Prospector: Navigating the API Jungle with Jungloid Mining and Synthesis

David Mandelin, Lin Xu, Ras Bodik, Doug Kimelman (IBM)

Administrativia: coming up

- Tue next week: transformational synthesis
  - a classic paper (TBD)
  - Ras presents
- Thu: Synthesis of garbage collectors [PLDI'06]
  - guest speaker Eran Yahav
- Tue: synthesis of sparse scientific codes
  - a paper from the Bernoulli project
  - a student presenter?
- Thu: similar topic
  - guest speaker: Prof. Kathy Yelick

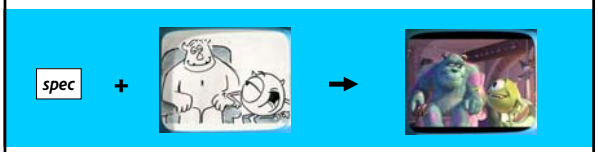
Administrativia

- Paper summaries
  - submit for each paper by email by 7pm the evening before
  - directions now on the web site
- Signing up for papers
  - sign for topics; we'll agree on a paper later
  - email your ranked preferences today

Administrativia

- Challenge problems (this Thursday)
  - 5-to-10-minute presentations, show-and-tell style
  - describe a dream language/tool/system
  - posed problem need not be solvable soon, or even well-stated
  - but should be motivated by a real problem
- Sign up now
  1. Ras
  2. Dave Mandelin
  3. Alan
  4. Amir + Jimmy
  5. Rusty
  6. AJ
  7. Armando

Prospector: fits same "sketching formula"



specification: hard to write, so not provided

sketch

multiple implementations, each different behavior, user chooses one, indirectly providing the spec

Software reuse: the promise

- Software reuse success stories
  - productivity increased up to 900%
  - defect rate improves dramatically once 80% of code reused
- Can we reuse even more? Some key obstacles [Lampson]:
  - only three components easily extensible (OS, DB, web browser)
  - rest is hard to (1) sell, (2) compose, (3) understand
- Recent trends solved some problems
  - ☺ **Sell:** it pays to make them free: J2EE, Eclipse
  - ☺ **Compose:** extensibility via design patterns, …
  - ☹ **Understand:** flexibility demands many fine-grain "LEGO pieces"

Software reuse: the reality

Using Eclipse 2.1, parse a Java file into an AST

```
IFile file = …
ICompilationUnit cu = JavaCore.createCompilationUnitFrom(file);
ASTNode node = AST.parseCompilationUnit(cu, false);
```

Productivity < 1 LOC/hour **Why so low?**

1. follow expected design? two levels of file handlers
2. class member browsers? two unknown classes used
3. grep for ASTNode? method returns subclass: CompilationUnit

7

Our goal: Synthesize desired code

- Programmer expresses intent, system supplies code
- But programmer's intent is often vague
  - "parse my file"
- **Idea:** Let programmer supply partial intent
  - System will synthesize several candidate code snippets
  - Programmer only needs to select desired code
- User's experience is like a search engine
  - (Remember, the system will synthesize, not search)

8

Problem Statement

- **Input from data sources**: knowledge about API
  - API declarations (class declarations, method signatures)
    - Prospector I: basic synthesizer (first part of talk)
  - Sample client code
    - Prospector II: enhanced synthesizer (second part of talk)
- **Input from user**: specification of intent
  - easy for programmer, yet specific enough for synthesizer
- **Output:** API client code
  - synthesized code, ready for insertion into program
  - several candidates, ranked

9

Input specification: *have-want query*

- 1st observation
  - Many reuse problems can be described with a have-one-want-one query *q=(h,w):*

> "What code will transform
> a (single) object of (static) type *h* into
> a (single) object of (static) type *w*?"

