

Formal Program Re-design
or (more specifically)
Automating
“Replace Temp with Query”

Ran Ettinger
Programming Tools Group
Computing Laboratory
University of Oxford

VASTT/ASTReNet Slicing Day Workshop,
November 8th, 2004

Slice extraction refactoring

- **Replace Temp with Query** (Fowler, 2000)
 - “You are using a temporary variable to hold the result of an expression.”
 - “Extract the expression into a method. Replace all references to the temp with the expression. The new method can then be used in other methods.”
- **Tuck transformation** (Lakhotia and Deprez, 1999)
Tuck = Wedge + Split + Fold
- **Replace Temp with Query** = Tuck + Inline Temp

Tuck: selecting *seed* and *scope*

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
        totalSalary += people[i].salary;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

S

T

t = {totalSalary}

Wedge

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
        totalSalary += people[i].salary;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

S

T

slice(S, t)

t = {totalSalary}

Split: identify complement seeds

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
        totalSalary += people[i].salary;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

S

T

$S \setminus \text{slice}(S, t)$

Split(2): slice from the complement seeds

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
        totalSalary += people[i].salary;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

S

T

$\text{slice}(S, S \setminus \text{slice}(S, \{totalSalary\}))$

Split(3)

```
String computeReport() {  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalSalary += people[i].salary;  
    }  
    String result = "";  
    double totalAge = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

Fold

```
String computeReport() {  
    double totalSalary = computeTotalSalary();  
    String result = "";  
    double totalAge = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}  
  
double computeTotalSalary () {  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalSalary += people[i].salary;  
    }  
    return totalSalary;  
}
```

Inline Temp

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + computeTotalSalary();  
    return result;  
}  
  
double computeTotalSalary () {  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalSalary += people[i].salary;  
    }  
    return totalSalary;  
}
```

Problems with Tuck

- Limited proof framework
 - Based on program dependence graphs and HPR operational semantics
- Language restrictions:
 - No global variables
 - No procedures
- Too conservative
 - Rejects a transformation any time an out variable (i.e. a variable that is live at the end) is modified both in the slice and the complement
- Complement may be too large
 - In case of dead code

Our approach

- A framework for proving correctness of slicing-based transformations
- Syntax
 - Dijkstra's guarded commands (mainly focusing on a deterministic subset)
 - Extensions by Morgan: procedures and parameter passing
- Semantics
 - Weakest preconditions

Identifying the complement

- Using decomposition slices
 - (Gallagher and Lyle, 1991)
 - Instead of slicing from the end + any output statement, we slice from the end only
 - We treat output statements as modifying a designated *out* variable
 - $\text{slice}(S, v)$ where S is a statement and v is a set of variables is a subset of S computing v
- Slice for the complement variables:
 - $\text{co-slice}(S, v) = \text{slice}(S, \text{co-}v)$ where $\text{co-}v = \text{def}(S) \setminus v$

Towards replace temp with query

$$[[\text{var } t \bullet S ; T]] = \text{co-slice}(S, t); [[\text{var } t \bullet \text{slice}(S, t) ; T]]$$

Provided $\text{co-t} \cap \text{def}(\text{slice}(S, \text{co-t})) \cap \text{in}(\text{co-slice}(S, \text{co-t})) = \emptyset$

and $t \cap \text{free}(\text{co-slice}(S, t)) = \emptyset$

Where $\text{co-t} = \text{def}(S) \setminus t$.

Proof:

$$\begin{aligned} & [[\text{var } t \bullet S ; T]] \\ = & \{ \text{(simplified) extract slice with } v := \text{co-t} : \text{co-t} \cap \text{def}(\text{slice}(S, \text{co-t})) \\ & \quad \cap \text{in}(\text{co-slice}(S, \text{co-t})) = \emptyset \text{ since } \text{slice}(S, \text{co-t}) = \text{co-slice}(S, t) \} \\ & [[\text{var } t \bullet \text{slice}(S, \text{co-t}) ; \text{co-slice}(S, \text{co-t}) ; T]] \\ = & \{ \text{def. of co-slice, twice} \} \\ & [[\text{var } t \bullet \text{co-slice}(S, t); \text{slice}(S, t) ; T]] \\ = & \{ \text{reduce scope of variable declaration: } t \cap \text{free}(\text{co-slice}(S, t)) = \emptyset \} \\ & \text{co-slice}(S, t); [[\text{var } t \bullet \text{slice}(S, t) ; T]] \quad \square \end{aligned}$$

The **seed**'s complement

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
        totalSalary += people[i].salary;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

S {

T {

$\text{co-t} = \text{def}(S) \setminus t = \{avgAge, totalAge, result\}$

Slicing for the complement variables

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
        totalSalary += people[i].salary;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

S {

T {

co-slice(*S*, *t*) = slice(*S*, *co-t*)

Problem

- The produced complement may be too large
 - If the slice of a variable in the complement is strongly-dependent on the seed's slice
 - slice($S, \{x\}$) is said to be strongly-dependent on slice($S, \{y\}$) if the former is included in the latter
 - (Gallagher and Lyle, 1991)

Too large complement

```
String computeReport() {  
    String result = "";  
    double totalAge = 0;  
    double totalSalary = 0;  
    for (int i=0; i < people.length; i++) {  
        totalAge += people[i].age;  
        totalSalary += people[i].salary;  
    }  
    double avgAge = (totalAge / people.length);  
    result += "average age:" + avgAge;  
    result += "total salary:" + totalSalary;  
    return result;  
}
```

S' {
T' {

slice(S', {totalAge}) is strongly-dependent on
slice(S', {avgAge})

Solution

- Instead of

$$\text{co-slice}(S, v) = \text{slice}(S, \text{def}(S) \setminus v)$$

- have

$$\text{co-slice}(S, v) = \text{slice}(S, \text{maximal}(S) \setminus v)$$

- where maximal(S) is the set of variables whose decomposition slice of S is not included in any other (single variable's) decomposition slice
 - Again, taken with a minor modification from Gallagher and Lyle

Thank you!