Satsy Design Document

Team

May14_13

Date

Sunday, April 28, 2013

Members

Carl Chapman
Cody Hoover
Cole Groff
Kaitlin McAbee
Trevor Lund

Advisor

Kathryn Stolee

Introduction

Project Definition

This project is a web-based search engine for source code called Satsy. Satsy is based on the fact that every program has one or more executable paths, and that Satisfiability Modulo Theory (SMT) solvers can be used to evaluate whether or not a particular executable path in a program can satisfy an input-output specification.

For testing, a small set of programs have had their paths encoded into the SMT-LIB2 [1] format. Users will interface with Satsy client through the browser to send queries in the form of input-output pairs. On the server side, Satsy will combine these user-defined input-output pairs with the complete set of translated source code methods and execute Microsoft's Z3 [2] SMT solver on each combination. Satsy will parallelize the execution of Z3 on these combinations so this search can be done more quickly.

The results of Z3's execution on each program path, input-output specification pair are collected and returned to the user. The intention of this project is to rank these results based on the number of input-output pairs a particular program was able to satisfy, and other related metrics to be explored in the future. Once ranked results are displayed to the user, the user will be able to refine their search and search again as desired.

Project Goals

- Satsy should be able to return source code that satisfies the user-specified input-output pairs, ranked by its ability to satisfy the specification.
- Satsy should be able to conduct its search quickly and begin to return a user's search results in less than 5 seconds per search.
- Multiple users should be able to run searches at the same time. Satsy should be able to support up to 10 concurrent users at a time with modest server resources, and this number should scale with a more capable server.
- Satsy should perform consistently as the number of code snippets to search increases.

Deliverables

- A flexible and extensible platform to perform usability studies to validate Satsy as a viable source code search engine.
- Code containing search logic that parallelizes execution of Z3 and combines results.
- An algorithm to sort the results of the Z3 search in an order that brings the most relevant blocks of source code to the top of the results.
- A single-page web-client that sends queries to the server and displays results to the user.

System Level Design

System Requirements

Functional Requirements

- Satsy will be able to accept multiple input-output pairs as search terms from each user, up to 10 pairs.
- Satsy will be accessible from anywhere that has internet access.
- Satsy will return a set of source code snippets given a set of input-output pairs.
- The source code that Satsy returns should be able to take the given inputs as parameters and produce the given output as a returned result.
- The source code results nearer to the top of the search result page should be satisfiable by most or all of the input-output pairs.
- Source code results that are satisfiable by all of the input-output pairs will be sorted by other criteria to be specified later.
- Satsy needs to provide a clean interface to interact with the user, to be determined by user feedback.
- Satsy will provide feedback to the user if they didn't format their search terms correctly or if an error occurred.
- Satsy will display an input search box and an output search box.
- Users will be able to add additional sets of search boxes on the search page.
- The system will handle Java string, int, boolean, and character inputs, and allow extensions for further language support.

Non-Functional Requirements

- The number of threads per search should be configurable so that it can match the resources on the computing platform wherever Satsy is deployed.
    - Metric: A configuration value 'nThreadsPerSearch' controls the number of threads per search.
- Logging should provide information on the health and effectiveness of the system
    - Metric: Trace and detail level logs will indicate search times, errors, and information about the results returned.
- Users should quickly be shown the results of their search upon submitting their search criteria.
    - Metric: There shall be no more than 6 seconds between the user clicking the search button and the user beginning to see their search results.
    - Metric: Timeouts on at the overall search and individual solver level will keep searches finite in length.
- Satsy needs to support returning many different source code results.

    - Metric: Up to 1000 source code results can be returned.
- Satsy will be able to handle multiple user searches at once.
    - Metric: Up to 10 searches at once will be supported.
- Results should be accurate.
    - Metric: Results should return with a false positive rate of <15% and a false negative rate of <5%.

Functional Decomposition

High Level Decomposition

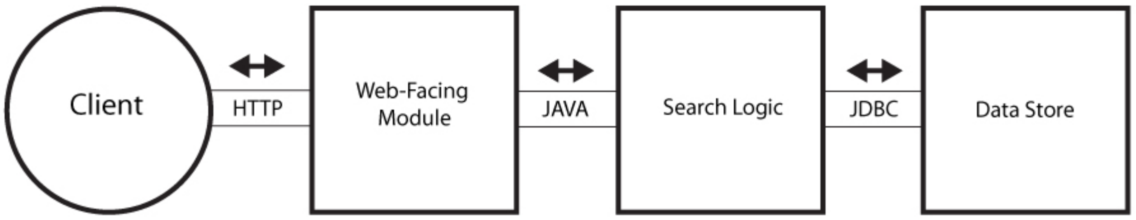


Figure 1: Functional Decomposition

At a high level, Satsy can be decomposed into three modules. The web-facing module interacts with the client through HTTP, and communicates with the search logic module through Java. The search logic module communicates with the web-facing module through Java and with the data store module using JDBC. This architecture closely matches the ‘state-logic-display’ pattern.

Web-Facing Module

The Web-Facing Module module accepts input from the user and displays results, using the three core functions of Satsy: 'Initialize Search', 'Poll' and 'Request Update'.

