Semantics-Based Code Search

Steven P. Reiss
Brown University

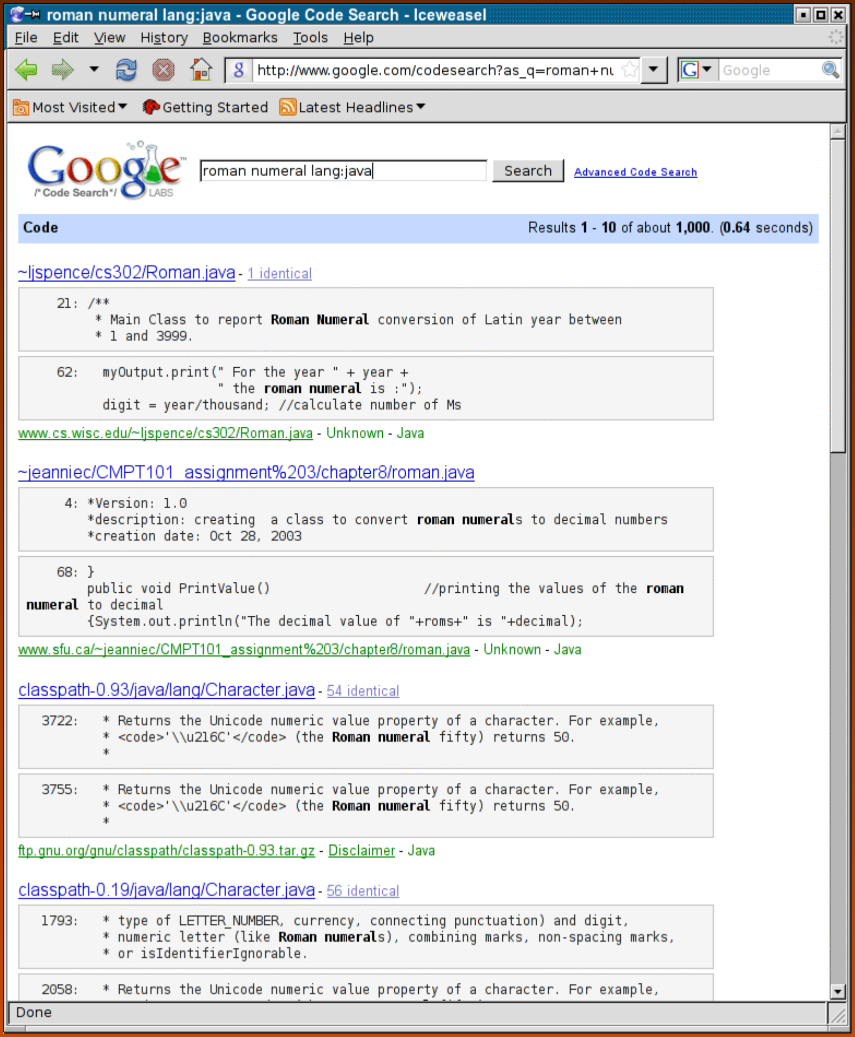
Problem

- Why write code?
  - When someone else already has?
  - Lots of open source code available
  - Its all been written already
- Instead just find the code you want
  - That’s what search engines are good for
  - And you’ll save lots of time
- Unfortunately, this doesn’t work

Example

- Convert an integer to a roman numeral
- This is ambiguous
  - What language? (Java)
  - Upper or lower case
  - How to handle large numbers
  - How to handle negative numbers
  - How to handle special cases (e.g. 4)

Code Search


roman numeral lang:java - Google Code Search - Iceweasel
File Edit View History Bookmarks Tools Help
http://www.google.com/codesearch?as_q=roman+nu
Google
Most Visited
Getting Started
Latest Headlines
Google /* Code Search */ LABS
roman numeral lang:java
Search
Advanced Code Search
Done

Code Results **1** - **10** of about **1,000**. (**0.64** seconds)

