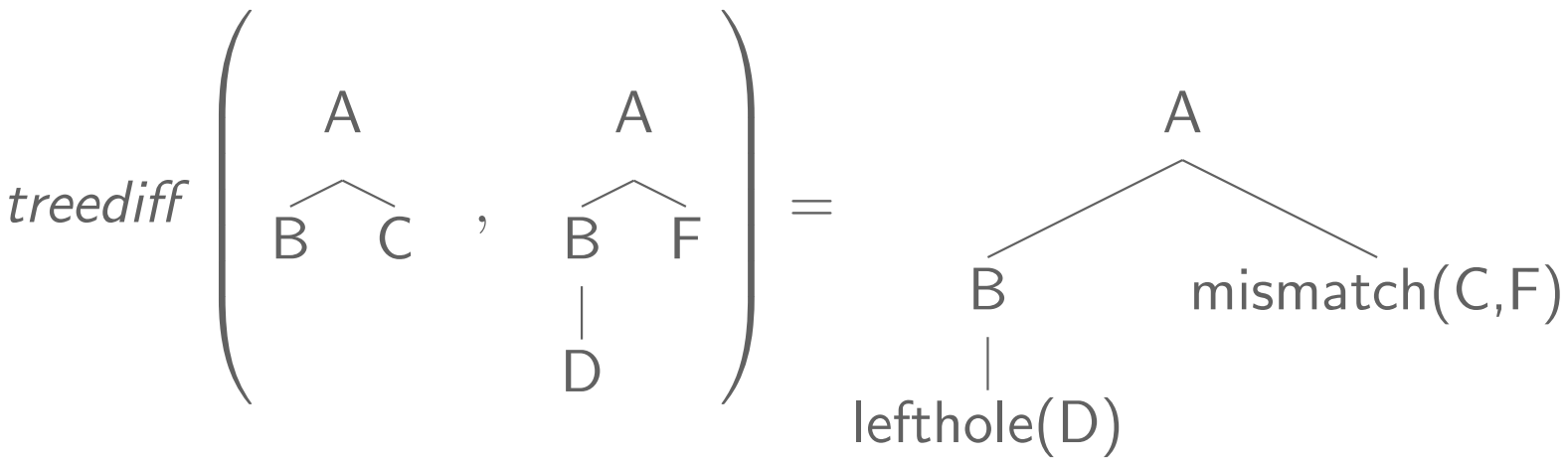
```
AAppl "ExpStmt"
  [AAppl "PostIncrement"
    [AAppl "ExpName"
      [AAppl "Name"
        [AList
          [AAppl "Ident" [AAppl "\"i\"" []]]]]]]
```

Generic tree structure—programming language agnostic.

Easy to modify parsers to generate ATerms.