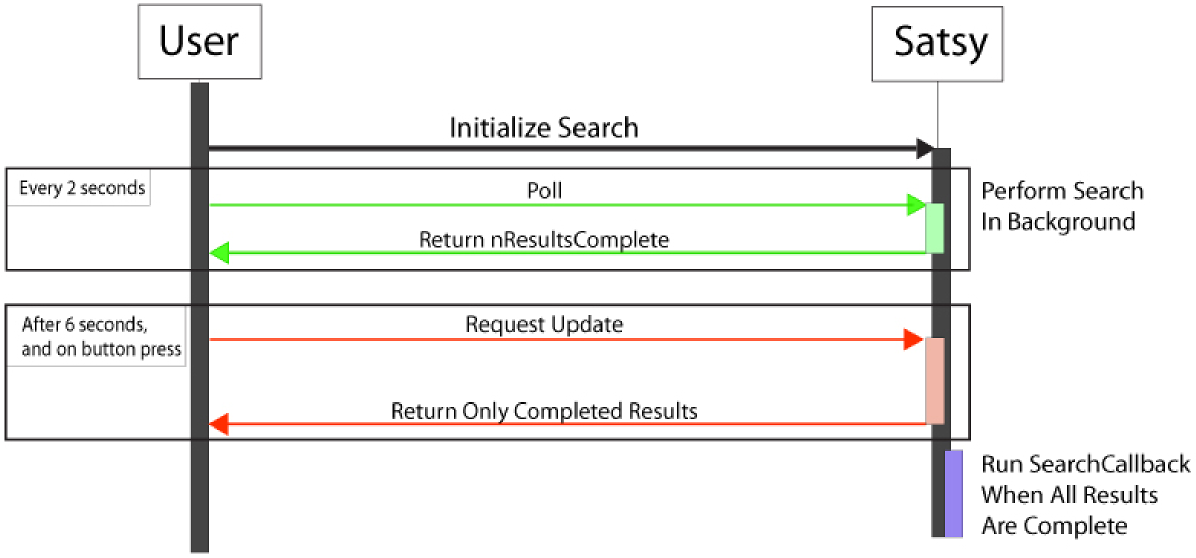
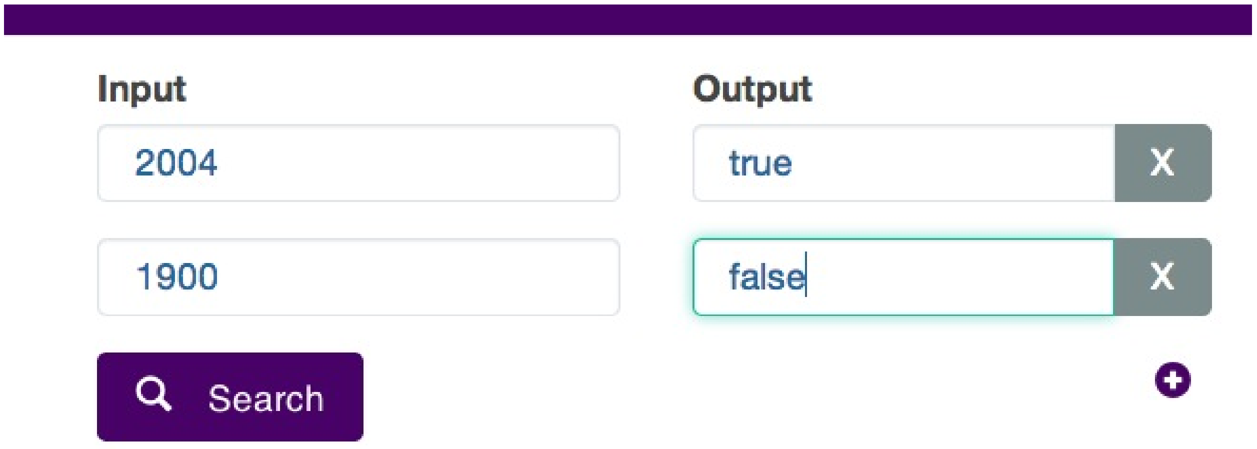

User
Satsy
Initialize Search
Every 2 seconds
Poll
Return nResultsComplete
Perform Search In Background
After 6 seconds, and on button press
Request Update
Return Only Completed Results
Run SearchCallback When All Results Are Complete

Figure 2: Satsy Sequence Diagram

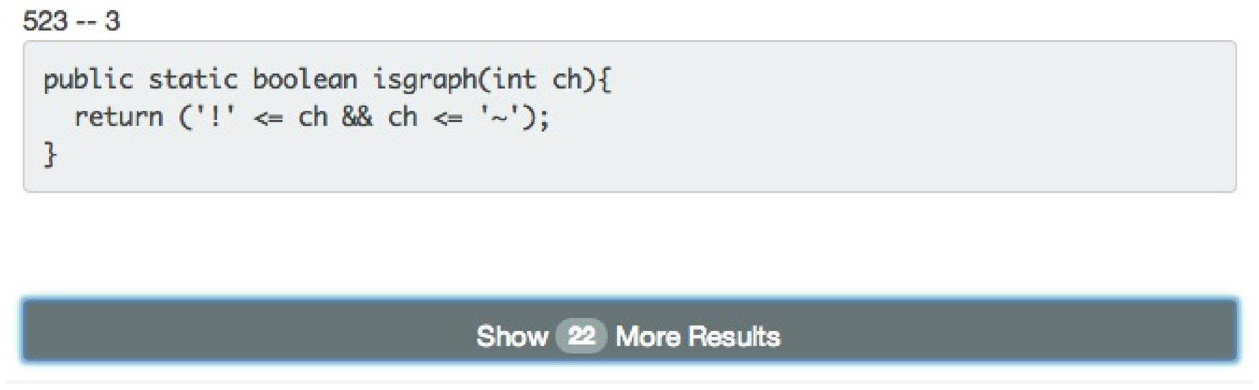
When users enter input-output pairs (IOPairs) in the form and press 'Search', the 'Initialize Search' function of the Search Logic module is called as shown in Figure 2.