~ljspence/cs302/Roman.java - 1 identical

```
21: /**
     * Main Class to report Roman Numeral conversion of Latin year between
     * 1 and 3999.
```

```
62:   myOutput.print(" For the year " + year +
                     " the roman numeral is :");
      digit = year/thousand; //calculate number of Ms
```

www.cs.wisc.edu/~ljspence/cs302/Roman.java - Unknown - Java

~jeanniec/CMPT101_assignment%203/chapter8/roman.java

```
 4: *Version: 1.0
    *description: creating  a class to convert roman numerals to decimal numbers
    *creation date: Oct 28, 2003
```

```
68: }
    public void PrintValue()                    //printing the values of the roman
numeral to decimal
    {System.out.println("The decimal value of "+roms+" is "+decimal);
```

www.sfu.ca/~jeanniec/CMPT101_assignment%203/chapter8/roman.java - Unknown - Java

classpath-0.93/java/lang/Character.java - 54 identical

```
3722:   * Returns the Unicode numeric value property of a character. For example,
        * <code>'\\u216C'</code> (the Roman numeral fifty) returns 50.
        *
```

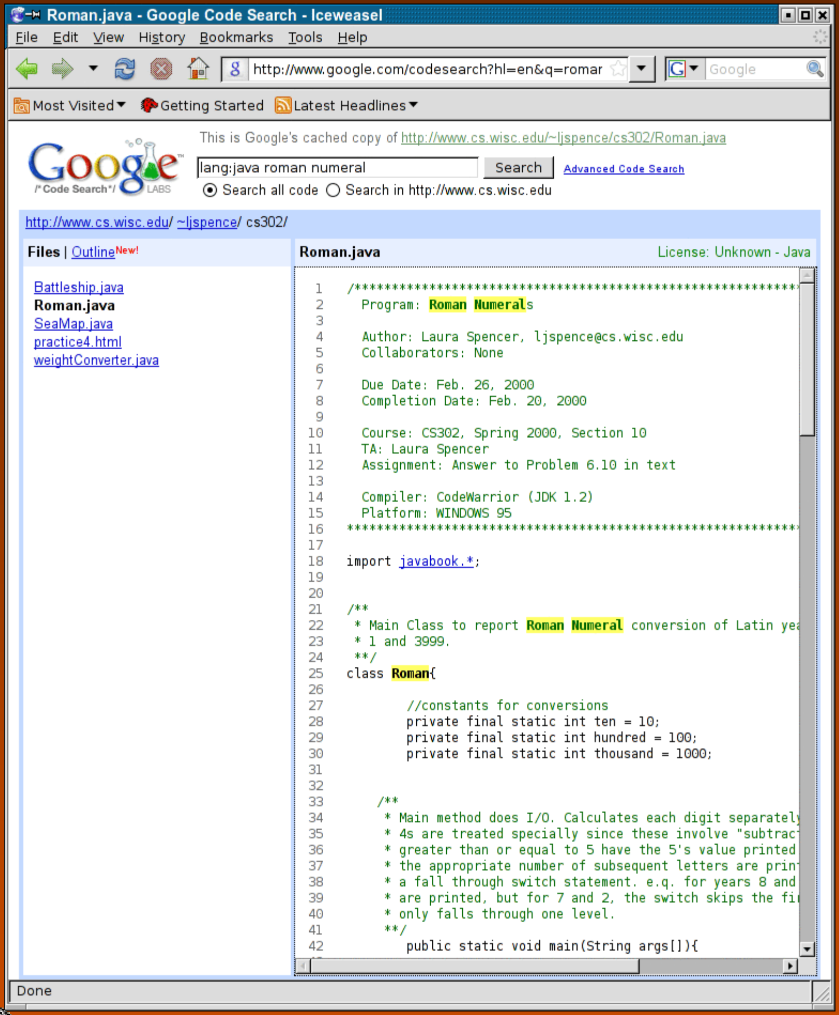
```
3755:   * Returns the Unicode numeric value property of a character. For example,
        * <code>'\\u216C'</code> (the Roman numeral fifty) returns 50.
        *
```

ftp.gnu.org/gnu/classpath/classpath-0.93.tar.gz - Disclaimer - Java

classpath-0.19/java/lang/Character.java - 56 identical

```
1793:   * type of LETTER_NUMBER, currency, connecting punctuation) and digit,
        * numeric letter (like Roman numerals), combining marks, non-spacing marks,
        * or isIdentifierIgnorable.
```

```
2058:   * Returns the Unicode numeric value property of a character. For example,
```


Roman.java - Google Code Search - Iceweasel
File Edit View History Bookmarks Tools Help
http://www.google.com/codesearch?hl=en&q=romar
Google
Most Visited
Getting Started
Latest Headlines
Google /* Code Search */ LABS
lang:java roman numeral
Search
Advanced Code Search
Done

This is Google's cached copy of http://www.cs.wisc.edu/~ljspence/cs302/Roman.java

● Search all code ○ Search in http://www.cs.wisc.edu

http://www.cs.wisc.edu/ ~ljspence/ cs302/

Files | Outline New!