- Our parsing example: *q = (IFile, ASTNode)*

```
IFile file = …
ICompilationUnit cu = JavaCore.createCompilationUnitFrom(file);
ASTNode node = AST.parseCompilationUnit(cu, false);
```

10

Output code: *jungloid*

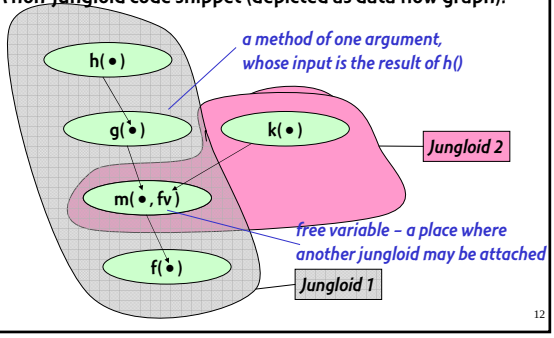
- 2nd observation:
  - most queries can be answered with a jungloid
- jungloid:
  - a unary expression composed of unary expressions:
    - field access
    - call to an instance method with 0 arguments
    - call to a static method or constructor with 1 argument
    - widening conversion (i.e., conversion to supertype)

```
IFile file = …
ICompilationUnit cu = JavaCore.createCompilationUnitFrom(file);
ASTNode node = AST.parseCompilationUnit(cu, false);
```

11

Decomposing complex code into jungloids

A non-jungloid code snippet (depicted as data flow graph):



12

Coverage

An informal experiment:

- using 16 coding headaches, collected by us

- **Can the query express interesting problems?**
  - yes, for 12 out of 16 coding problems

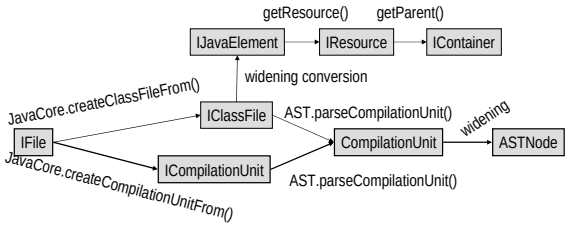
- **Can queries be answered with a jungloid?**
  - yes, all 12 queries answered with jungloids
    - 9 of them need one jungloid
    - 3 of them composed from multiple jungloids

13

Jungloid Synthesis

Synthesis: Type signature graph

Any path from *h* to *w* is a *(h,w)*-jungloid



- **3rd observation:**
  - desired jungloid typically among *k* shortest paths *(k=5)*

15

Integrating synthesis with IDEs

- **How do we present jungloid synthesis to programmers?**
- **Integrate with IDE "code completion"**

```
public void parse(IFile file) {
    file.
}
```

16

Integrating synthesis with IDEs

- **How do we present jungloid synthesis to programmers?**
- **Integrate with IDE "code completion"**

```
IEditorPart editor;

public void parse(IFile file) {
    ASTNode ast =
```

have types

want type

Queries: (IFile, ASTNode)
(IEditorPart, ASTNode)

17

Demo

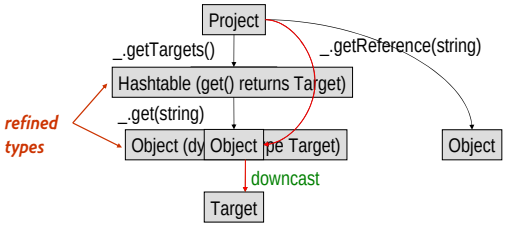
Prospector II

Prospector I + jungloid mining

Jungloids with downcasts

```
Project project = ...;
Hashtable targets = project.getTargets();
Target t = (Target) targets.get(name);
```

Example from Ant (an extensible Java make)



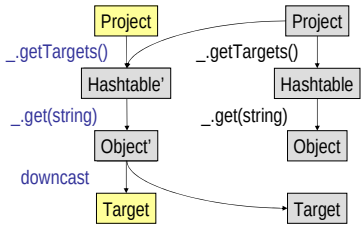
20

Refining the signature graph

- How do we find refined types for the graph?
- Potential solutions
  - example: parametric type inference
- Our solution
  - mine a corpus of API uses for legal downcasts

21

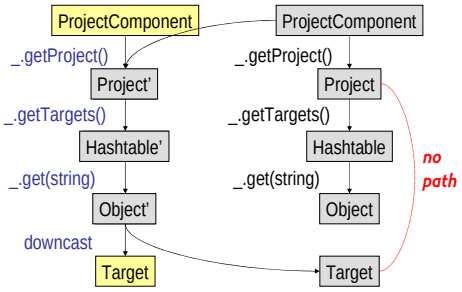
Mining jungloids with downcasts (example)



