Program Sliding
A Novel Approach for the Automation of Slicing-Based Refactoring
June 2007

Ran Ettinger
In Haifa University’s
MIS Department Seminar
Agenda

- When Refactoring Met Slicing...
  - Motivating example and some definitions
- Original Research Question (2001)
  - “How can program slicing and related analyses assist in building automatic tools for refactoring?”
- Initial Results (2003)
  - “Untangling: a slice extraction refactoring” [AOSD04]
  - The need for more...
- Final Results (2006)
  - Sliding on top of a formal theory for slicing-based refactoring...
- Some Further Challenges
On Sliding and Code-Motion Transformations

```java
int sum=0;
int prod=1;
for (int i=0; i<a.length; i++) {
    sum += a[i];
    prod *= a[i];
}
System.out.println("The sum is " + sum);
System.out.println("The product is " + prod);
```
On Sliding and Code-Motion Transformations

```java
int sum=0; for (int i=0; i<a.length; i++) {
    sum += a[i];
}
```

```java
int computeSum(int[] a) {
    int sum=0;
    for (int i=0; i<a.length; i++) {
        sum += a[i];
    }
    return sum;
}
```

```java
int sum=computeSum(a);
int prod=1;
for (int i=0; i<a.length; i++) {
    prod *= a[i];
}
System.out.println("The sum is " + sum);
System.out.println("The product is " + prod);
```
int sum=0;
for (int i=0; i<a.length; i++) {
    sum += a[i];
}
return sum;

int prod=1;
for (int i=0; i<a.length; i++) {
    prod *= a[i];
}
System.out.println("The sum is " + computeSum(a));
System.out.println("The product is " + prod);
When Refactoring Met Slicing…

- **Refactoring** is a discipline of change
  - Of the internal structure of a software system
  - Through small and well-defined source-code transformations
  - That preserve the (observable) behavior of the original system (program)

- **Slicing** is the study of meaningful subprograms
  - Given a program and a variable (at a point) of interest, a slice of the program on that variable is a subprogram that preserves the original behavior, with respect to that variable

- **Sliding** is a mini refactoring of slice extraction
  - A kind of code-motion transformation, like in compilers
  - Trading code reusability and readability with runtime efficiency
  - Given a program and a variable of interest, sliding extracts the variable’s slice from its complement; but what’s in the complement?
    - The union of all remaining slices?
    - or something smaller: reusing the extracted variable’s final value
Original Research Question (2001)

- **Initial observation:**
  - “Programmers use slices when refactoring”
    - Paraphrasing Weiser [CACM82]

- **Context**
  - Intercomp’s Cobol-to-Java
  - OOPSLA’00
  - Fowler’s refactoring book [Fow]
    - The video-store example: from procedural design to objects

- **The research question:**
  - “How can program slicing and related analyses assist in building automatic tools for refactoring?”
Initial Results (2003)

- A definition of an “Extract Slice” refactoring
  - Generalizing “Extract Method” and “Replace Temp with Query”

- Nate: an Eclipse plugin
  - Supporting slice extraction on a small subset of Java
  - Supported by an Eclipse Innovation Award from IBM
  - With Mathieu Verbaere: the slicer and MSc thesis

- An investigation of untangling refactorings to introduce aspects

- “Untangling: a slice extraction refactoring” [AOSD04, with Mathieu]
Initial Results (2003) - cont.: The Need for More

- **State-of-the-art slice-extraction solutions**
  - Tucking by Lakhotia and Deprez [IST98]
  - Procedure extraction by Komondoor and Horwitz [POPL00, IWPC03]
  - Block-based slicing by Maruyama [SSR01]

- **Identified limitations (Nate and the related work)**
  - Strong preconditions
    - E.g. no duplication of assignments
    - Leading to low applicability
  - Informal (unconvincing) correctness proof

- **Observation/Realization:**
  - Weakening of preconditions (i.e. improving applicability) will require a careful (preferably formal) investigation
Final Results (2006)

- A formal theory of slicing-based refactoring
  - Based on “predicate calculus and program semantics” [Dijkstra and Scholten] and the refinement calculus [Back, Morgan]
  - Supporting a simple imperative programming language