- Battleship.java
- **Roman.java**
- SeaMap.java
- practice4.html
- weightConverter.java

Roman.java License: Unknown - Java

```
/**************************************************************
  Program: Roman Numerals

  Author: Laura Spencer, ljspence@cs.wisc.edu
  Collaborators: None

  Due Date: Feb. 26, 2000
  Completion Date: Feb. 20, 2000

  Course: CS302, Spring 2000, Section 10
  TA: Laura Spencer
  Assignment: Answer to Problem 6.10 in text

  Compiler: CodeWarrior (JDK 1.2)
  Platform: WINDOWS 95
**************************************************************

import javabook.*;


/**
 * Main Class to report Roman Numeral conversion of Latin yea
 * 1 and 3999.
 **/
class Roman{

        //constants for conversions
        private final static int ten = 10;
        private final static int hundred = 100;
        private final static int thousand = 1000;


    /**
     * Main method does I/O. Calculates each digit separately
     * 4s are treated specially since these involve "subtrac
     * greater than or equal to 5 have the 5's value printed
     * the appropriate number of subsequent letters are prin
     * a fall through switch statement. e.q. for years 8 and
     * are printed, but for 7 and 2, the switch skips the fi
     * only falls through one level.
     **/
        public static void main(String args[]){
```

What’s Wrong Here

- Too much to look at
- Need to read each example
  - Is it relevant to the search?
  - Does it do what you want?
  - Does it work in all cases of interest?
  - Will it be fast enough, secure enough, … ?
- Then you need to transform it
  - To fit your application and style
- More work than writing the code

Our Goal

- Make this practical
- Specify exactly what you want to find
  - Give the syntax and semantics
  - For either classes or methods
- System finds approximate code
  - That might accomplish the task
  - What search engines do today
- System transforms that code
  - To meet the actual specifications
  - Even to meet the programmer’s style
- System creates a working result