Example Jungloid Graph Signature Graph

22

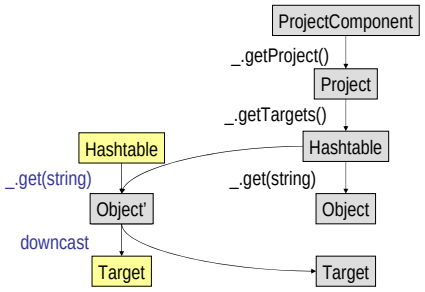
Long examples miss jungloids



Example too long → fail to synthesize desired jungloid

23

Short examples make bad jungloids

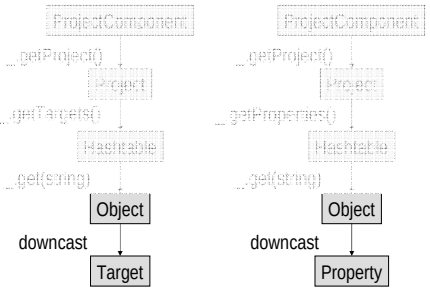


Example too short → synthesize jungloids that throw exceptions

24

Finding the right example "context"

Rule: find min context that uniquely determines downcast



25

Mining jungloids with downcasts

- Algorithm overview
  - Select a corpus of API client code
  - Extract jungloids containing downcasts
  - Find best context for extracted jungloids
  - Refine signature graph with extracted jungloids
- Ideally, only correct jungloids are synthesized
  - correct = it is possible to write a client code in which the jungloid's downcast succeeds
- In the limit, we meet the ideal
  - limit = infinitely large, bug-free corpus
  - bug-free = no ClassCastException for at least one input

26

Evaluation

Experiment 1 (ranking test)

- hypothesis:
  - to find desired code, user needs to examine only top 5 candidate jungloids
- experimental design:
  - used 20 real-world coding tasks
  - collected from FAQs, newsgroups, practice, emails

Rank	% at rank	Cumulative %
1	55	55
2	10	65
3	15	80
4	10	90
Not found	10	100

28

Experiment 2 (user study)

hypothesis:

- Prospector-equipped programmers solve API problems
  - in less time
  - with more concise code

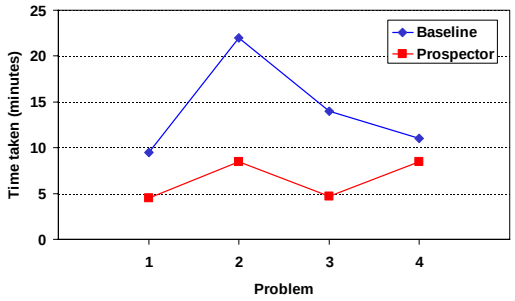
methodology:

- 4 problems, each user did 2 with Prospector and 2 without
- sample problem:

"We will provide you a string containing a URL that points to a sound file. Write code to play the sound."

29

Experiment 2 (user study). Results.



30

Experiment 2 (user study). Results.

- Prospector helps enable reuse
  - non-Prospector users sometimes reimplemented
- Prospector helps avoid mistakes
  - Reimplemented class does not satisfy its interface contract
  - Some non-prospector users used a bad cast
  - Prospector's mined solution was correct

31

Summary and Related Work

- Synthesize API client code
  - Constraint-based synthesis: Steffen, et al. (1994)
  - Type-based synthesis (Demeter, Persephone): Lieberherr, et al.
- Have-want queries describe reuse problems
  - Signature matching queries: Zaremski and Wing; Fischer and Ye
  - Context queries (Hipikat, Strathcona): Murphy et al., Holmes (2005)
  - Type-based queries
- Mining jungloids for downcasts
  - extract and clean up examples to use in synthesis

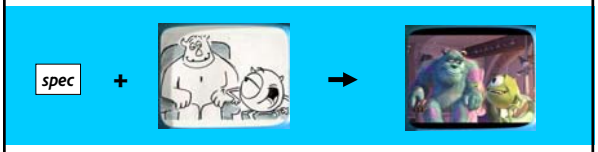
32

Future Work

- Synthesis for new kinds of reuse problems
  - e.g. "I called method foo(), observed no output"
  - synthesize code to make foo() behave correctly
- Mining to solve more analysis problems
  - e.g., infer String "subtypes" (URL, date, …)

33

Solving the "sketching equation"



1. Implementation Gap (StreamBit, SKETCH)
   - implementation is an "optimization" of the spec
2. Spec selection (Prospector)
   - easier to select the spec than to write it

34