Input	Output
2004	true
1900	false

Figure 3: IOPair examples

Every 2 seconds, JavaScript in the client's browser automatically calls the 'Poll' function of the Search Logic module, which simply returns the number of completed results. The number of completed results obtained by the Poll function is displayed in a badge on the 'Show More Results' button as shown in Figure 4. When the user presses this button, the 'Request Update' function of the Search Logic module is called, which creates a list of all the completed results, ranks them and returns them for display. The 'Request Update' function is also called automatically 6 seconds after the 'Search' button is pressed as a convenience for the client.



523 -- 3

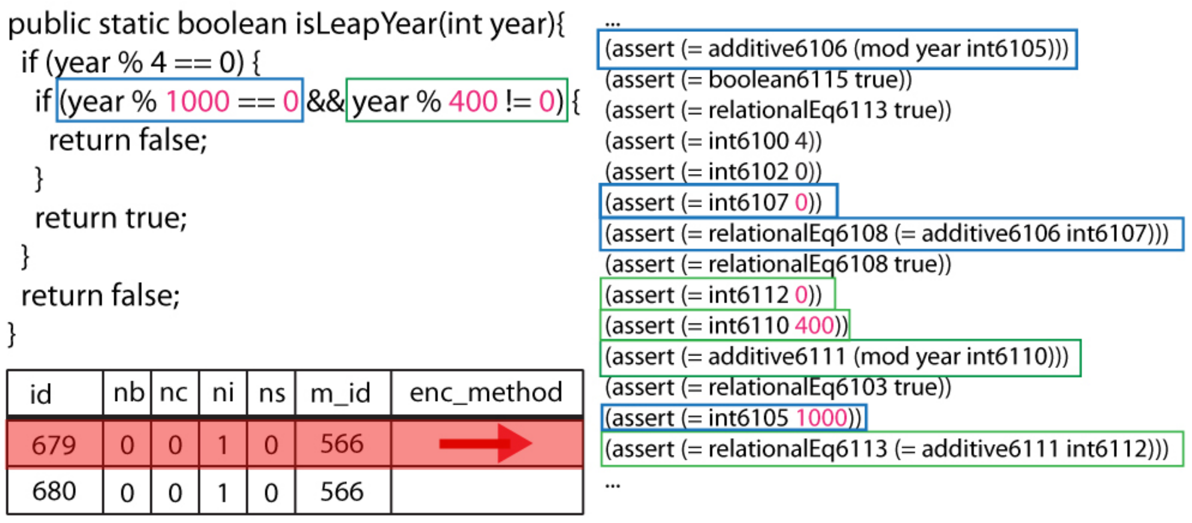
```
public static boolean isgraph(int ch){
  return ('!' <= ch && ch <= '~');
}
```

Figure 4: Result and "More Results" Indicator

After a user has pressed Search, they can add, remove or change search parameters to modify their search. If the user presses 'Search' with new parameters, everything about the previous search is discarded and an entirely new search process is performed just like the initial search.

Data Store Module

Satsy searches a database of source code snippets. Each source code snippet is represented in two separate ways: the source code text, and all of the unique equivalent paths through the program. There can be many unique paths through a single method.



```
public static boolean isLeapYear(int year){
 if (year % 4 == 0) {
  if (year % 1000 == 0 && year % 400 != 0) {
   return false;
  }
  return true;
 }
 return false;
}
```

id	nb	nc	ni	ns	m_id	enc_method
679	0	0	1	0	566	→
680	0	0	1	0	566	

```
...
(assert (= additive6106 (mod year int6105)))
(assert (= boolean6115 true))
(assert (= relationalEq6113 true))
(assert (= int6100 4))
(assert (= int6102 0))
(assert (= int6107 0))
(assert (= relationalEq6108 (= additive6106 int6107)))
(assert (= relationalEq6108 true))
(assert (= int6112 0))
(assert (= int6110 400))
(assert (= additive6111 (mod year int6110)))
(assert (= relationalEq6103 true))
(assert (= int6105 1000))
(assert (= relationalEq6113 (= additive6111 int6112)))
...
```

Figure 5: An example of an encoded path through a source code snippet

As shown in Figure 5, the method ‘isLeapYear’ has two encoded paths in the database, each of which has a different enc_method String where a particular path through the method has been encoded in SMT. The blue and green boxes surround logical expressions as encoded in java and SMT.

