Bridging Formal Methods and Data Science

Vijay Murali
Rice University

IEEE MetroCon 2015

Slides courtesy of Swarat Chaudhuri
Data Science / Data mining

Algorithms and tools to extract knowledge or insights from large volumes of data
Formal methods

Algorithms and tools to model, design and mathematically analyze complex computational systems
Find inputs on which a program doesn’t terminate

Prove that a program satisfies an assertion

Generate a program from a set of test cases

Fix a program so that it no longer violates an assertion

Show that a program uses an API incorrectly
There’s much progress...

... in static analysis, dynamic analysis, type systems, model checking.
But more needs to be done

Average number of bugs per 1000 lines of code is 2.3x greater in 2013 (.69) than in 2008 (.3)

—2013 Coverity Scan report
Can data mining help?
Formal methods vs. Data mining

- System
  - Model
    - Inference
      - Logical proofs
- Data
  - Model
    - Inference
      - Probabilistic predictions
Formal methods vs. Data mining

System

Model

Inference

Logical proofs

• Rules that run the system (semantics)
• What the user wants (specification)

Inference

Probabilistic predictions
Formal methods vs. Data mining

**Formal methods**
- Logical proofs

**Data mining**
- Probabilistic predictions
- Inference
- Model
- Data

Rules that explain a dataset (probability distribution)
Why data matters: Specifications

• Formal methods rely on specifications
  – “Prove that a program satisfies its assertions”

• Formal specifications can be hard to write
  – Formalize how to use the Facebook API!
  – Formalize the requirements for a self-driving car!
Why data matters: Specifications

Here’s how I do it!

To see what I do, read my code!

And here’s my method!

How should I use the POSIX API?
Notions of correctness can be democratically elected
Why data matters: Algorithmic issues

• Theorem proving is computationally hard

• Too many...
  – system states
  – implementations for a specification
  – candidate proofs, bugs, fixes

• Needle in a haystack
Why data matters: Algorithmic issues

I need to fix my code

Here’s how I do it!

To see what I do, read my code!
The crowd can lend you algorithmic insights
Programs and proofs are just forms of data

Lots of open-source code to learn from
Big Code

Number of open-source projects on GitHub
Big Code

Source: ohloh.net
Formal methods
+ Big Code
THE $11M TOOL THAT COULD HELP COMPUTERS WRITE THEIR OWN CODE

DARPA funds $11 million tool that will make coding a lot easier

COMING SOON: AUTOCOMPLETE FOR PROGRAMMERS
DARPA WANTS TO FIND A WAY TO AVOID WRITING THE SAME CODE OVER AND OVER.

By Dan Moren   Posted November 11, 2014
The Pliny approach

- Synthesis
- Verification
- Debugging and Repair

Magic in the cloud!
The Pliny approach

Interactive mode

Pliny Reasoning Framework
(Big Compute + Small Data)

Batch mode

Pliny Statistical Database
(Big Compute + Big Data)

User

Pliny user interface

Pliny Language Framework

Code corpora

GitHub

source forge

debian

Pliny

User Code

corpora
The Pliny approach: Batch mode
The Pliny approach: Interactive mode

- Statistical database
  - Inference on program abstractions

- Query (features)
- Guidance
  - Specifications
  - Algorithmic insights

- Reasoning framework
  - Combinatorial search
  - Deduction
  - Symbolic data structures

- User
- Underspecified task
- Feedback/Interaction

- Pliny Language Framework
- Features
- Goals

- How should I use the POSIX API?
The rest of the talk

- The Pliny user interface
- Pliny in action (1)
- The Pliny language framework
- The Pliny statistical database
- The Pliny reasoning framework
- Pliny in action (2)
Stratocode: The Pliny user interface

- Write code in browser, run on the cloud
- Originally developed for MOOCs by Barnett, Cox, Rixner and others
The rest of the talk

- The Pliny user interface
- Pliny in action (1)
- The Pliny language framework
- The Pliny statistical database
- The Pliny reasoning framework
- Pliny in action (2)
Data-driven program synthesis

```c
int binsearch
(int x, int v[], int n)
{
    int low, high, mid;
    low = 0;
    high = ??;
    while (low <= high) {
        ??
    }
    return -1;
}
```

Led by John Feser, Yanxin Lu, and Vijay Murali
Data-driven program repair

```c
int binsearch
(int x, int v[], int n)
{
    int low, high, mid;
    low = 0;
    high = n - 1;
    while(low < high) {
        mid = (low + high) / 2;
        if(x < v[mid]) {
            high = mid - 1;
        } else if(x > v[mid]) {
            low = mid + 1;
        } else {
            return mid;
        }
    }
    return -1;
}
```

Incorrect program

Test cases/constraints

Corrected program

Led by Yanxin Lu and Hassan Eldib
int binsearch (int x, int v[], int n)
requires v != null && 0 <= v.Length;
requires Sorted(v);
ensures 0 <= index ==>
    index < a.Length && a[index] == value;
ensures index < 0 ==>
{
    ...
    while (low <= high)
        invariant ??
        {
            ...
        }
    return -1;
}
Learning temporal specifications

fd1 = open("file1.txt");
fd2 = open("file2.txt");
while ((s = read(fd1, 100)) != 0) {
    write(fd2, s);
lseek(fd1, 100);
}
close(fd1);
close(fd2);

How should one use the POSIX file API?

Led by Vijay Murali and Rohan Mukherjee
Data-driven superoptimization

```
void stenld (float[4,N] X) {
    for (int t=1; t<4; ++t)
        for (int i=1; i<N-1; ++i)
            X[t, i] = X[t-1, i-1] + X[t-1, i+1];
}
```
The rest of the talk

- The Pliny user interface
- Pliny in action (1)
- The Pliny language framework
- The Pliny statistical database
- The Pliny reasoning framework
- Pliny in action (2)
The Pliny Language Framework

Functions, types, classes, ...