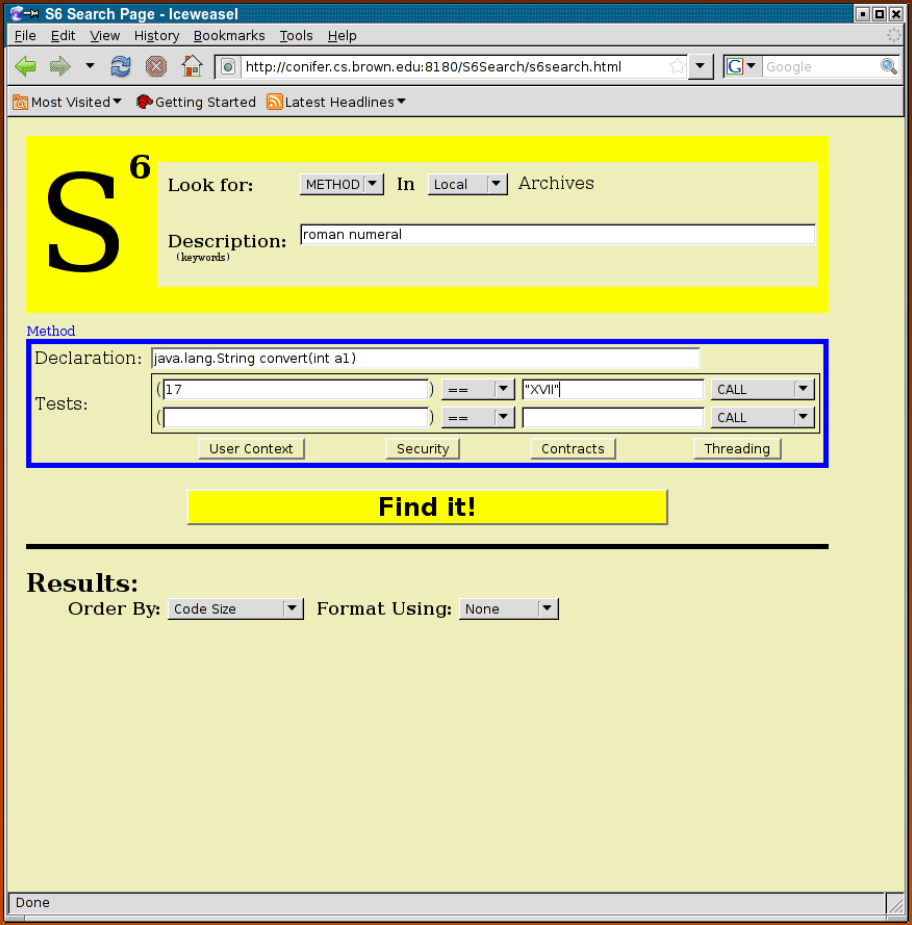
Specifying What to Find

- Description of what is wanted
  - Keywords
- Signature
  - For a method or class
- Functional semantics
  - Test cases
  - Contracts
- Non-functional semantics
  - Security, threading, context, style, …

To Be Practical

- Must be easy to use
  - Simple to specify semantics
  - Easier than writing the code
- Must be able to trust the result
  - Small set of results
- Code must be returned ready to use
  - In the form the programmer needs it
  - Converting is as hard as writing

Semantics-Based Search



S6 Search Page - Iceweasel

http://conifer.cs.brown.edu:8180/S6Search/s6search.html

Look for: METHOD In Local Archives

Description: roman numeral (keywords)

Method

Declaration: java.lang.String convert(int a1)

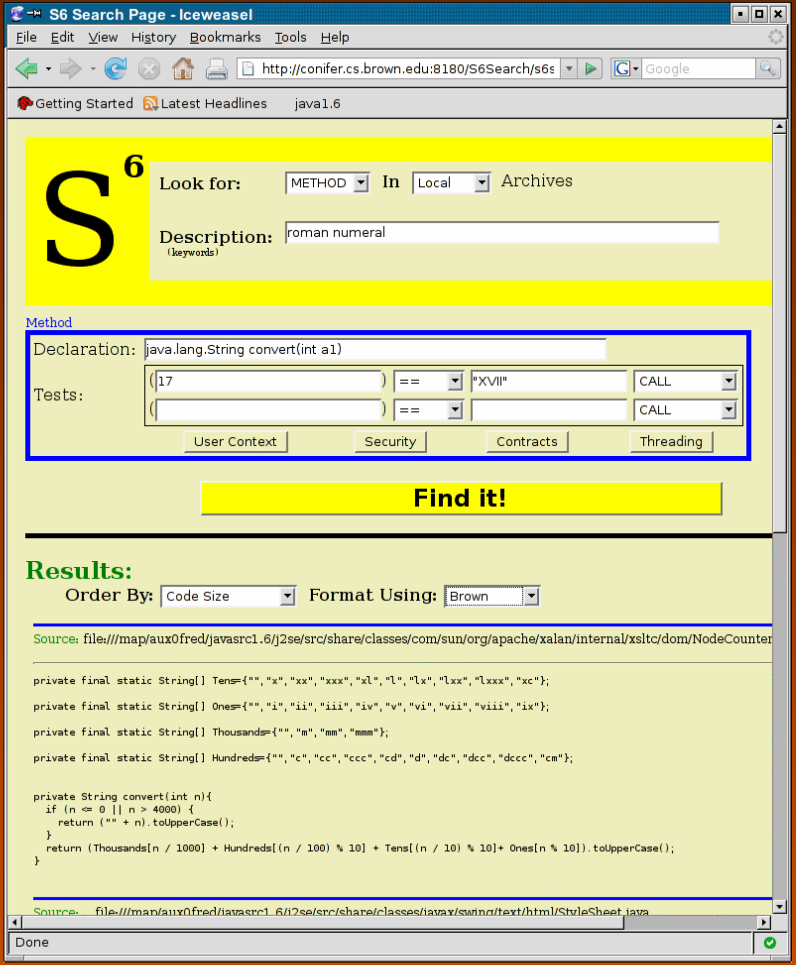
Tests: (17) == "XVII" CALL

Find it!

