Slicing the SCAM Mug:

A Case Study in Semantic Slicing

Dr Martin Ward

Software Technology Research Lab
De Montfort University

and

Software Migrations Ltd
St Albans, Herts

Martin.Ward@durham.ac.uk
http://www.dur.ac.uk/martin.ward/
http://www.cse.dmu.ac.uk/~mward/

Program Slicing

A slice **S** of program **P** is a *reduced*, *executable program* obtained from **P** by removing statements such that **S** replicates part of the behaviour of **P**. [Weiser 1984]

Example Program:

$x := y + 1;$
$y := y + 4;$
$x := x + z$

Slice on the final value of x:

$x := y + 1;$
$x := x + z$

These two programs are *not* semantically equivalent.

Program Slicing

But if we remove y from the final state space, then the modified programs *are* equivalent:

$x := y + 1;$
$y := y + 4;$
$x := x + z;$
remove(y)

and:

$x := y + 1;$
$x := x + z;$
remove(y)

Slicing in the Middle of a Program

A "middle slice" can be converted to an "end slice" by adding a variable:

$i := 0;\ s := 0;$
while $i < n$ **do**
    slice := slice $+\!\!+ \langle i \rangle$;
    $s := s + i$;
    $i := i + 1$ **od**;
$i := 0;$
remove(i, s, n)

is equivalent to:

$i := 0;$
while $i < n$ **do**
    slice := slice $+\!\!+ \langle i \rangle$;
    $i := i + 1$ **od**;
remove(i, s, n)

The Reduction Relation

A syntactic relation between programs which "models" the effect of deleting statements.

For any program:

$$\mathbf{S} \sqsubseteq \mathbf{S}$$

For any proper sequence:

$$\textbf{skip} \sqsubseteq \mathbf{S}$$

For any program where $n > 0$ is the largest integer such that there is an **exit**$(n+k)$ within $k \geqslant 0$ nested **do** ... **od** loops in **S**:

$$\textbf{exit}(n) \sqsubseteq \mathbf{S}$$

If $\mathbf{S}'_1 \sqsubseteq \mathbf{S}_1$ and $\mathbf{S}'_2 \sqsubseteq \mathbf{S}_2$ then:

$$\textbf{if B then } \mathbf{S}'_1 \textbf{ else } \mathbf{S}'_2 \textbf{ fi} \sqsubseteq \textbf{if B then } \mathbf{S}_1 \textbf{ else } \mathbf{S}_2 \textbf{ fi}$$

If $\mathbf{S}' \sqsubseteq \mathbf{S}$ then:

$$\textbf{while B do } \mathbf{S}' \textbf{ od} \sqsubseteq \textbf{while B do S od}$$

$$\textbf{var } \langle v := e \rangle : \mathbf{S}' \textbf{ end} \sqsubseteq \textbf{var } \langle v := e \rangle : \mathbf{S} \textbf{ end}$$

$$\textbf{var } \langle v := \bot \rangle : \mathbf{S}' \textbf{ end} \sqsubseteq \textbf{var } \langle v := e \rangle : \mathbf{S} \textbf{ end}$$

Slicing Non-Terminating or Non-Deterministic Programs

Slicing on the final value of x:

$x := 1;$
while $n > 1$ **do**
  if even?(n) **then** $n := n/2$
    else $n := 3 * n + 1$ **fi od**;
if true $\rightarrow$ $x := 1$
$\square$ **true** $\rightarrow$ $x := 2$ **fi**

Questions:

1. Can we delete the **while** loop?

2. Can we delete the final **if** statement?

Slicing Non-Terminating or Non-Deterministic Programs

A slice **S** of program **P** is a *reduced*, *executable program* obtained from **P** by removing statements such that **S** replicates part of the behaviour of **P**. [Weiser 1984]

Claim:

- We *don't* want to replicate non-termination
- We *do* want to replicate nondeterminism

Semi-Refinement

A *semi-refinement* of $\mathbf{S}$ is any program $\mathbf{S}'$ such that

$$\Delta \vdash \mathbf{S} \;\approx\; \{\mathrm{WP}(\mathbf{S}, \mathbf{true})\};\ \mathbf{S}'$$

The semi-refinement relationship is denoted $\Delta \vdash \mathbf{S} \preccurlyeq \mathbf{S}'$.

Semi-refinement:

- Allows refinement of a non-terminating program
- Ensures semantic equivalence for a terminating program

So semi-refinement is precisely the semantic relationship we need for slicing.

Syntactic Slice

A *Syntactic Slice* of **S** on X is any program **S**′ such that **S**′ $\sqsubseteq$ **S** and

$$\Delta \vdash \mathbf{S};\ \mathbf{remove}(W \setminus X) \preccurlyeq \mathbf{S}';\ \mathbf{remove}(W \setminus X)$$

where W is the final state space for **S** and **S**′.