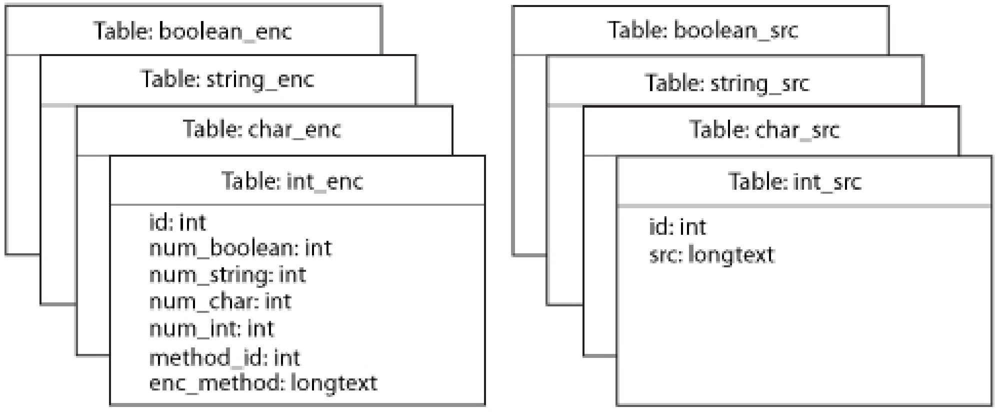


Figure 6: Database Schema

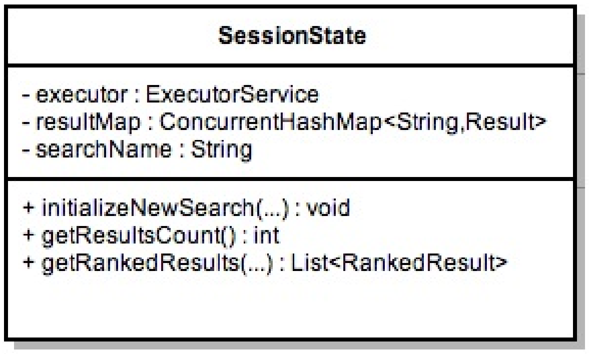
Because each method's output is always one of the four supported types (booleans, strings, chars and ints), the encoded and source code tables are partitioned by output type as shown in Figure 6

Search Logic Module Overview

For each single search that is initialized, many solvers are started. All of the encoded paths whose input-output signatures match the given IOPairs are evaluated to see if the path can satisfy the given IOPair. Each unique evaluation should evaluate to SAT or UNSAT (unless errors occur). If an evaluation results in SAT, that means that the IOPair will be satisfied by the encoded path.

Search Logic Module Detail

The Search Logic Module performs three core functions: 'Initialize Search', 'Poll', and 'Request Update'. When a client connects to Satsy for the first time, the Satsy servlet uses the Session ID provided by Tomcat to create a SessionState object associated with that user. This SessionState object has an ExecutorService and a ConcurrentHashMap that maps each method id to a Result object as shown in Figure 7.


SessionState
- executor : ExecutorService
- resultMap : ConcurrentHashMap<String,Result>
- searchName : String
+ initializeNewSearch(...) : void
+ getResultsCount() : int
+ getRankedResults(...) : List<RankedResult>

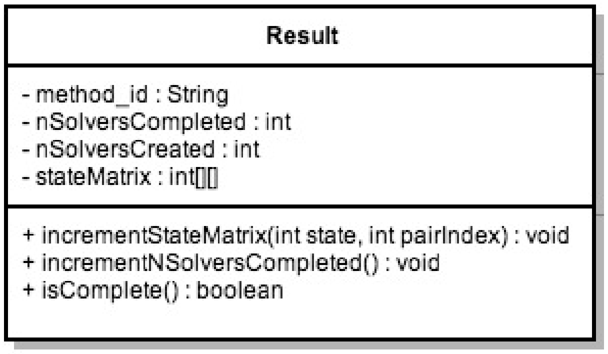

Result
- method_id : String
- nSolversCompleted : int
- nSolversCreated : int
- stateMatrix : int[][]
+ incrementStateMatrix(int state, int pairIndex) : void
+ incrementNSolversCompleted() : void
+ isComplete() : boolean

Figure 7: Partial class diagrams for SessionState and Result classes

The Result object is where the outcome of individual Z3 executions are recorded. A Result is complete only when all the solvers created for that result have completed.
The 'Poll' function uses the SessionState's getResultCount() method, which simply iterates through the resultMap and counts how many Results are complete. The 'Request Update' function makes a new list of results that are complete, ranks them and returns the ranked list to the Web Facing Module.
The 'Initialize Search' function uses the initializeNewSearch() method. When the initializeNewSearch() method is called, the old executor is stopped and a new executor, resultMap and searchName are created. An SQL query is performed that obtains all the encoded paths that have the same input-output signature as the IOPairs. The new, empty resultMap is populated with one empty Result for each unique method id contained in the outcome of the SQL query. An AtomicSearchSolver is created for each IOPair-encoded path combination. See Figure 8.

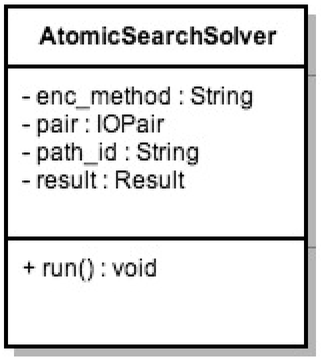


Figure 8: Partial class diagram for AtomicSearchSolver

Once all of the AtomicSearchSolvers are created, they are submitted as a group to the executor, which handles the task of running the solvers using a configuration-specified number of threads.

Each solver goes through four steps:

1. Check to see if the search has timed out.

2. Create a temporary smt file that combines the IOPair and the encoded path.
3. Build a Z3 process and execute it, waiting for the output stream.
4. Read the output of the stream and record the end state of the solver in the Result.

When all of the results in the resultMap are completed, a callback is executed that can do whatever testing or development tasks are desired like logging, profiling, validation, etc.

Standards

- **SMT-LIB Standard Version 2.0**
  The SMT-LIB Standard version 2.0 [1] is used in creation of SMT files. These files are used by Z3 [2] to determine if certain code segments satisfy given search criteria. Z3 is a theorem prover developed by Microsoft. Given a system of values, functions, and constraints, Z3 can determine if the system is satisfiable or not.