Results:

Order By: Code Size Format Using: None

Done



S6 Search Page - Iceweasel

http://conifer.cs.brown.edu:8180/S6Search/s6s

Look for: METHOD In Local Archives

Description: roman numeral (keywords)

Method

Declaration: java.lang.String convert(int a1)

Tests: (17) == "XVII" CALL

Find it!

Results:

Order By: Code Size Format Using: Brown

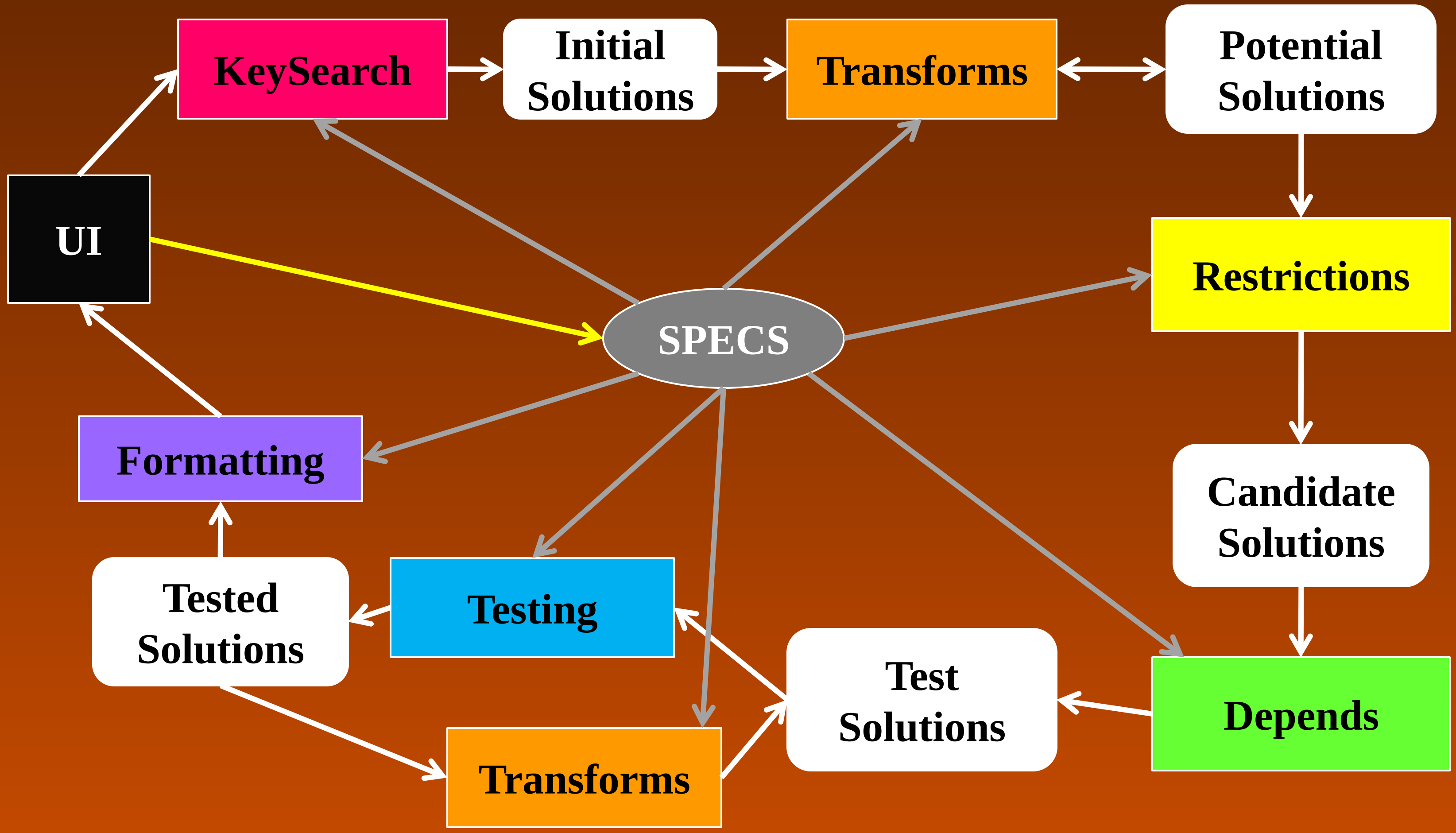
Source: file:///map/aux0fred/javasrc1.6/j2se/src/share/classes/com/sun/org/apache/xalan/internal/xsltc/dom/NodeCounter

```
private final static String[] Tens={"","x","xx","xxx","xl","l","lx","lxx","lxxx","xc"};

private final static String[] Ones={"","i","ii","iii","iv","v","vi","vii","viii","ix"};

private final static String[] Thousands={"","m","mm","mmm"};

private final static String[] Hundreds={"","c","cc","ccc","cd","d","dc","dcc","dccc","cm"};


private String convert(int n){
  if (n <= 0 || n > 4000) {
    return ("" + n).toUpperCase();
  }
  return (Thousands[n / 1000] + Hundreds[(n / 100) % 10] + Tens[(n / 10) % 10]+ Ones[n % 10]).toUpperCase();
}
```