galois

Structural diff

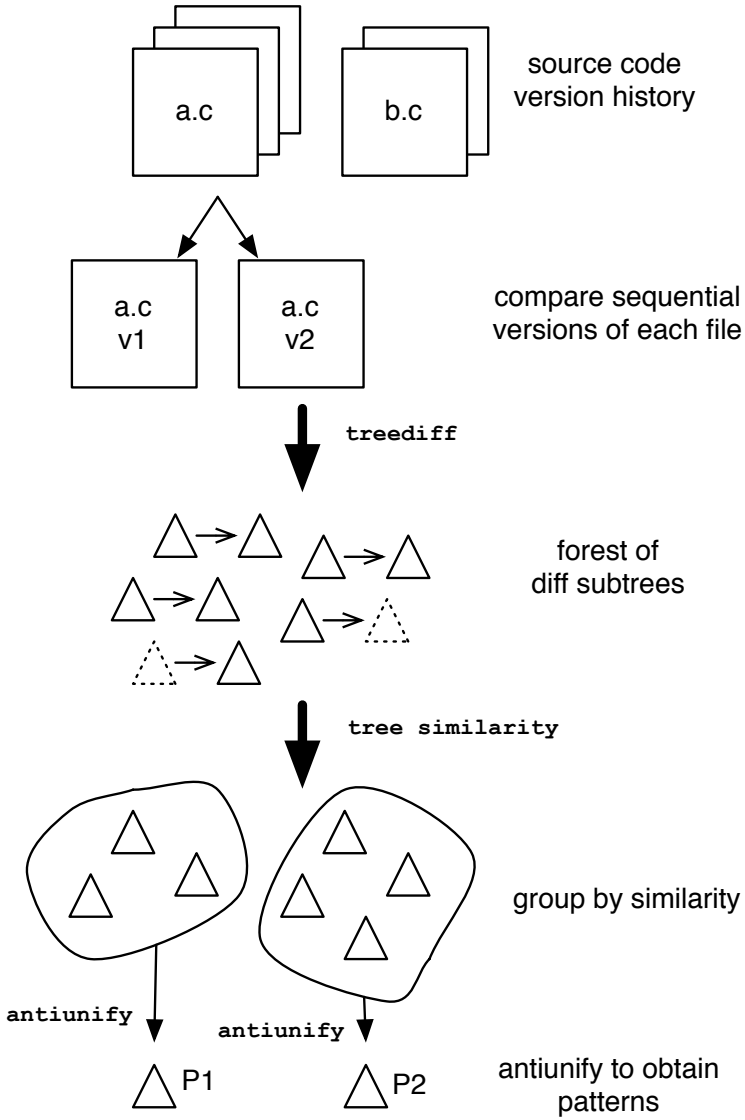


Keep just the differences with a bit of context:

Output also gives us an edit distance.

galois

Workflow



galois

Similarity grouping

We define the similarity score by:

$$\Delta(t_a, t_b) := \frac{min(d(t_a, t_b), d(t_b, t_a))}{max(|t_a|, |t_b|)}$$

where d is the tree edit distance score.

Similarity matrix D given by $D_{ij} = \Delta(t_i, t_j)$.

Given threshold $\tau \in [0, 1]$ we say t_i and t_j are similar if $D_{ij} \geq \tau$.

Group trees such that all elements in the group are within τ.

galois

ANTLR similarity groups with $\tau = 0.01$

10 similarity groups from ANTLR source, when $\tau = 0.01$:

7 are patterns:

```
□;

if( □ ) □;

if( □ ) { □ } □;

return □;

for( □ □ : □ ) □;

for( □ = □; □ < □; □ ) □;

throw RuntimeException( □ + □ );
```

galois

ANTLR similarity groups with $\tau = 0.01$

3 are constants (no □s):