System Analysis

The process of using the Z3 SMT-Solver is known to be NP-Hard. As such, an exhaustive search of the problem space increases dramatically as the size of the problem space increases. In our case, this problem space is the database of methods to search. To balance this, the speed and number of processors need to increase as that database increases. In some ways, the utility of the system is directly related to the size of the database, which results in managing three primary variables: cost, performance, and utility. The current database is relatively small, but the low performance free server solutions such as Amazon AWS do not meet our requirements. A low-cost dedicated commodity server has demonstrated that it provides the necessary performance at a reasonable cost for the current database size with room to grow.

Detailed Description

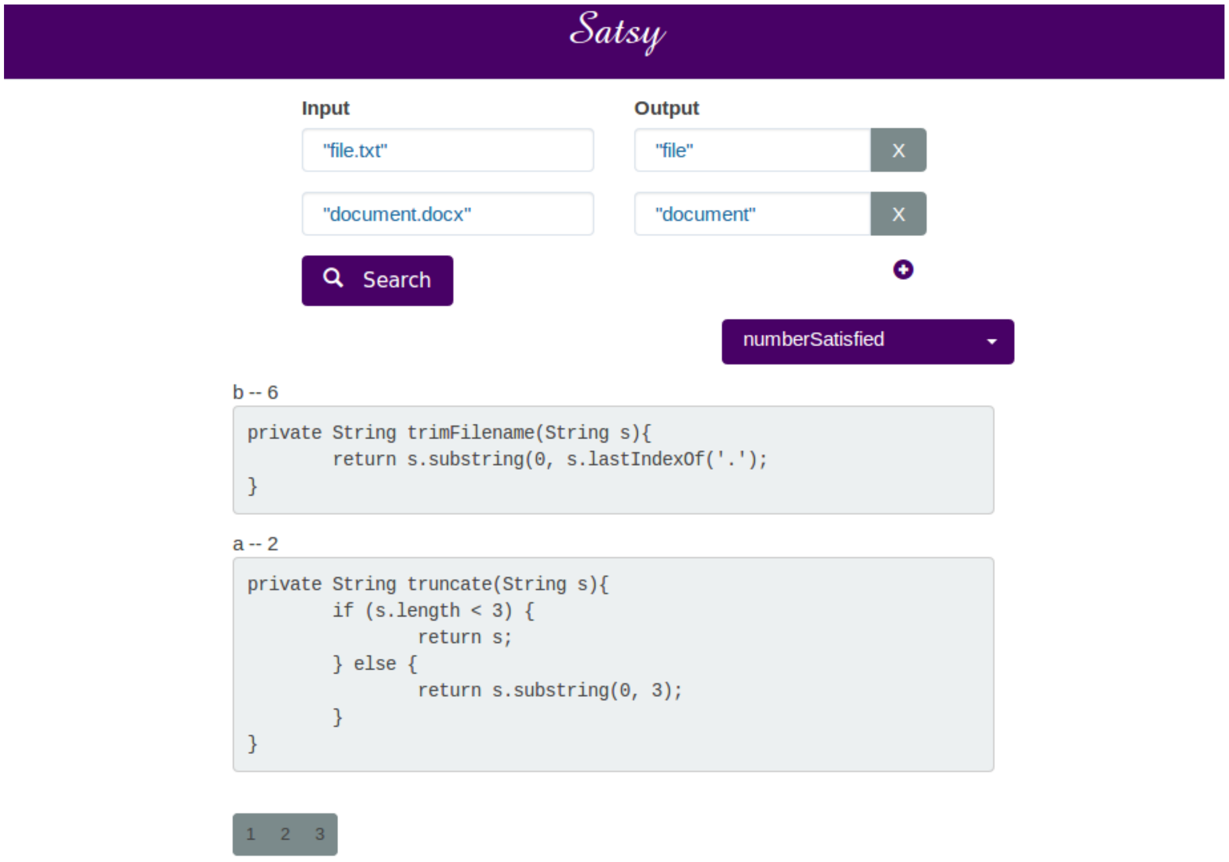
Interface Specifications

The user interface will be a single page web-app with two primary stages:

1. Search Stage: users will enter input/output strings representative of the desired behavior. Results will be returned when the user hits search.

2. Results Stage: the user's query will remain visible and, if modified, will allow the user to requery. Below this will be an area of paginated results. There will be a counter that lets the user see how many additional results the server has found. When this is clicked the results will be returned and the displayed order will

change to reflect a combined ranking of all of the previously returned results and the newly returned results. The results will show the method signature (if applicable) and the code, as well as relevant ranking information.



b -- 6

```
private String trimFilename(String s){
        return s.substring(0, s.lastIndexOf('.');
}
```

a -- 2

```
private String truncate(String s){
        if (s.length < 3) {
                return s;
        } else {
                return s.substring(0, 3);
        }
}
```

Figure 9: Web Client with Search Area and Results Section

Hardware/Software Specifications

- **System Overview**

  The system consists of a software component that can run on most modern computers and a piece of commodity hardware to host it. This will be managed and run out of Dr. Kathryn Stolee's office.

- **Hardware Specifications**
  - Mac Mini with OS X Server
    - i. 4GB Memory
    - ii. 2.3GHz Quad-core Intel Core i7
    - iii. 2 x 1TB HDD
    - iv. OS X Server
- **Software Specifications**
  - Apache Tomcat 7.0.26
  - Z3 High-Performance Theorem Prover 4.3

- MySQL Server 5.5.32
- Java 1.6

Validation and Verification

I. Validation:

Validation answers the question of "did we build the right thing?" as well as making sure that the product satisfies the client. To ensure that we are building the correct project, we will have both requirement and design reviews.