Source: file:///map/aux0fred/javasrc1.6/j2se/src/share/classes/javax/swing/text/html/StyleSheet.java

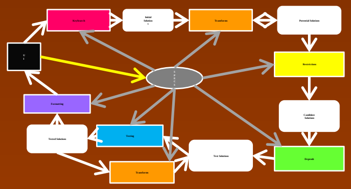
Done

What's Behind the Scenes



Getting Initial Solutions

- Keyword Search
  - Use existing global code search engines
    - Google, Koders, Krugle
  - Local search
    - Beagle, Labrador
  - This yields a set of source files
- Initial solutions
  - All methods/classes in the found files
  - Represented as annotated Eclipse ASTs
    - With compilation information
    - Note that we can't assure compilation

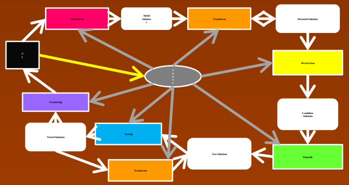


Example

```
public static String toRoman(long n){
   int i;
  String s;
  s="";
  while (n > 0) {
    for (i=0; i < syms.length; i++) {
      if (syms[i].value <= n) {
        int shift=i + (i % 2);
        if (i > 0 && shift < syms.length && (syms[i - 1].value - syms[shift].value) <= n) {
          s=s + syms[shift].symbol + syms[i - 1].symbol;
          n=n - syms[i - 1].value + syms[shift].value;
          i=-1;
        }
 else {
          s+=syms[i].symbol;
          n-=syms[i].value;
          i=-1;
        }
      }
    }
  }
  return s;
}
```

Transformations

- Need to adapt code to specifications
  - Conform to signature provided
  - Identify hidden functionality
  - Eliminate unneeded functionality
    - Method does A and B, we only want A
  - Ensure compilability
- Build new solutions
  - From original and transformed solutions
  - Iterated until no new solutions found
  - Solutions represented via deltas



Transformations

- Heuristics to avoid too many solutions
  - All transformations are conditioned
    - To avoid exponential blowup
  - Duplicate solutions ignored
  - Transformation applied once per solution
- Categories of transformations
  - Signature conformance
  - Building new code
  - Handling the user's context
  - Compilation conformance
  - Test-result based

Signature Transforms

- NAME
  - Change the name if signature matches
- RETURN
  - Change return type if parameters match
- PARAMETER TYPES
  - Change parameter types to match
- INT PARAMETERS
  - Handle integer parameter type conversion
- PARAMETER ORDER
  - Reorder parameters to match
- EXCEPTION
  - Remove unmatched throws

Signature Transforms

- STATIC
  - Ensure method is static if necessary
- REMOVE STATIC
  - Convert static method to class method
- REMOVE PACKAGE
  - Remove package statement from a class
- STATIC CLASS
  - Ensure classes are static, not nested

New Transforms

- CHUNK
  - Find subset of the code that computes a value of the target type given input types
    - Useful for finding embedded functionality
  - Methodology: backward slice
    - Find variables; compute uses and definitions
    - Find each statement computing target type
    - For each prior statement that is needed
      - Include it and recompute active variables
    - Whenever active variables match parameters
      - Generate a new function