Compare with Weiser's definition:

- **S**′ is an *executable program* (by definition)
- **S**′ $\sqsubseteq$ **S** means that **S**′ is a *reduction* of **S**
- **S**′ preserves part of the behaviour of **S**

Semantic Slice

A *semantic slice* of **S** on X is any program $\mathbf{S}'$ such that:

$$\Delta \vdash \mathbf{S};\ \mathbf{remove}(W \setminus X) \preccurlyeq \mathbf{S}';\ \mathbf{remove}(W \setminus X)$$

We have simply deleted the syntactic constraint and left in the semantic constraint.

The SCAM Mug

```
while (p(i))
{       if (q(c))
                { x := f();
                  c := g(); }
        i := h(i)
}
```

The problem is to determine which lines do not affect the value of `x`.

WSL translation:

while $p?(i)$ **do**
  if $q?(c)$
    then $x := f;$
      $c := g$ **fi**;
  $i := h(i)$ **od**

Transformation and Slicing

Unroll the first iteration of the loop:

```
if p?(i)
  then if q?(c)
         then x := f; c := g fi;
       i := h(i);
       while p?(i) do
         if q?(c)
           then x := f; c := g fi;
         i := h(i) od fi
```

Expand the **if** $q?(c)\ldots$ statement forwards over the next two statements:

```
if p?(i)
  then if q?(c)
         then x := f; c := g;
              i := h(i);
              while p?(i) do
                if q?(c)
                  then x := f; c := g fi;
                i := h(i) od
          else i := h(i);
              while p?(i) do
                if q?(c)
                  then x := f; c := g fi;
                i := h(i) od fi fi
```

Transformation and Slicing

Simplify:

```
if p?(i)
   then if q?(c)
           then x := f; c := g;
                i := h(i);
                while p?(i) do
                   if q?(c)
                      then x := f; c := g fi;
                   i := h(i) od
            else i := h(i);
                while p?(i) do
                   i := h(i) od fi fi
```

Syntactic Slice on the final value of x

```
if p?(i)
   then if q?(c)
           then x := f;
                c := g;
                i := h(i);
                while p?(i) do
                   if q?(c)
                      then x := f; c := g fi;
                   i := h(i) od fi fi
```

Transformation and Slicing

Constant Propagation:

if $p?(i)$
  then if $q?(c)$
    then $x := f;$
      $c := g;$
      $i := h(i);$
      while $p?(i)$ **do** $i := h(i)$ **od fi fi**

Syntactic Slice:

if $p?(i)$ **then if** $q?(c)$ **then** $x := f$ **fi fi**

Align Nested Statements:

if $p?(i) \wedge q?(c)$ **then** $x := f$ **fi**

This is a *minimal semantic slice*

A Minimal Syntactic Slice

Unroll and Expand the original program:

if $p?(i)$
  then if $q?(c)$
    then $x := f;\ c := g;$
      $i := h(i);$
      while $p?(i)$ **do**
        if $q?(c)$
          then $x := f;\ c := g$ **fi**;
        $i := h(i)$ **od**
    else $i := h(i);$
      while $p?(i)$ **do**
        if $q?(c)$
          then $x := f;\ c := g$ **fi**;
        $i := h(i)$ **od fi fi**

A Minimal Syntactic Slice

Delete the assignment $c := g$ and apply Constant Propagation:

if $p?(i)$
  then if $q?(c)$
    then $x := f;$
      $i := h(i);$
      while $p?(i)$ **do**
        if $q?(g)$
          then $x := f$ **fi**;
        $i := h(i)$ **od**
    else $i := h(i);$
      while $p?(i)$ **do**
        if $q?(c)$
          then $x := f$ **fi**;
        $i := h(i)$ **od fi fi**

A Minimal Syntactic Slice

Simplify

```
if p?(i)
   then if q?(c)
           then x := f fi;
        i := h(i);
        while p?(i) do
           if q?(c)
              then x := f fi;
           i := h(i) od fi
```

Roll up the loop to get:

```
while p?(i) do
   if q?(c)
      then x := f fi;
   i := h(i) od
```

This is a *minimal syntactic slice*

(Proof: Deleting any statement gives a program which computes a different result for x)

Minimal Syntactic Slices

A minimal slice is not necessarily unique. For example:

$$x := 1;\ x := x + 2;\ x := 2;\ x := x + 1$$

Two different minimal slices are:

$$x := 1;\ x := x + 2 \qquad \text{and} \qquad x := 2;\ x := x + 1$$

Finding minimal syntactic slices is in general noncomputable.