A. Requirement review:

1. The client will look over the product at its current stage to confirm that it meets the requirements as expected. If not, adjustments can be made in the next iteration.
2. To ensure the everyone is on the same page for all of the requirements, we will talk over what is planned for the next iteration. If there is any confusion for a given requirement, we will discuss with the client until the issue has been clarified.

B. Design review

1. For every iteration, we will meet with our client to have her look over the product at its current stage in production. This will be to get feedback on if the changes we have made during the last iteration are going in the correct direction.
2. At the end of every iteration, each member of the team will present a brief summary of their work to update the entire team.

II. Verification:

Verification answers the question " did we build the product right?" as well as testing the product for bugs in the system.

A. Documentation

To help with the review and understanding of the code, the code should be well commented, including having Javadocs for each function. Major algorithms should be explained step-by-step in the comments.

B. Testing

1. JUnit testing
    a) All Plain Old Java Objects (POJOs) should have unit tests
    b) Ranking modules should be unit tested.
2. Manual testing
    a) The Web Facing Module should be able to be tested independently of the Search Logic Module using mocks and debugging flags.
    b) The Search Logic Module should be tested for correctness after every major change, using debugging flags.
3. Automated testing
    a) Verifiation of results should be checked using the ProtoResultMap

and SearchCallback features

b) Profiling of the system should be done to inform future optimization efforts.

Testing Results

Validation

As planned, we continued to communicate with the client to make sure that the system would satisfy her needs.
At the beginning of the second semester, we had several meetings with the client where we discussed the performance of our initial version of Satsy. After some discussion, we came up with the new system that provides partial results as the search continues in the background.

Verification

- JUnit Testing
  - 100% of JUnit tests for Ranking and POJOs have passed at each iteration.
- Manual Testing
  - The Web Facing Module has been manually tested at each iteration, and the system behaves as desired.
  - The Search Logic Module has been manually tested at each iteration, and the system behaves as desired.
  - Manual inspection showed some database entries to have incorrect input-output mappings, which resulted in unexpected problems. The system was modified to handle problems with database entries more gracefully, treating them as errors. Later, these entries were repaired by our client, who is the only person who knows how to generate encoded paths from source code snippets.

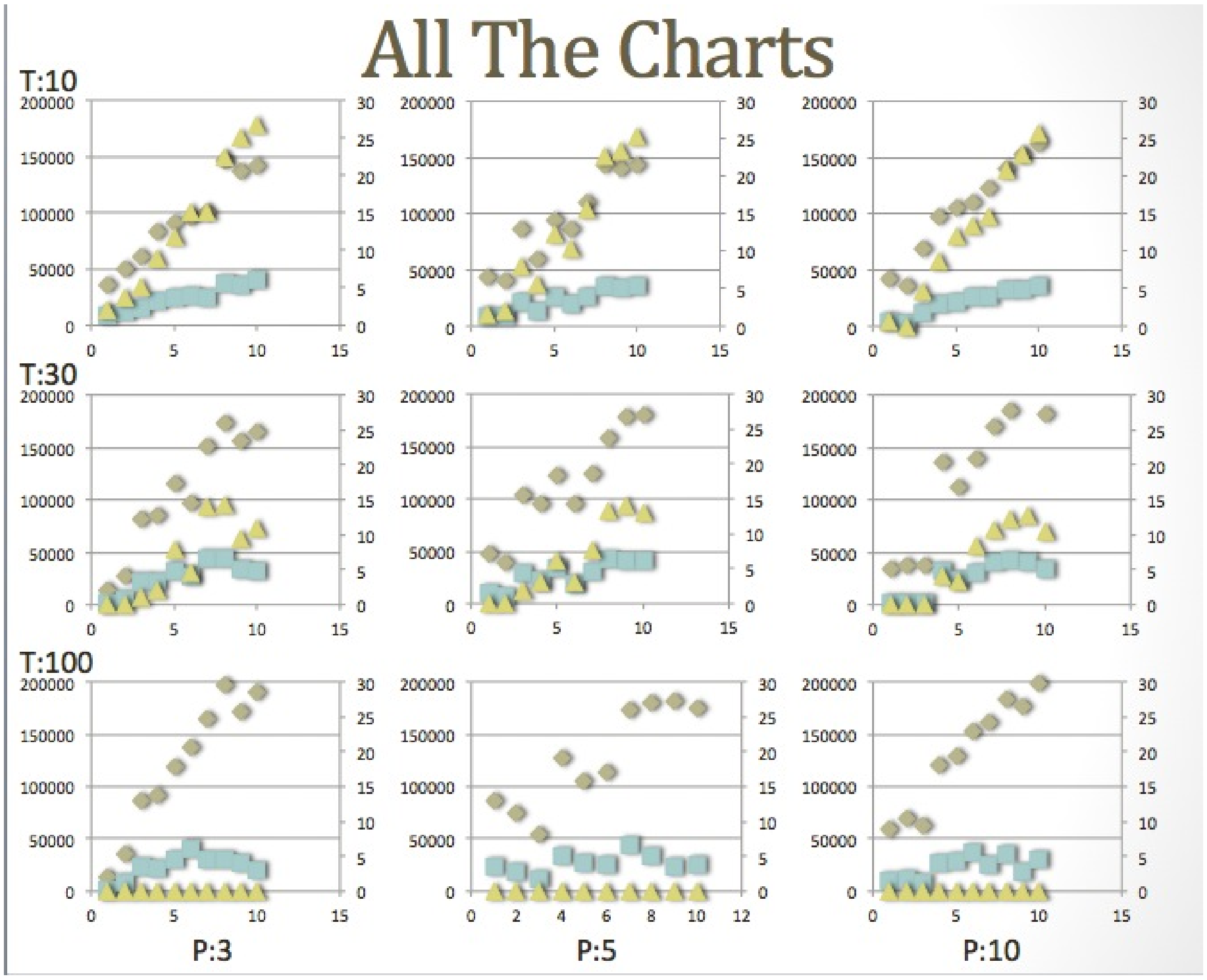


Figure 10: Results of Automated Chart-Writing program run on the AWS-based Satsy system performed in January 2014.