New Transforms

- EXTRA PARAMETERS
  - Replace extra parameters with assignments
  - Try different assignments
    - For booleans: both true and false
    - For integers: both 0 and 1
    - For all:
      - Any value that appears in a conditional with variable
      - All switch cases based on variable
  - Each assignment is a new solution
- GENERALIZE
  - Replace user-defined with specified types
    - Union-Find for Temporary
    - Where the type isn’t used as such

Context Transforms

- CONTEXT TYPES
  - Map types in found code to those in user's context
    - Handle fields based on type compatability
    - Handle methods based on type compatability
  - Try all possible combinations

Compilation Transforms

- IMPLEMENTS
  - Remove extends clause for user types
  - Remove implements clause unless spec'd
- REMOVE UNDEFINED
  - Remove any statements that access undefined fields, methods, or types
  - Don't remove final statements, returns
- THROW
  - Remove any throw clauses that aren't in specification
  - Replace internal throws appropriately

Example

- Apply INT PARAMETERS

```
public static String toRoman(int s6__n){
 long n=s6__n;
 int i;
 String s;
 s="";
 while (n > 0) {
   for (i=0; i < syms.length; i++) {
     if (syms[i].value <= n) {
       int shift=i + (i % 2);
       if (i > 0 && shift < syms.length && (syms[i - 1].value - syms[shift].value) <= n) {
         s=s + syms[shift].symbol + syms[i - 1].symbol;
         n=n - syms[i - 1].value + syms[shift].value;
         i=-1;
       }
 else {
         s+=syms[i].symbol;
         n-=syms[i].value;
         i=-1;
       }
     }
   }
 }
 return s;
}
```

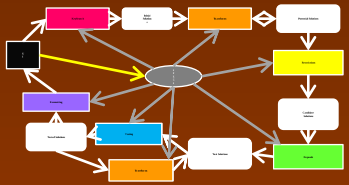
Example

- **Apply CHANGE NAME**

```
public static String convert(int s6__n){
 long n=s6__n;
 int i;
 String s;
 s="";
 while (n > 0) {
   for (i=0; i < syms.length; i++) {
     if (syms[i].value <= n) {
       int shift=i + (i % 2);
       if (i > 0 && shift < syms.length && (syms[i - 1].value - syms[shift].value) <= n) {
         s=s + syms[shift].symbol + syms[i - 1].symbol;
         n=n - syms[i - 1].value + syms[shift].value;
         i=-1;
       }
 else {
         s+=syms[i].symbol;
         n-=syms[i].value;
         i=-1;
       }
     }
   }
 }
 return s;
}
```

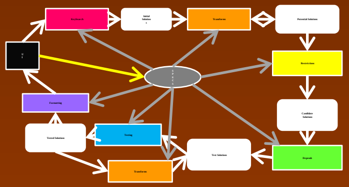
Testing

- Create a runable test program
  - Containing the solution code
  - Containing the test cases
  - Compile and test
  - Check contracts
  - Get test results
- Not as easy as it seems
  - Code often won't compile directly
    - Requires helper methods, fields, imports
    - Class might have extra methods
    - Need to change package/class names



Dependency Analysis

- Do static analysis of solution & its file
  - Determine what is required
  - Inner classes, fields, methods
  - Imports
  - Done transitively
  - Using user context if provided
- If there are still unresolved symbols
  - Invalidate the solution
- Construct minimal solution
  - Method search
  - Class search



Example

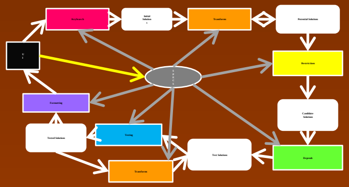
- Add dependencies

```
public static Roman.SymTab syms[]={new Roman.SymTab('M',1000),
new Roman.SymTab('D',500), new Roman.SymTab('C',100),
new Roman.SymTab('L',50), new Roman.SymTab('X',10),
new Roman.SymTab('V',5), new Roman.SymTab('I',1)};

public static class SymTab {
  char symbol;
  long value;
  public SymTab(  char s,  long v){
    this.symbol=s;
    this.value=v;
  }
}
```

- Fix class name
  - Roman ➔ <local class>

Testing

- **Run JUNIT using ANT**
  - **Compile and run**
  - **Use jmlc for contract checking**
  - **Check for compilation errors**
  - **Check for test success**
  - **Check for test failure**
    - **And why the test failed**
- **Apply transforms to failing tests**
  - **Then retest**