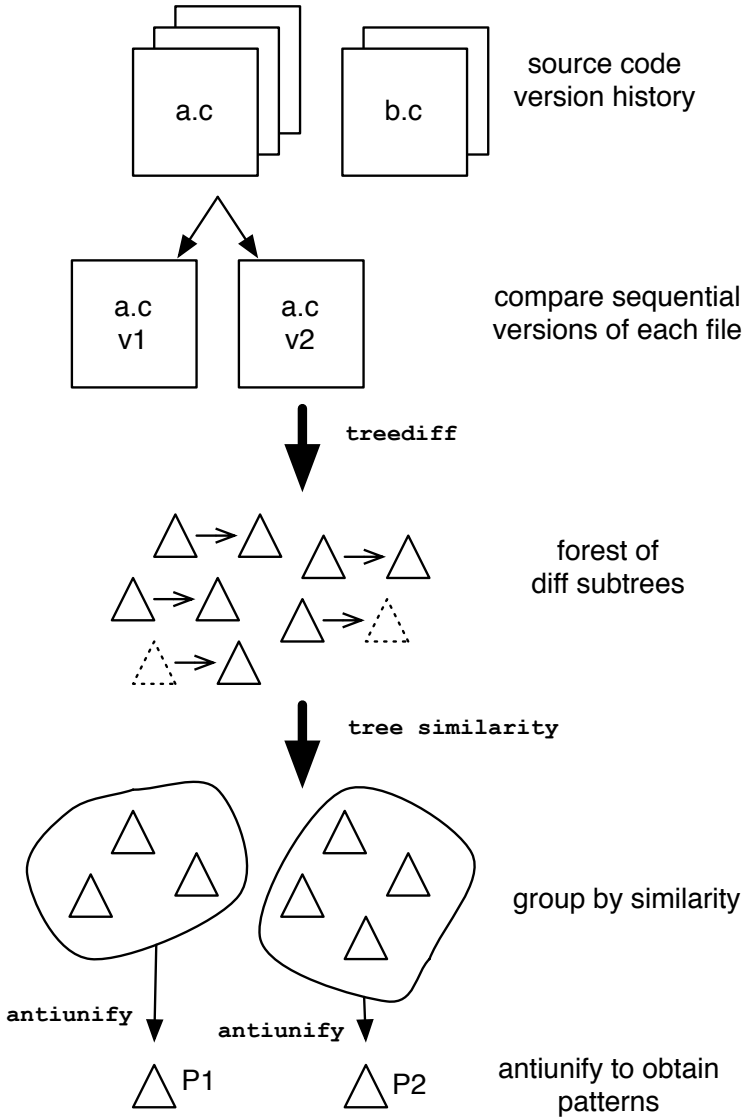
```
try {
  walker.grammarSpec();
} catch (RecognitionException re){
  ErrorManager.internalError("bad grammar AST structure",re);
}
```

```
while(sp !=
      StackLimitedNFAToDFAConverter.NFA_EMPTY_STACK_CONTEXT)
{
  n++;
  sp = sp.parent;
}
```

```
switch(gtype) {
  case ANTLRParser.LEXER_GRAMMAR:
    return legalLexerOptions.contains(key);
  case ANTLRParser.PARSER_GRAMMAR:
    return legalParserOptions.contains(key);
  case ANTLRParser.TREE_GRAMMAR:
    return legalTreeParserOptions.contains(key);
  default:
    return legalParserOptions.contains(key);
}
```

galois

Workflow



galois

$$au\left(\begin{array}{c} \text{A} \\ \text{B}\quad\text{C} \end{array}\ ,\ \begin{array}{c} \text{A} \\ \text{B}\quad\text{F} \\ \text{D} \end{array}\right) = \left(\begin{array}{c} \text{A} \\ \square_1\quad\square_2 \end{array}\ , subst_l, subst_r\right)$$