- Automated Testing
    - Correctness validation on the initial system showed that the only incorrect results were a result of timeouts caused by AWS cpu throttling NOTE FIGURE.
    - Automatic Verification of the new system is still in development at the time of this report.

Automated Testing Results Explained

Each of the 9 charts has three series: the brown diamond series which is the total number of milliseconds per user for a set of searches, the blue square series which is the average number of milliseconds per user search, and the yellow triangle series which is the total number of failures per run. Each node of each series is the result of 3 averaged test runs. The values on the x-axis convey the number of users. The values on the y-axis convey number of failures for the yellow triangle series, and milliseconds for the other two series. The capital ‘T’ values varying vertically in the image denote different timeout values in seconds. The capitol ‘P’ values varying horizontally in the image denote different pauses between the searches in seconds. Some distortion in these charts is due to the fact (unknown until after the charts were made) that the free tier of AWS is cpu-throttled at some regular interval. The most interesting result found by this investigation was that a low timeout value is the sole cause of failure. This is the case because the threads performing the search are not able to finish because the cpu is throttled by AWS about every 5 or 6 minutes. This means that the first few searches consume most of the CPU time provided for the 5 minute interval, and then the rest of the searches proceed much more slowly.

Implementation Issues and Challenges

- **Learning how to use Amazon Web Services (AWS)**
  Amazon Web Services was used to setup a web server which allows clients to interface with the Satsy system. Amazon Web Services provides free-tiered versions of their Elastic Compute Cloud (EC2), which can be used to run micro instances with server-oriented operating systems. After figuring out how to signup for and set up a micro instance, it is easy to install a web server which provides the interface necessary for clients to interface with the Satsy system. The biggest challenge with setting up an EC2 free-tiered micro instance is digging through Amazon’s convoluted trove of AWS documentation to find out how to setup and use an EC2 micro instance. Once this hurdle is overcome and an EC2 micro instance is set up and working, everything becomes smooth sailing.
- **Configuring Apache Tomcat to run on port 80**
  Apache’s Tomcat web server software runs on port 8080 by default. The standard port for serving content via HTTP is 80. Changing Tomcat’s port to 80 is as simple as changing an XML configuration file. What can be challenging, though, is when another web server is already using port 80. An example is when using Apache’s standard HTTP web server along with Tomcat. This can cause configuration headaches trying to get the two to work together. Fortunately, Satsy’s web server is only running Tomcat and there is no need to get two different web servers working nicely together.
- **Configuring development environment using Eclipse and Tomcat**
  The easiest way to set up a development environment which allows Eclipse and Tomcat to work well together is to use Eclipse’s latest version of their IDE for Java EE Developers (Kepler Service Release 1). This version of Eclipse comes with the Apache Tomcat 7.0 adapter installed. Installing this adapter with older versions of Eclipse can be cumbersome. With the Apache Tomcat 7.0 adapter already installed the only other setup needed is to let Eclipse know where Tomcat’s binary directory is located. Once Eclipse knows the location of Tomcat, server startup and web application execution are as simple as pressing a “go” button.

- **Setting up a Bitbucket repository to allow collaborative development**
  Working in a collaborative development environment sometimes requires a central repository for storing and maintaining project code and files. Bitbucket is a service that allows users to store and access Git repositories. One challenge is figuring out how to get past Bitbucket's maximum allowed repository collaborators, which is set at 5. Fortunately, Bitbucket allows users with educational email addresses (ending with .edu) to have an unlimited number of collaborators.

- **Getting JDBC to work with MySQL**
  JDBC (Java Database Connectivity) can be used to allow Java code to communicate with and use MySQL. It can be challenging figuring out what version of the JDBC driver is necessary for certain projects and how to integrate them with Java code.

- **Converting Redis database to MySQL database**
  Redis is an in-memory key-value data store, which was being used in Dr. Stolee's prototype to store encoded methods and source code. Because the product needs to serve multiple clients, handling concurrent requests effectively, MySQL was chosen as the data store technology for the project. Some effort was required to transition from using Redis to using MySQL to store encoded methods and source code. These challenges included learning about Redis, redesigning the schema, and tracing through prototype code so that the existing encoded methods and source code could be exported into the new MySQL data store.

- **Converting System from Synchronous to Asynchronous Model**
  The system was originally designed to behave synchronously such that a client would make a query and wait for a complete result set to be returned. It became obvious that our search was taking far too long so the system was redesigned. This brought with it a lot more complexity which caused challenges in implementation. It created more points of interaction between the server and the client, timers to keep track of, and a more complex user experience to manage.

