CodeHunter: Context-Driven Component-Based Synthesis from Big Data

{ Yu Feng, Yuepeng Wang, Arati Kaushik, Isil Dillig }@UT Austin

Objective and Goals

CodeHunter is a system for component-based program synthesis in Java. Unlike previous component-based program synthesis approaches, our approach is:

1. ***Data-driven:*** Rather than using a fixed set of components, CodeHunter *dynamically* discovers relevant components in large code corpora.
2. ***Context-driven:*** CodeHunter uses *context* of the program to be synthesized to identify pertinent code snippets. We interpret *program context* in a broad way: it includes method signature, comments, test cases, program history …

Challenges

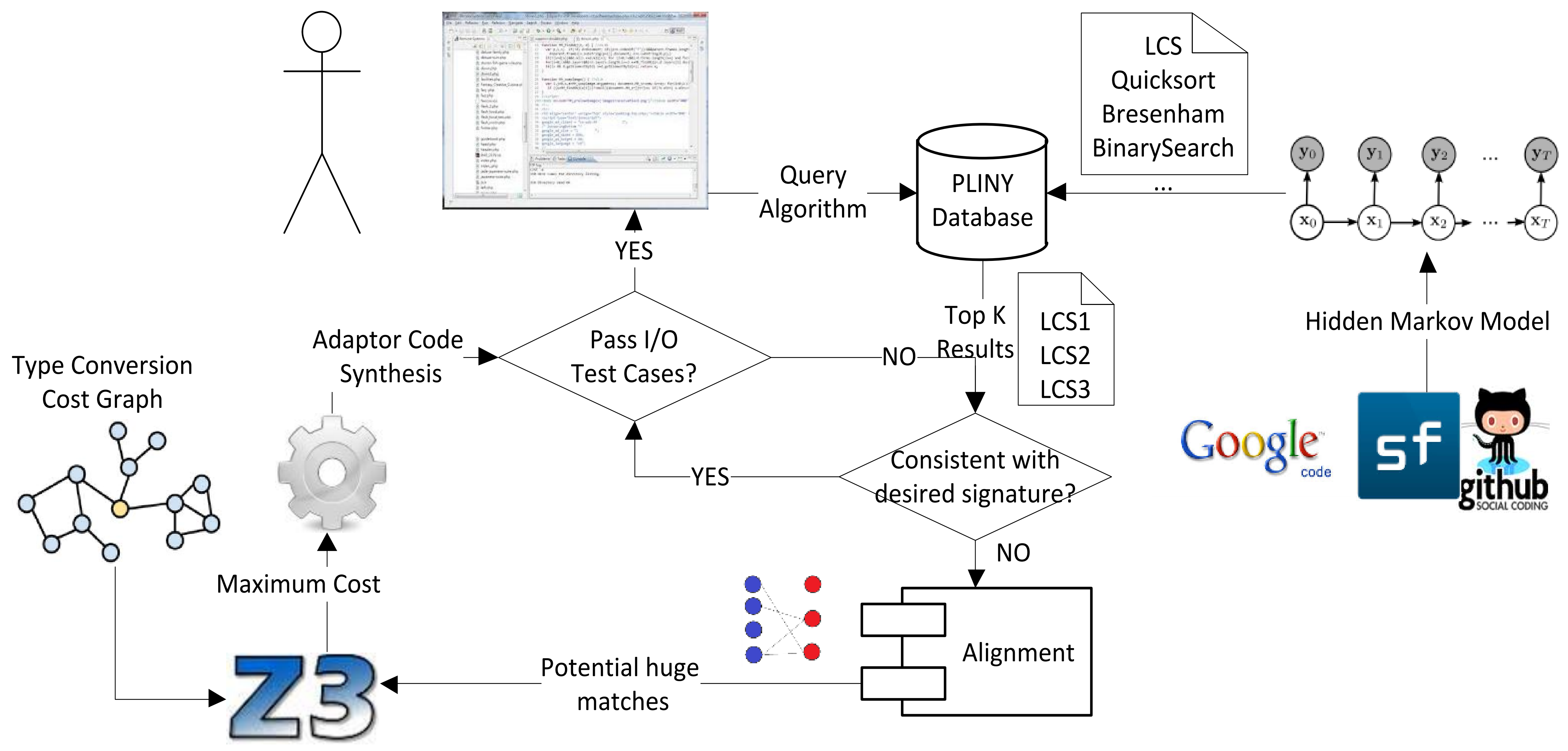
- ***Context abstraction:*** Identify a good abstraction of program context to drive database queries for discovering relevant components
- ***Interface alignment:*** Once we identify relevant component, figure out correspondence between this component's interface and existing context elements
- ***Adaptor code synthesis:*** We need to synthesize adaptor code to integrate component into program context