The Generalised Mug Problem

while $p(i)$ **do**
  if $q?(c, i)$
    then $x := f;\ x := g(i)$ **fi**;
  $i := h(i)$ **od**

Split the **while** loop on the condition $\neg q?(c, i)$ using the converse to *Loop Merging*

Loop Merging

while B do S od

is equivalent to:

while B $\wedge$ **B**$'$ **do S od**; **while B do S od**

The Generalised Mug Problem

Split the loop and add some assertions:

while $p(i) \wedge \neg q(c,i)$ **do**
  if $q?(c,i)$
    then $x := f;\ x := g(i)$ **fi**;
  $i := h(i)$ **od**;
$\{p(i) \Rightarrow q(c,i)\}$;
if $p(i)$
  then $\{q(c,i)\}$;
    if $q(c,i)$
      then $x := f;\ c := g(i)$ **fi**;
    $i := h(i)$;
    while $p(i)$ **do**
      if $q(c,i)$
        then $x := f;\ c := g(i)$ **fi**;
      $i := h(i)$ **od fi**

Simplify (using the assertions):

while $p(i) \wedge \neg q(c,i)$ **do**
  $i := h(i)$ **od**;
if $p(i)$
  then $x := f;\ c := g(i)$;
    $i := h(i)$;
    while $p(i)$ **do**
      if $q(c,i)$
        then $x := f;\ c := g(i)$ **fi**;
      $i := h(i)$ **od fi**

The Generalised Mug Problem

Apply Constant Propagation:

while $p(i) \wedge \neg q(c,i)$ **do**
$i := h(i)$ **od**;
if $p(i)$
then $x := f;\ c := g(i);$
$i := h(i);$
while $p(i)$ **do**
if $q(c,i)$
then $c := g(i)$ **fi**;
$i := h(i)$ **od fi**

and Syntactic Slice:

while $p(i) \wedge \neg q(c,i)$ **do**
$i := h(i)$ **od**;
if $p(i)$ **then** $x := f$ **fi**

Collapse the loop to get a minimal semantic slice:

$x :=$ **if** $\forall j.\,(H(i,j,c) \Rightarrow p(j))$ **then** f **else** x **fi**

where $H(i,j,c)$ is:

$$\exists n.\,(j = h^n(i) \wedge \forall m < n.\,(p(h^m(i)) \wedge \neg q(h^m(i))) \wedge (\neg p(h^n(i)) \vee q(h^n(i))))$$

Further Generalisations

If a program **S** satisfies these two conditions:

- The program is deterministic;
- The variable x is assigned a constant value in one or more places.

Then we can derive a minimal semantic slice:

$x :=$ **if** WP$(\mathbf{S}, x = e_1)$ **then** e_1
elsif ...
elsif WP$(\mathbf{S}, x = e_n)$ **then** e_n
else x **fi**

The Representation Theorem

Let $\mathbf{S}\colon V \to V$, be any statement and let $\mathbf{x}$ be a list of all the variables in V. Then $\mathbf{S}$ is equivalent to:

$[\neg \mathrm{WP}(\mathbf{S}, \mathbf{false})];$
$\mathbf{x} := \mathbf{x}'.(\neg \mathrm{WP}(\mathbf{S}, \mathbf{x} \neq \mathbf{x}') \wedge \mathrm{WP}(\mathbf{S}, \mathbf{true}))$

If $\mathbf{S}$ is null-free (true for all WSL statements above the kernel level), then $\mathrm{WP}(\mathbf{S}, \mathbf{false})$ is $\mathbf{false}$ and $\mathbf{S}$ is equivalent to:

$\{\mathrm{WP}(\mathbf{S}, \mathbf{true})\};\ \mathbf{x} := \mathbf{x}'.(\neg \mathrm{WP}(\mathbf{S}, \mathbf{x} \neq \mathbf{x}'))$

So, by the definition of semi-refinement:

$$\mathbf{S} \preccurlyeq \mathbf{x} := \mathbf{x}'.(\neg \mathrm{WP}(\mathbf{S}, \mathbf{x} \neq \mathbf{x}'))$$

Therefore:

For any WSL program there exists a semantic slice which contains a single statement

Example

From [Tip 1985]:

if $p = q$
  then $x := 18$
  else $x := 17$ **fi**;
if $p \neq q$
  then $y := x$
  else $y := 2$ **fi**

The minimal semantic slice for y is:

$$y := y'.(\neg \mathsf{WP}(\textbf{if } p = q \textbf{ then } x := 18 \textbf{ else } x := 17 \textbf{ fi}; \textbf{if } p \neq q \textbf{ then } y := x \textbf{ else } y := 2 \textbf{ fi}, y \neq y'))$$

which simplifies to:

$$y := \textbf{if } p = q \textbf{ then } 2 \textbf{ else } 17 \textbf{ fi}$$

Slicing in FermaT

The FermaT transformation system implements most of these transformations, including syntactic slicing and constant propagation.

FermaT is available under the GNU GPL (General Public Licence) from the following web sites:

http://www.dur.ac.uk/~dcs6mpw/fermat.html
http://www.cse.dmu.ac.uk/~mward/fermat.html