Appendices

Appendix I: Operation Manual

1. Install Tomcat and MySQL on your server
   a. First you need to download the binary file for the correct version of Tomcat from http://tomcat.apache.org complete the installation.
   b. For MySQL, you need to download it from http://dev.mysql.com/downloads/
      i. The context.xml file of Tomcat should have these tags:

```
<context-param>
```

```
          <description>an external testing switch for detailed examination of the
inner states of the search process</description>
   <param-name>detail</param-name>
   <param-value>false</param-value>
</context-param>
<context-param>
          <description>an external debugging switch for GUI code</description>
   <param-name>guiDebug</param-name>
   <param-value>false</param-value>
</context-param>
<context-param>
          <description>The maximum search duration in
Milliseconds</description>
          <param-name>maxSearchDurationMS</param-name>
   <param-value>100000</param-value>
</context-param>
<context-param>
          <description>The number of threads to create per
search</description>
   <param-name>nThreadsPerSearch</param-name>
   <param-value>8</param-value>
</context-param>
<context-param>
          <description>an external debugging switch for solver
code</description>
   <param-name>solverDebug</param-name>
   <param-value>false</param-value>
</context-param>
<context-param>
          <description>an external testing switch for tracing the most critical
milestones of the search process</description>
   <param-name>trace</param-name>
   <param-value>true</param-value>
</context-param>
<context-param>
          <description>the absolute path to the folder containing the z3
executable where temp files will be written</description>
   <param-name>z3Path</param-name>

<param-value>/Users/carlchapman/Documents/Schoolwork/ISU7_fall_2013/4
91/z3stuff/bin</param-value>
</context-param>
```

```
<context-param>
         <description>the number of seconds to allow z3 to run on an atomic
search until giving up</description>
   <param-name>z3TimeoutSEC</param-name>
   <param-value>10</param-value>
</context-param>
```

ii. The web.xml file of Tomcat should have these tags inserted after the default servlet
(The default servlet is shown here so do not copy the <servlet> tag)

```
<servlet>
   <servlet-name>default</servlet-name>
   <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
         <init-param>
      <param-name>debug</param-name>
      <param-value>0</param-value>
         </init-param>
         <init-param>
      <param-name>listings</param-name>
          <param-value>false</param-value>
         </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>

<context-param>
   <param-name>Z3Path</param-name>
   <param-value>/home/ubuntu/z3</param-value>
</context-param>
```

c. External JAR files
   i. Tomcat looks for JAR files in the folder /usr/share/tomcat7/lib
   ii. When a new JAR file is added, Tomcat must be restarted
   iii. List of External JAR dependencies
      1. GSON (http://code.google.com/p/google-gson/downloads/list)
      2. mysql-connector-java-x.x.x-bin.jar (http://dev.mysql.com/downloads/connector/j/)

d. Z3 directory and executable permission with Tomcat
   i. Change the z3 directory and all of its contents to have 'tomcat7' as the owner and 'staff' to be the group.
   ii. Add users 'tomcat7' and 'ubuntu' to the group 'staff'.

iii. Change the z3 directory and all of its contents to have the permissions value ‘770’. This restricts read/write/execute to only the owner and members of the group.

iv. Restart the server.

2. mysqldump import
    a. Log into mysql as root
        i. shell-prompt#mysql –u root –p
    b. Create a database called satsy
        i. mysql-prompt>create database satsy;
    c. Exit mysql
        i. mysql-prompt>exit
    d. Load the dump file
        i. shell-prompt#mysql –u root –p satsy < satsy.sqldump
3. Deploy .war file
    a. Right-click on project in eclipse > Export > WAR File (needs J2EE eclipse).
    b. Go to your server’s manager location.
    c. Login using manager-gui credentials.
    d. Go to Deploy > WAR file to deploy.
    e. Select WAR file to upload.
    f. Click Deploy button.

Appendix II: Alternative Versions

- **Initial Synchronous Model**

The system was originally designed to behave synchronously such that a client would make a query and wait for a complete result set to be returned. Each time the user hit “search”, an HTTP GET request was sent to the server which handled it in a doGet method. This handled starting our search sequence and waiting for the entire search space to be tested, and then matching results would be ranked and returned to the client which was blocked and waiting for these results. This process took several minutes to complete, and in some cases took significantly longer. This was clearly unacceptable performance and led to the redesign of what we have now.

- **RESTful API Model**

Understanding that this was to be a web accessible solution, it seemed useful to have alternative ways to query for results programmatically. To accomplish this, the server could respond to HTTP requests sent to well defined URL’s conforming to RESTful API conventions to return results. This seemed feasible and advantageous on top of our initial synchronous model described above, but became too complex to implement in time on top of the new asynchronous model.

Appendix III: Other Considerations.

- **NP-Hard is Hard**

Despite having what we thought was a small problem space, the NP-Hard nature of the problem caused very noticeable performance issues. It made it very clear the importance of heuristics and approximations, concurrent searching, and reduction of the problem space. There is no “fast” way to solve our problem, but there are ways to get around that.

- **AWS Throttling**

Our initial platform for development and testing used the free tier of Amazon Web Services’ Elastic Compute Cloud platform (Amazon EC2). While in the middle stages of development we noticed that Satsy’s search query responses would become very slow or halt altogether. At other times, query responses would arrive more quickly. Through investigation and recording of response times, it became evident that the Amazon EC2 service was implementing some form of CPU throttling. We were being granted a certain amount of CPU resources every 5-6 minutes. Once these resources were consumed (which took about 15 seconds of 1 user searching), we were left with whatever spare CPU bursts were available.

References

[1] http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r12.09.09.pdf, http://www.smtlib.org/
[2] http://z3.codeplex.com/
[3] http://aws.amazon.com/ec2/
[4] http://releases.ubuntu.com/precise/
[5] http://tomcat.apache.org/tomcat-7.0-doc/
[6] http://www.mysql.com/
[7] http://www.oracle.com/technetwork/java/javase/jdbc/index.html
[8] http://getbootstrap.com/2.3.2/