Program elements + Feature vectors + Similarity measures
Features in Pliny

Code Corpus

Extract useful fragments and their abstracted traits

Program Element

- Code fragment
- Invariants
- Tracelets
- Constants
- Natural language
- Skeletons
- I/O samples
- Type signatures
- API traces

- node
- edge
- graph
- subgraphs
- types & operations
- (int, /)
- (float, *)
- (int*, *)
Feature example

Control Flow Graph

4-subgraphs (DFS)
The rest of the talk

- The Pliny user interface
- Pliny in action (1)
- The Pliny language framework
- The Pliny statistical database
- The Pliny reasoning framework
- Pliny in action (2)
The Pliny Statistical Framework

Large-scale machine learning on program features

- k-nearest neighbors
- Markov random fields
- HMMs and generalizations
- ...

Implemented on a new distributed object store + compute platform (PDB)
The Pliny Database (PDB)

- Flexibility of Hadoop/Spark
  - User-defined types
    - lists, graphs, trees
  - User-defined computations
    - Top-k similarity search
    - Gradient descent
    - Hidden Markov models

- Performance of relational databases
  - Schemas allow control over data movement, layout
Programming model

```cpp
class Foo : public PDBStoredDataType{
    int a;
    int b;
};
```

```cpp
class Bar : public PDB StoredDataType{
    int c;
    int d;
};
```

PDBClient::addData()

Data Insertion

PDB

cloud-based cluster

node 1

node 2

node 3

node 4

PDBClient::runQuery()

Queries, Updates, Deletes

```cpp
class MyQuery : public PDBQueryExecutor {
    operator +=;
    void aggregate();
};
```
Can be used in other ways too…

- Large-scale program analysis
- Large-scale combinatorial search
The rest of the talk

- The Pliny user interface
- Pliny in action (1)
- The Pliny language framework
- The Pliny statistical database
- The Pliny reasoning framework
- Pliny in action (2)
The Pliny reasoning framework

Functional synthesis problem

Pliny reasoning framework

Solution to synthesis instance

Find a mathematical function that...

1. ... satisfies a set of logical constraints
2. ... can be expressed in a syntactic space built from corpus components
3. ...is optimal by a set of quantitative criteria

Combinatorial search in space of lambda-terms

+ Automated deduction
The Pliny reasoning framework

Find a mathematical function that satisfies some logical constraints and can be expressed in a syntactic space built from corpus components.

3. ...is optimal by a set of quantitative criteria

• No divide-by-zero errors
• Passes user-supplied tests

• Parsimony
  Structure considered “likely” by database


Refinement-based search

\[
\lambda x. \mathbf{[\ ]} \quad \lambda x. x \quad \ldots \quad \lambda x. \text{map } x (\mathbf{[\ ]}) \quad \lambda x. \text{foldl } x (\mathbf{[\ ]})
\]
Use of database queries

\[ \lambda x. [ ] \quad \lambda x. x \quad \ldots \quad \lambda x. \text{map} x (\ldots) \quad \lambda x. \text{foldl} x (\ldots)[] \]

\[ \lambda x. \text{map} x (\ldots) \quad \lambda x. \text{map} x (\ldots) \]

Statistical framework
Deduction

- Deduce subgoals
- Derive conflicts
- Parallels with SAT solvers
The rest of the talk

• The Pliny user interface
• Pliny in action (1)
• The Pliny language framework
• The Pliny statistical database
• The Pliny reasoning framework
• Pliny in action (2)
Problem #1: Code completion

```c
int binsearch
(int x, int v[], int n)
{
    int low, high, mid;
    low = 0;
    high = ??;
    while(low <= high) {
        ??
    }
    return -1;
}
```
• ~3 million program elements (functions)
• ~10 features

Corpus

Feature extraction

Draft

Completion

main()
-- --
??
-- -- --

Statistical framework

Query Response

Reasoning framework

• Top-k search (KNN) at scale

• Query for 5 similar programs
• Generate candidate completions
• Combinatorial search to instantiate holes
Problem #2: Program repair

```c
int binsearch(int x, int v[], int n) {
    int low, high, mid;

    low = 0;
    high = n - 1;
    while (low <= high) {
        mid = (high - low) / 2;
        if (x <= v[mid]) {
            high = mid - 1;
        } else if (x > v[mid]) {
            low = mid + 1;
        } else {    /* found match */
            return mid;
        }
    }
    return -1;    /* no match */
}
```
Statistical framework

Query generation

main()

Incorrect program

Candidate generation

Type-aware search

Testing

Ranked list of fixes

Query

Similar programs

Similar programs

Candidate substitutions

Bug localization

Template
Problem #3: Learning API specifications

- “Always read only from a file that has been opened”
- “Do not read and write into the same file”

Learn from sequences of calls, and constraints among their arguments, generated from real code
Learning GNU Corpus

Feasible call sequences with ??

Feature Extraction

foo[p1] bar[p2]
-- -- -- -- --
foo[p3] bar[p2]
-- -- -- --
-- foo[p1] --

Feasible call sequences with predicates

main() {
  -- --
  ??
  -- --
  bar()
}

Incomplete new program

Feature Extraction

Statistical models of call sequences with predicates

Markov random fields

Learning

Prediction

Probability distribution for ?? along each path

Aggregate Metrics

?? foo, ...

Final Prediction(s)
The takeaways

Formal Methods + Big Data

= 

• New playground for research
  – New bridge between logic and statistics
  – Many algorithmic and engineering challenges

• Lots of energy and activity
  – Coming to Github in Spring 2016!
Thank you!

Questions?

vijay@rice.edu

http://vijay.rice.edu