where,

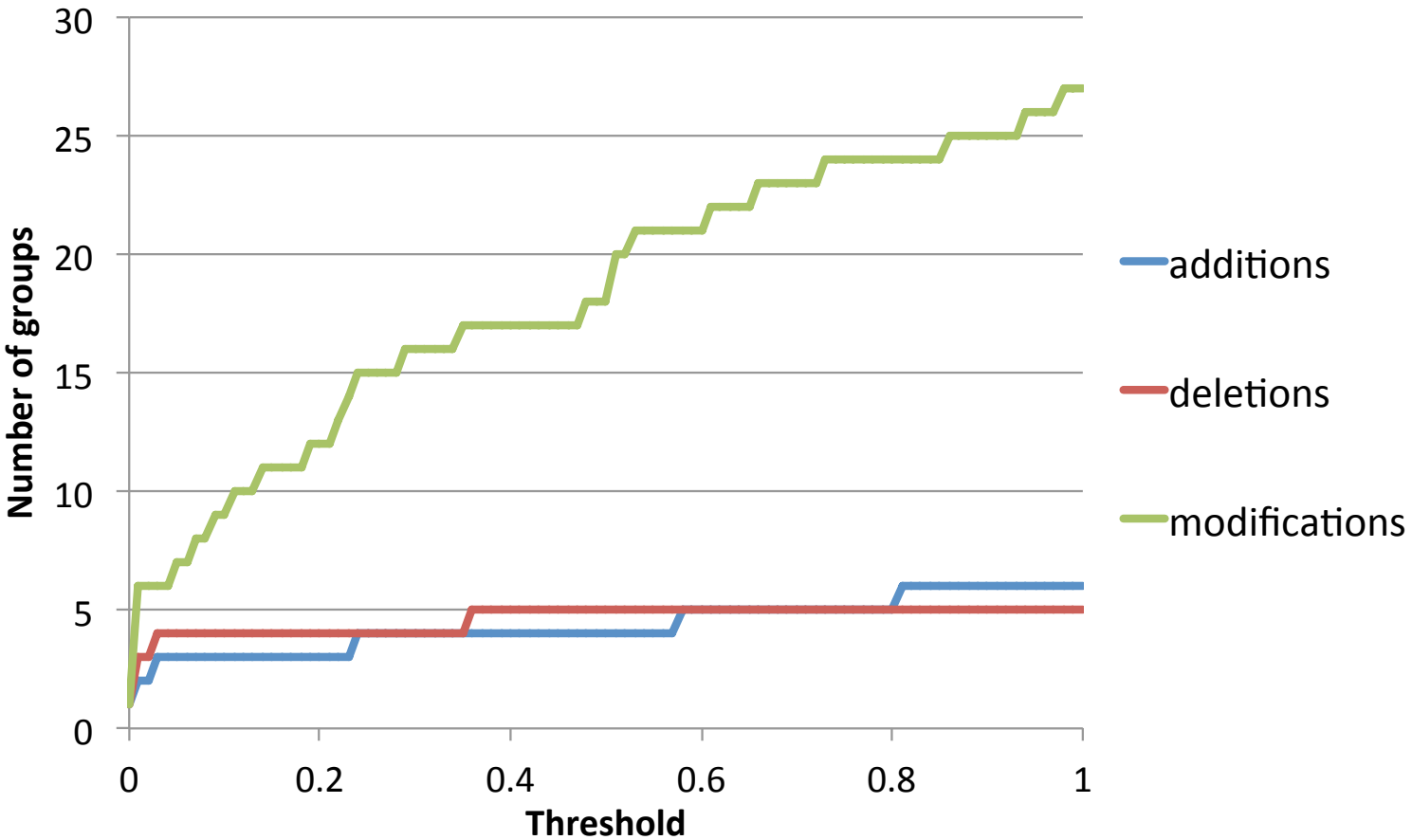
$$subst_l = \{\square_1 \mapsto B\ ,\ \square_2 \mapsto C\}$$

$$subst_r = \{\square_1 \mapsto \begin{array}{c} \text{B} \\ \text{D} \end{array}\ ,\ \square_2 \mapsto F\}$$

galois

Similarity groups versus threshold

What happens to similarity groups when we vary the threshold?



Number of additions, deletions, and modifications by threshold for the Clojure source.

galois

Patterns as a function of threshold

Generic Loop pattern, $\tau = 0.15$:

```
for (□ = □; □ < □; □) {
    □
}
```

Loop counter is initialized to zero, $\tau = 0.25$:

```
for (□ = 0; □ < □; □) {
    □
}
```

Loop termination criteria becomes more specific, $\tau = 0.35$:

```
for (□ = 0; □ < □.□; □) {
    □
}
```

galois

Future work

- We only consider structural patterns
  - Example: We don't detect design patterns
- Not semantically aware
  - Example: changing the name of a loop variable leads to □
- Generate rewrite rules based on before and after patterns
- Use patterns for searching as a structural `grep`-like mechanism
- Correlate patterns with bug fixes

galois

Thank you!

Questions?

This work was supported in part by the US Department of Energy Office of Science, Advanced Scientic Computing Research contract no. DE-SC0004968. Additional support was provided by Galois, Inc.

galois