Our Solution

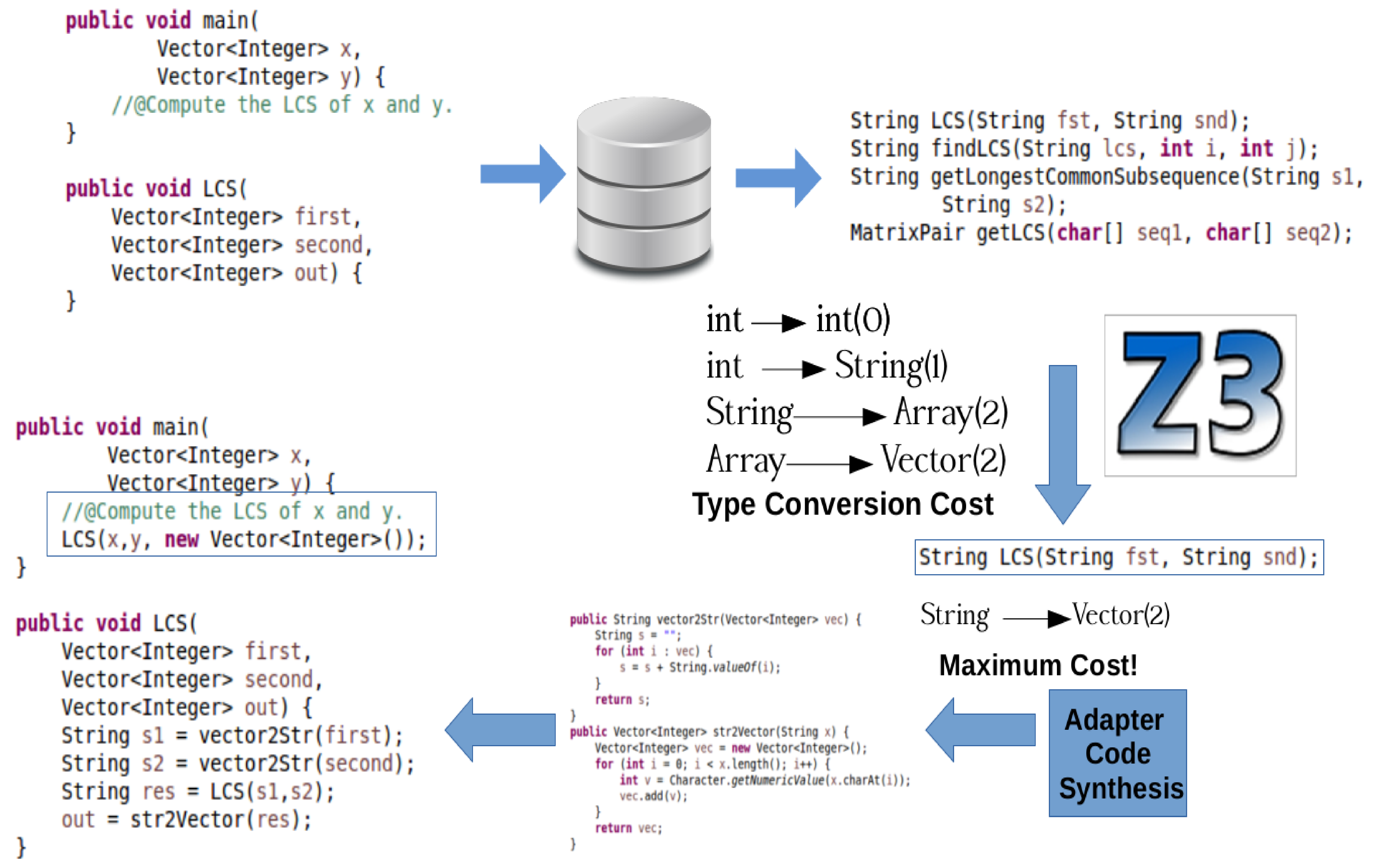
We combine the advantages of statistical and SMT-based techniques:

1. Use Hidden Markov Models to learn a good abstraction of program context to drive database queries
2. Formulate interface assignment as a *satisfiability modulo the theory of costs* problem (i.e., find maximum cost matching that satisfies a set of integrity constraints)
3. Use type-directed synthesis to generate adaptor code

Overview of our approach



Our approach by example



```
public void main(
        Vector<Integer> x,
        Vector<Integer> y) {
    //@Compute the LCS of x and y.
}

public void LCS(
    Vector<Integer> first,
    Vector<Integer> second,
    Vector<Integer> out) {
}
```

```
String LCS(String fst, String snd);
String findLCS(String lcs, int i, int j);
String getLongestCommonSubsequence(String s1,
        String s2);
MatrixPair getLCS(char[] seq1, char[] seq2);
```

```
String LCS(String fst, String snd);
```

```
public String vector2Str(Vector<Integer> vec) {
    String s = "";
    for (int i : vec) {
        s = s + String.valueOf(i);
    }
    return s;
}
public Vector<Integer> str2Vector(String x) {
    Vector<Integer> vec = new Vector<Integer>();
    for (int i = 0; i < x.length(); i++) {
        int v = Character.getNumericValue(x.charAt(i));
        vec.add(v);
    }
    return vec;
}
```

```
public void main(
        Vector<Integer> x,
        Vector<Integer> y) {
    //@Compute the LCS of x and y.
    LCS(x,y, new Vector<Integer>());
}

public void LCS(
    Vector<Integer> first,
    Vector<Integer> second,
    Vector<Integer> out) {
    String s1 = vector2Str(first);
    String s2 = vector2Str(second);
    String res = LCS(s1,s2);
    out = str2Vector(res);
}
```

Future Work

We plan to continue our current work and integrate it into Eclipse IDE to reduce the workload of daily program task; We also plan to generate program specification for large corpus based on comments in natural language. Finally, currently our techniques only support Java program and we plan to extend our framework to support other languages like C and C++.

Acknowledgments

This material is based on research sponsored by the Air Force Research Laboratory, under agreement number FA8750-14-2-0270.

References

[1] Perelman, Daniel, et al. Type-directed completion of partial expressions In *PLDI,2012*

[2] Feser, John, Swarat Chaudhuri, and Isil Dillig. Synthesizing Data Structure Transformations from Input-Output Examples. In *PLDI 2015*