# Ctcompare: Comparing Multiple Code Trees for Similarity

**Warren Toomey**
*School of IT, Bond University*

Using lexical analysis with techniques borrowed from DNA sequencing, multiple code trees can be quickly compared to find any code similarities

# Why Write a Code Comparison Tool?

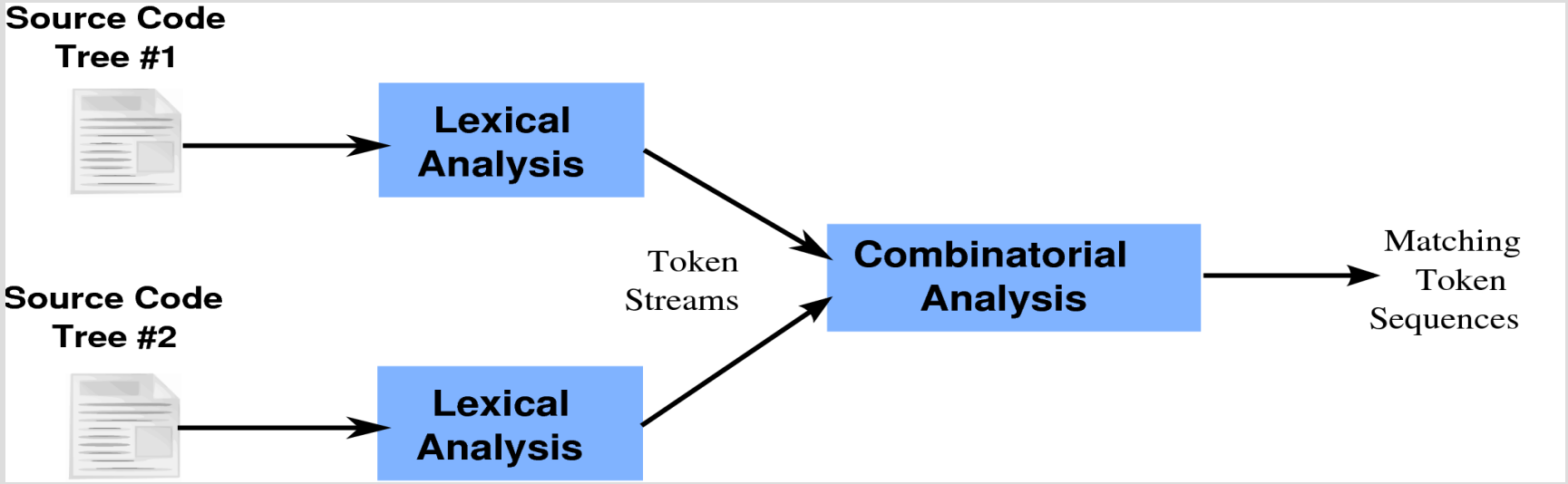
- To detect student plagiarism
- To determine if your codebase is `infected' by proprietary code from elsewhere, or by open-source code covered by a license like the GPL
- To trace code genealogy between trees separated by time (e.g. versions), useful for the new field of computing history

# Code Comparison Issues

- Can rearrangement of code be detected?
  - per line? per sub-line?
- Can “munging of code” be detected?
  - variable/function/struct renaming?
- What if one or both codebases are proprietary? How can third parties verify any comparison?
- Can a timely comparison be done?
- What is the rate of false positives?
  - of missed matches?

# Lexical Comparison

- First version: break the source code from each tree into lexical tokens, then compare runs of tokens between trees.
- This removes all the code's semantics, but does deal with code rearrangement (but not code “munging”).



# Performance of Lexical Approach

- Performance was O(M*N), when M and N the # of tokens in each code tree
- Basically: slow, and exponential
- I also wanted to compare multiple code trees, and also find duplicated code within a tree

# When is Copying Significant?

- Common code fragments not significant:

  for (i=0; i< val; i++) 13 tokens

  if (x > max) max=x; 10 tokens

  err= stat(file, &sb); if (err) 14 tokens

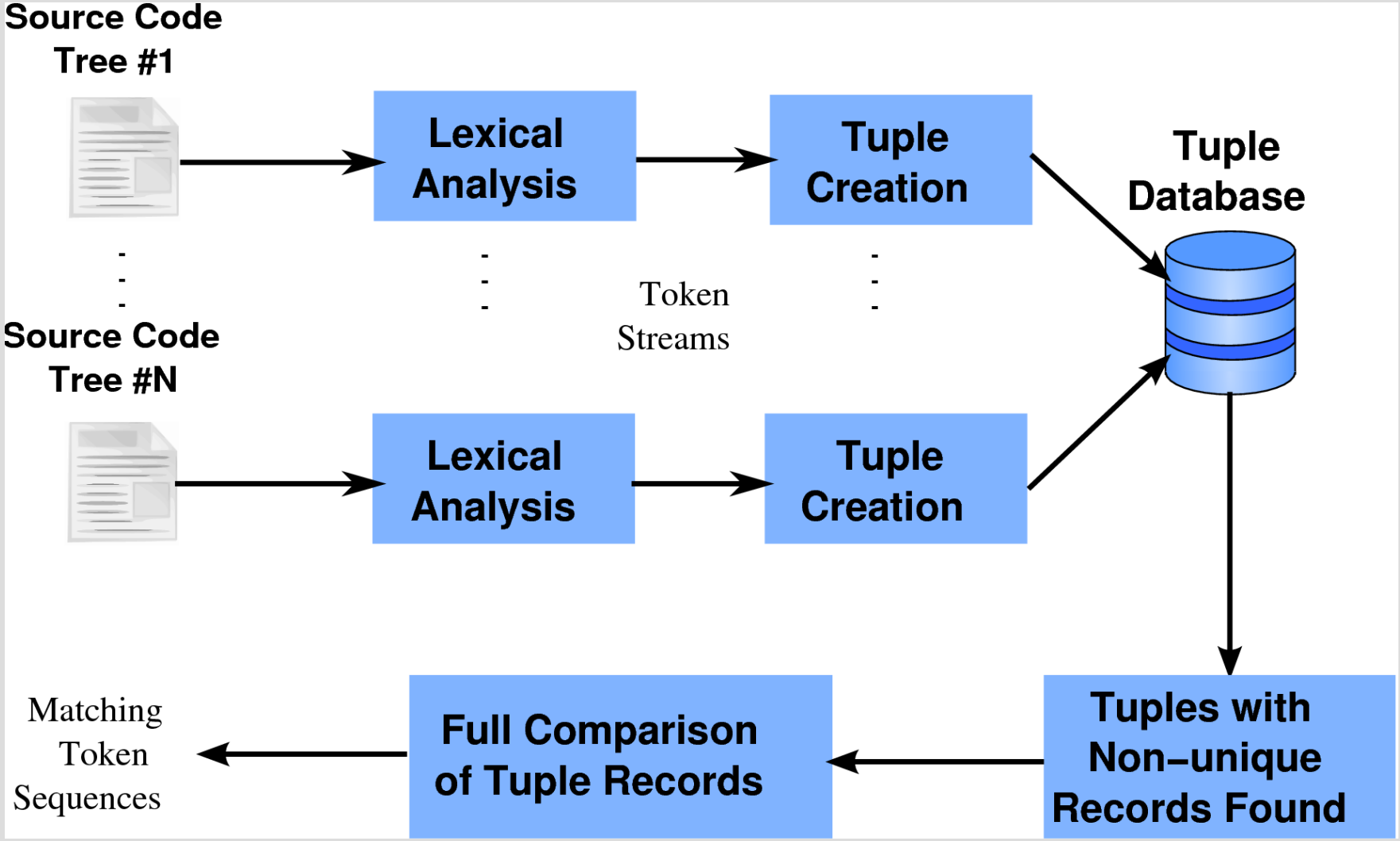
- **Axiom:** runs of < 16 tokens insignificant

- **Idea:** use a run of 16 tokens as a lookup key to find other trees with the same run of tokens

# Approach in Second Version

- Take all 16-tokens runs in a code tree
- Use each as a key, and insert the runs into a database along with position of the run in the source tree
- Keys (i.e. runs of 16 tokens) with multiple values indicate possible code copying of at least 16 tokens
- For these keys, we can evaluate all code trees from this point on to find any real code similarity

# Approach in Second Version



# Algorithm

```
for each key in the database with multiple record nodes {
  obtain the set of record nodes from the database;

  for each combination of node pairs Na and Nb {
    if (performing a cross-tree comparison)
      skip this combination if Na and Nb in the same tree;
    if (performing a code clone comparison)
      skip this combination if Na and Nb in the same file;

    perform a full token comparison beginning at Na and Nb to
    determine the full extent of the code similarity, i.e. determine
    the actual number of tokens in common;

    add the matching token sequence to a list of "potential runs";
  }
}
walk the list of potential runs to merge overlapping runs, and
remove runs which are subsets of larger runs;

output the details of the actual matching token runs found.
```

# Hashing Identifiers and Literals in Each Code Tree

- Simply comparing tokens yields false positives, e.g

```
if (xx > yy) { aa = 2 * bb + cc; }
if (ab > bc) {ab = 5 * ab + bc; }
```

- I hash each identifier and literal down to a 16-bit value

- This obfuscates the original values while minimising the amount of false positives

# Isomorphic Comparison

- Hashing identifiers helps to ensure code similarities are found
- Does not help if variables have been renamed
- This can be solved with isomorphic comparison: keep a 2-way variable isomorphism table
- Code is isomorphic when variable names are isomorphic across a long run of code

