Program Slicing and Sliding for Refactoring

Mirko Stocker

January 8, 2009
Introduction
- Problem
- Solution

Program Sliding
- What is Sliding?
- Sliding Demonstration
- Conclusion and Limitations

Enhancing Refactorings
- Split Temporary Variable
- Extract Method
- Extract Subclass
- Extract Superclass

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 imagineable.
What’s the Problem With Current Tool Support?

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

Extract Method

Let’s take a look at an example: *Extract Method*. 
What’s the Problem With Current Tool Support?

Extract Method Limitations

```python
sum, prod = 0, 1
for i in range(1, 11):
    sum += i
    prod *= i
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

```python
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

```
s, p = 0, 1
for i in 1..10
    s += i
    p *= i
end
print s, p
```

Can we extract the calculation of \texttt{s}um?  
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

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

- variables need to be cloneable,
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.
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

<table>
<thead>
<tr>
<th>Extracting Refactorings</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Extract Method</td>
</tr>
<tr>
<td>- Extract Class / Package</td>
</tr>
<tr>
<td>- Extract Sub- and Superclass</td>
</tr>
<tr>
<td>- Pull Up Method</td>
</tr>
<tr>
<td>- Form Template Method</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local Scope Refactorings</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>Split Loop</strong></td>
</tr>
<tr>
<td>- Split Temporary Variable</td>
</tr>
<tr>
<td>- Replace Assignment with Initialization</td>
</tr>
<tr>
<td>- Reduce Scope of Variable</td>
</tr>
</tbody>
</table>
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
A temporary variable holds several unrelated values during its lifetime:

\[
\text{temp} = 2 \ast (_\text{height} + _\text{width})
\]

print \text{ temp}

\[
\text{temp} = _\text{height} \ast _\text{width}
\]

print \text{ temp}
Split Temporary Variable

A temporary variable holds several unrelated values during its lifetime:

\[
\begin{align*}
\text{temp} &= 2 \times (\_\text{height} + \_\text{width}) \\
\text{print} & \quad \text{temp} \\
\text{temp} &= \_\text{height} \times \_\text{width} \\
\text{print} & \quad \text{temp}
\end{align*}
\]

Do we need slicing?
A temporary variable holds several unrelated values during its lifetime:

```python
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.
A temporary variable holds several unrelated values during its lifetime:

\[
\text{temp}_1 = 2 \times (\_\text{height} + \_\text{width})
\]
\[
\text{print temp}_1
\]
\[
\text{temp}_2 = \_\text{height} \times \_\text{width}
\]
\[
\text{print temp}_2
\]

Do we need slicing?

Not really, converting to the static single assignment form and renaming is enough.
A temporary variable holds several unrelated values during its lifetime:

```python
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.
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?
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.
Yes, slicing can help us to build better refactoring tools!