Testing-Based Transforms

- FIX RETURN
  - Change return value based on test results
  - Boolean:
    - If always wrong, invert the result
  - Numeric:
    - Detect fixed delta, change result (off by one)
  - String:
    - Handle case differences

Example

- Apply FIX RETURN

```
public static String convert(int s6__n){
  long n=s6__n;
  int i;
  String s;
  s="";
  while (n > 0) {
    for (i=0; i < syms.length; i++) {
      if (syms[i].value <= n) {
        int shift=i + (i % 2);
        if (i > 0 && shift < syms.length && (syms[i - 1].value - syms[shift].value) <= n) {
          s=s + syms[shift].symbol + syms[i - 1].symbol;
          n=n - syms[i - 1].value + syms[shift].value;
          i=-1;
        }
 else {
          s+=syms[i].symbol;
          n-=syms[i].value;
          i=-1;
        }
      }
    }
  }
  return (s).toLowerCase();
}
```

Displaying the Results

- Show the code as part of the front end
- Let the user sort the results
  - By code size, complexity, run time
- Let the user reformat the code
  - In various styles
  - Eventually in their own style
- Let the user see the license information
  - Extracted from original file
- Interactive
  - User can add new test cases, change keywords

Experience: Method Examples

Name	Keywords	Signature	# Tests
Simple Tokenizer	tokenize	List<String> tokenize(String)	1
Quote Tokenizer	command tokenize split argument quote list	List tokenize(String)	2
Robots.txt	robots.txt	boolean check(URL)	3
Log2	log base	int log2(int)	3
To Roman	roman numeral	String toRoman(int)	1
From Roman	roman numeral	Int convert(String)	1
Primes	prime number	boolean checkPrime(int)	3
Perfect numbers	“perfect number”	boolean isPerfect(int)	3
Easter	Easter date holiday year	Date computeEaster(int)	1

Experience: Class Examples

Name	Keywords	Signature	# Calls

Multimap	multiset multimap	class Multimap {
		  Multimap();
		  void add(Object);
		  int count(Object);
		}
			11

Union-find	union find	class UnionFind {
		  UnionFind();
		  void add(Object);
		  Object find(Object);
		  void union(Object,Object);
		}
			7

Text delta	text delta	Class TextDelta {
		  TextDelta(String n,String o);
		  String Apply(String o);
		}
			3

Experience: Results

Example	Engines	1-Time	8-Time	#Src	#Init	#Total	#Test	#Rslt
Simple Tokenizer	Labrador	65.026	29.167	138	3862	4173	39	17
Quote Tokenizer	Labrador	17.676	11.499	3	162	205	8	6
Robots.txt	Kruble, Labrador	80.762	25.067	124	145	742	20	1
Log2	Google	1346.711	366.893	100	2430	5588	1106	51
To Roman	Labrador	19.444	14.155	7	234	255	5	2
From Roman	Google	703.362	242.169	106	2530	3723	401	28
Primes	Google, Labrador	187.384	55.581	228	4449	5250	129	19
Perfect Numbers	Google	25.512	9.824	31	93	111	13	7
Easter	Koders	8.968	8.454	6	72	75	1	1
Multimap	Labrador	82.785	48.019	149	183	1110	25	1
UnionFind	Labrador	454.443	190.132	149	416	5400	11	1
Text Delta	Google, Labrador	28.243	22.254	249	365	996	1	1

Current Status

- Accessible on the web
  - http://conifer.cs.brown.edu/s6
- The system works (most of the time)
- System is dependent on
  - Quality of initial search
  - The set of transformations
- Time varies considerably
  - Based on # solutions, # tests
  - These vary with signature
- Everything hasn't been done before
- Test cases aren't sufficient
- More transformations are required

Current and Future Work

- Additional transforms
  - Class ↔ function
  - Advanced type conversions
- Context information
  - Searching within the user's context
  - Using user code for compiling and testing
- Using related files
- Threading constraints
- Eclipse interface
- Performance
- Interactive specifications

Acknowledgements

- NSF support: CCR0613162
- Student support:
  - Christina Salvatore (formatting)
  - Paul Meier (formatting)
  - J. Travis Webb (Labrador)

Questions / Comments