

# Program Slicing and Sliding for Refactoring

Mirko Stocker

January 8, 2009

- 1 Introduction
  - Problem
  - Solution
- 2 Program Sliding
  - What is Sliding?
  - Sliding Demonstration
  - Conclusion and Limitations
- 3 Enhancing Refactorings
  - Split Temporary Variable
  - Extract Method
  - Extract Subclass
  - Extract Superclass
- 4 Conclusion

# What's the Problem With Current Tool Support?

- Refactoring tools already quite powerful ...

# What's the Problem With Current Tool Support?

- Refactoring tools already quite powerful ...
- but there are still many limitations ...

# What's the Problem With Current Tool Support?

- Refactoring tools already quite powerful ...
- but there are still many limitations ...
- and many improvements imaginable.

# What's the Problem With Current Tool Support?

- Refactoring tools already quite powerful ...
- but there are still many limitations ...
- and many improvements imaginable.

## Extract Method

Let's take a look at an example: *Extract Method*.

# What's the Problem With Current Tool Support?

## Extract Method Limitations

```
sum , prod = 0, 1
```

```
for i in 1..10
```

```
  sum += i
```

```
  prod *= i
```

```
end
```

```
print sum, prod
```

# What's the Problem With Current Tool Support?

## Extract Method Limitations

```
sum, prod = 0, 1
```

```
for i in 1..10
```

```
  sum += i
```

```
  prod *= i
```

```
end
```

```
print sum, prod
```

Can we extract the calculation of `sum`?

# What's the Problem With Current Tool Support?

## Extract Method Limitations

```
sum , prod = 0, 1
```

```
for i in 1..10
```

```
  sum += i
```

```
  prod *= i
```

```
end
```

```
print sum, prod
```

Can we extract the calculation of `sum`?

Of course!

# What's the Problem With Current Tool Support?

## Extract Method Limitations

```
sum , prod = 0, 1
```

```
for i in 1..10
```

```
  sum += i
```

```
  prod *= i
```

```
end
```

```
print sum, prod
```

Can we extract the calculation of `sum`?

Of course! But can our refactoring tool do it for us?

# What's the Problem With Current Tool Support?

## Extract Method Limitations

```
sum , prod = 0, 1
```

```
for i in 1..10
```

```
  sum += i
```

```
  prod *= i
```

```
end
```

```
print sum, prod
```

Can we extract the calculation of `sum`?

Of course! But can our refactoring tool do it for us?

Probably not

# What's the Problem With Current Tool Support?

## Extract Method Limitations

```
sum, prod = 0, 1
```

```
for i in 1..10
```

```
  sum += i
```

```
  prod *= i
```

```
end
```

```
print sum, prod
```

Can we extract the calculation of `sum`?

Of course! But can our refactoring tool do it for us?

Probably not (yet).

# Slicing Can Help Building More Powerful Tools

Slicing can be used to untangle computations based on the dependencies between statements and the variables they manipulate.

# Ran Ettinger's Thesis About Sliding

My seminar paper is based on **Ran Ettinger's** Ph. D. thesis about "Refactoring via Program Slicing and Sliding".

It is based on the metaphors of program *slides* and the sliding operation.



# Ran Ettinger's Thesis About Sliding

Developed a provable correct algorithm for program transformations.

Based on Dijkstra's Guarded Command Language, which in turn uses Hoare logic.

# Ran Ettinger's Thesis About Sliding

Developed a provable correct algorithm for program transformations.

Based on Dijkstra's Guarded Command Language, which in turn uses Hoare logic.

Not that important for us right now.

# Sliding Demonstration

## ① Initial Code

# Sliding Demonstration

- 1 Initial Code
- 2 Statement Duplication

# Sliding Demonstration

- 1 Initial Code
- 2 Statement Duplication
- 3 Final-Use Substitution

# Sliding Demonstration

- 1 Initial Code
- 2 Statement Duplication
- 3 Final-Use Substitution
- 4 Eliminating Dead Code

## Liveness Analysis

A variable is live at a point if it is used until it receives a new value or the scope ends.

# Sliding Demonstration

- 1 Initial Code
- 2 Statement Duplication
- 3 Final-Use Substitution
- 4 Eliminating Dead Code

## Liveness Analysis

A variable is live at a point if it is used until it receives a new value or the scope ends.

- 5 Cleanup

# Sliding Conclusion and Limitations

Nice to have a provable correct algorithm,

# Sliding Conclusion

## and Limitations

Nice to have a provable correct algorithm, but unfortunately just for a toy language with severe restrictions:

# Sliding Conclusion

## and Limitations

Nice to have a provable correct algorithm, but unfortunately just for a toy language with severe restrictions:

- variables need to be cloneable,

# Sliding Conclusion

## and Limitations

Nice to have a provable correct algorithm, but unfortunately just for a toy language with severe restrictions:

- variables need to be cloneable,
- no side effects in duplicated code are allowed.