- A designated slice-refinement proof method

- A novel program decomposition into small entities called “slides”

- A highly applicable solution to the slice-extraction refactoring problem, called “sliding”

- A concept of smaller slices called “co-slicing”

- A provably correct slicing algorithm (why is it important?)
Some Further Challenges

- Implementing the theoretical framework and algorithms
- Collecting empirical results
- Extending to “real” languages
- Apply sliding to more refactorings (e.g. “Separate Query from Modifier” [Fow], arbitrary method extraction)
- Apply the sliding-related refactorings in bigger reengineering challenges (e.g. Convert Procedural Design to Objects [Fow], componentization, conversion to SOA)
- Sliding beyond refactoring (e.g. in optimizing compilers, code obfuscation)
Thanks!

- **Acknowledgements**
  - My DPhil [Ett] supervisor Oege de Moor
    - And Mike Spivey, supervising during Oege’s sabbatical (2003)
  - Past and present members of the Programming Tools Group at Oxford, in particular Iván Sanabria, Yorck Hünke, Stephen Drape, Mathieu Verbaere, Damien Sereni
  - My thesis examiners Mark Harman and Jeremy Gibbons
References

- [CACM82] Programmers use slices when debugging, M. Weiser, 1982
- [Back] A calculus of refinements for program derivations, 1988
- [Dijkstra and Scholten] Predicate calculus and program semantics, 1990
- [Morgan] Programming from specifications (2nd ed.), 1994
- [IST98] Restructuring programs by tucking statements into functions, A. Lakhotia and J.-C. Deprez, 1998
- [POPL00] Semantics-preserving procedure extraction, R. Komondoor and S. Horwitz, 2000
- [SSR01] Automated method-extraction refactoring by using block-based slicing, K. Maruyama, 2001
- [IWPC03] Effective automatic procedure extraction, R. Komondoor and S. Horwitz, 2003
- [AOSD04] Untangling: a slice extraction refactoring, R. Ettinger and M. Verbaere, 2004
  - Submitted version http://progtools.comlab.ox.ac.uk/members/rani/sliding thesis esub101006.pdf
Backup
Example: Fowler’s video-store

Rental record for Bilbo Baggins
   The Godfather   5.0
   Amelie From Montmartre   2.0
   La Vita è Bella   5.0
   Lord of the Rings   1.0
   The Ring   6.0
Amount owed is 19.0
You earned 6 frequent renter points
public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Iterator rentals = _rentals.iterator();
    String result = "Rental record for " +
    getName() + "\n";
    while (rentals.hasNext()) {
        Rental each = (Rental) rentals.next();
        // show figures for this rental
        result += "\t" + each.getMovie().getTitle();
        result += "\t" + each.getCharge() + "\n";
        frequentRenterPoints += each.getFRP();
        totalAmount += each.getCharge();
    }
    //add footer lines
    result += "Amount owed is " +
    totalAmount + "\n";
    result += "You earned " +
    frequentRenterPoints +
    " frequent renter points";
    return result;
}
public String statement() {

    Iterator rentals = _rentals.iterator();
    String result = "Rental record for " +
        getName() + "\n";
    while (rentals.hasNext()) {
        Rental each = (Rental) rentals.next();
        // show figures for this rental
        result += "\t" + each.getMovie().getTitle();
        result += "\t" + each.getCharge() + "\n";
    }
    // add footer lines
    result += "Amount owed is " +
        getTotalCharge() + "\n";
    result += "You earned " +
        getTotalFRP() +
        " frequent renter points";
    return result;
}
```java
private double getTotalCharge() {
    double result = 0;
    Iterator rentals = _rentals.iterator();
    while (rentals.hasNext()) {
        Rental each = (Rental) rentals.next();
        result += each.getCharge();
    }
    return result;
}
```
public int getTotalFRP() {
    int result = 0;
    Iterator rentals = _rentals.iterator();
    while (rentals.hasNext()) {
        Rental each = (Rental) rentals.next();
        result += each.getFRP();
    }
    return result;
}