# Isomorphic Example

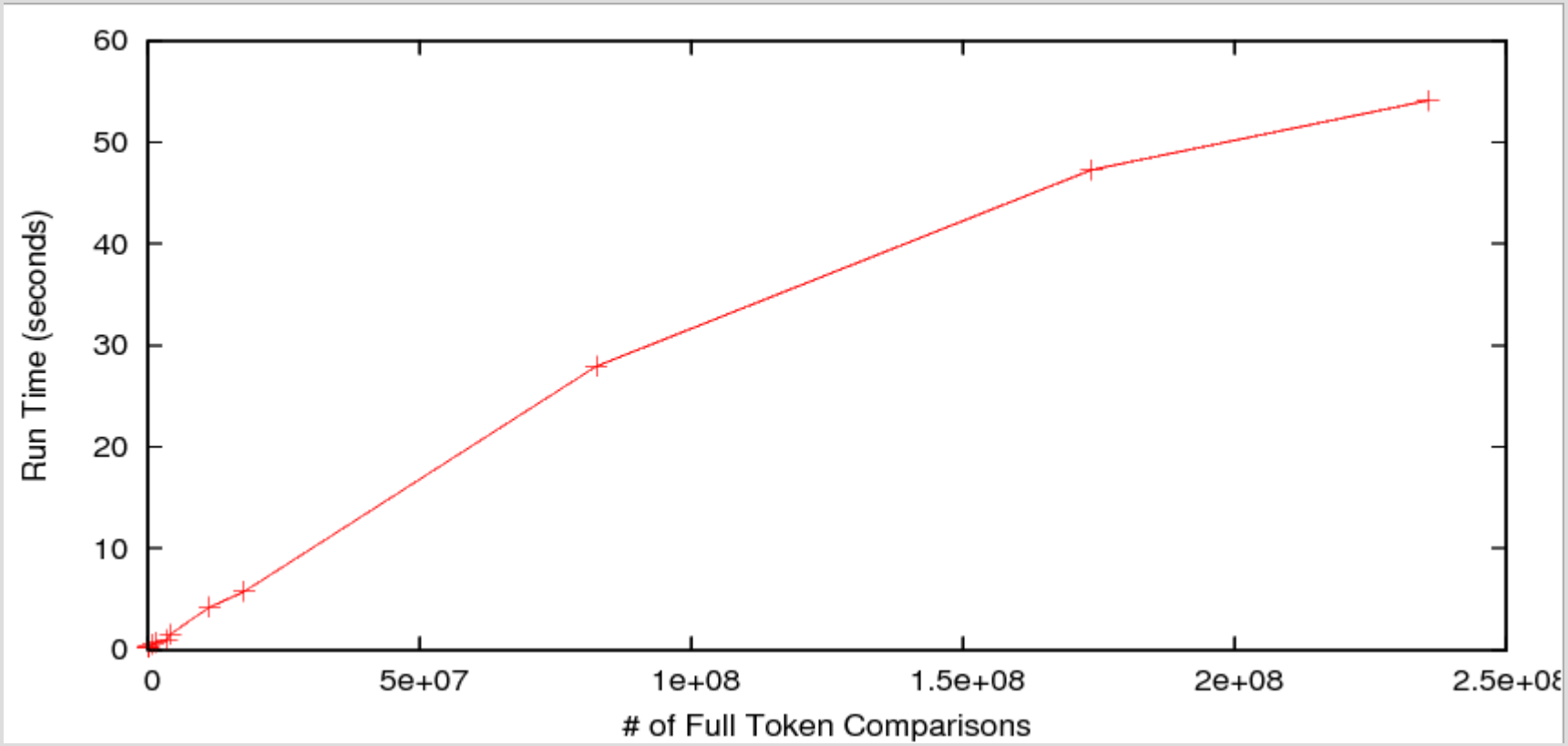
```
int maxofthree(int x, int y, int z)
 {
   if ((x>y) && (x>z)) return(x);
   if (y>z) return(y);
   return(z);
 }

int bigtriple(int b, int a, int c)
 {
   if ((b>a) && (b>c)) return(b);
   if (a>c) return(a);
   return(c);
 }
```

# Bottlenecks

- In the second version, main bottleneck is the merging of overlapping code fragments which have been found to match between two trees
- Similar to the problem with DNA sequencing when genes are split up into multiple fragments before sequencing
- Use of AVL trees here has helped to reduce the cost of merging immensely
- Other more mundane optimisations such as mmap() have also helped

# Current Performance




- 54 seconds to compare 14 trees of code totalling 1 million lines of C code

# Results

- Many lines of code written at UNSW in late 1970s still present in System V, and in Open Solaris
- Thousands of lines of replicated code inside Linux kernel
- AT&T vs BSDi lawsuit in early 1990s:
  - expert witness found 56 common lines in 230K lines between BSD and Unix
  - my tool find roughly same lines, plus 100+ more isomorphic similarities

# Question?