# Sliding Conclusion

## and Limitations

Nice to have a provable correct algorithm, but unfortunately just for a toy language with severe restrictions:

- variables need to be cloneable,
- no side effects in duplicated code are allowed.

Ettinger dropped the framework while “moving on to the real world”.

# Enhancing Existing Refactorings With Slicing

Suitable Candidates From Fowler's Catalogue

## Extracting Refactorings

## Local Scope Refactorings

# Enhancing Existing Refactorings With Slicing

Suitable Candidates From Fowler's Catalogue

## Extracting Refactorings

- Extract Method
- Extract Class / Package
- Extract Sub- and Superclass
- Pull Up Method
- Form Template Method

## Local Scope Refactorings

# Enhancing Existing Refactorings With Slicing

Suitable Candidates From Fowler's Catalogue

## Extracting Refactorings

- Extract Method
- Extract Class / Package
- Extract Sub- and Superclass
- Pull Up Method
- Form Template Method

## Local Scope Refactorings

- Split Loop
- Split Temporary Variable
- Replace Assignment with Initialization
- Reduce Scope of Variable

# Enhancing Existing Refactorings With Slicing

Suitable Candidates From Fowler's Catalogue

## Extracting Refactorings

- Extract Method
- Extract Class / Package
- Extract Sub- and Superclass
- Pull Up Method
- Form Template Method

## Local Scope Refactorings

- **Split Loop**
- Split Temporary Variable
- Replace Assignment with Initialization
- Reduce Scope of Variable

# Enhancing Existing Refactorings With Slicing

Suitable Candidates From Fowler's Catalogue

## Extracting Refactorings

- Extract Method
- Extract Sub- and Superclass

## Local Scope Refactorings

- Split Temporary Variable

# Split Temporary Variable

A *temporary variable* holds several unrelated values during its lifetime:

```
temp = 2 * (_height + _width)
```

```
print temp
```

```
temp = _height * _width
```

```
print temp
```

# Split Temporary Variable

A *temporary variable* holds several unrelated values during its lifetime:

```
temp = 2 * (_height + _width)
```

```
print temp
```

```
temp = _height * _width
```

```
print temp
```

Do we need slicing?

# Split Temporary Variable

A *temporary variable* holds several unrelated values during its lifetime:

```
temp = 2 * (_height + _width)
```

```
print temp
```

```
temp = _height * _width
```

```
print temp
```

Do we need slicing?

Not really, converting to the *static single assignment* form and renaming is enough.

# Split Temporary Variable

A *temporary variable* holds several unrelated values during its lifetime:

```
temp1 = 2 * (_height + _width)
```

```
print temp1
```

```
temp2 = _height * _width
```

```
print temp2
```

Do we need slicing?

Not really, converting to the *static single assignment* form and renaming is enough.

# Split Temporary Variable

A *temporary variable* holds several unrelated values during its lifetime:

```
perimeter = 2 * (_height + _width)
```

```
print perimeter
```

```
area = _height * _width
```

```
print area
```

Do we need slicing?

Not really, converting to the *static single assignment* form and renaming is enough.

# Extract Method

- Slicing helps us to distinguish interleaved but independent computations.
- Once we have separate computations, we can extract them.

# Extract Method

- Slicing helps us to distinguish interleaved but independent computations.
- Once we have separate computations, we can extract them.

## Improvements

- Suggest to extend the user's selection.

# Extract Method

- Slicing helps us to distinguish interleaved but independent computations.
- Once we have separate computations, we can extract them.

## Improvements

- Suggest to extend the user's selection.
- Analyze code and propose refactoring.

# Extract Subclass

## Problem

A class has features that are used only in some instances.

## Solution (Fowler)

Create a subclass for that subset of features.

# Extract Subclass

## Problem

A class has features that are used only in some instances.

## Solution (Fowler)

Create a subclass for that subset of features.

- How do we determine that subset of features?

# Extract Subclass

## Problem

A class has features that are used only in some instances.

## Solution (Fowler)

Create a subclass for that subset of features.

- How do we determine that subset of features?
- By slicing on the code from different clients.

# Extract Superclass

## Problem

You have two classes with similar features.

## Solution (Fowler)

Create a superclass and move the common features to the superclass.

# Extract Superclass

## Problem

You have two classes with similar features.

## Solution (Fowler)

Create a superclass and move the **common features** to the superclass.

- With slicing we can determine methods that are not equal but share common code.

# Extract Superclass

## Problem

You have two classes with similar features.

## Solution (Fowler)

Create a superclass and move the **common features** to the superclass.

- With slicing we can determine methods that are not equal but share common code.
- Similar to *Form Template Method* and *Pull up Method*.

# Conclusion

**Yes**, slicing can help us to build better refactoring tools!