1. 18 min talking
2. 12 min questions
3. Scripting langs are different
4. Plan: Start with motivating example
5. Plan: Introduce weirdness 1 step at a time

On the use of SSA with Scripting Languages

Paul Biggar and David Gregg

Department of Computer Science and Statistics
Trinity College Dublin

Static Single-Assignment Form Seminar
Autrans, France
27th April, 2009

On the use of SSA with Scripting Languages

Paul Biggar and David Gregg

Department of Computer Science and Statistics
Trinity College Dublin

Static Single-Assignment Form Seminar
Autrans, France
27th April, 2009

1. This PHP snippet can be 'intuitively' typed

Motivating Example

```php
function log ($printer, $prefix, $message) {
  $fout = "$prefix: $message";
  $printer->file_print ($fout);

  $cout = "$prefix: $message"
  $printer->console_print ($cout);
}
```

Motivating Example


```php
function log ($printer, $prefix, $message) {
  $fout = "$prefix: $message";
  $printer->file_print ($fout);

  $cout = "$prefix: $message"
  $printer->console_print ($cout);
}
```

1. Already in SSA - only 1 assignment to each var

In SSA

```
function log ($printer_0, $prefix_0, $message_0) {
  $fout_0 = $prefix_0 . ": " . $message_0;
  $printer_0->file_print ($fout_0);

  $cout_0 = $prefix_0 . ": " . $message_0;
  $printer_0->console_print ($cout_0);
}
```

In SSA


```
function log ($printer_0, $prefix_0, $message_0) {
  $fout_0 = $prefix_0 . ": " . $message_0;
  $printer_0->file_print ($fout_0);

  $cout_0 = $prefix_0 . ": " . $message_0;
  $printer_0->console_print ($cout_0);
}
```

Value numbering

```
1 function log ($printer_0, $prefix_0, $message_0) {
2   $fout_0 = $prefix_0 . ": " . $message_0;
3   $printer_0->file_print ($fout_0);
4
5   $printer_0->console_print ($fout_0);
6 }
```

Value numbering

```
1 function log ($printer_0, $prefix_0, $message_0) {
2   $fout_0 = $prefix_0 . ": " . $message_0;
3   $printer_0->file_print ($fout_0);
4
5   $printer_0->console_print ($fout_0);
6 }
```

Aliased parameters?

```
function log (&$printer, $prefix, $message) {
   ...
}

$p = new Printer;
log ($p, &$p->pre, &$p->mes);
```

Aliased parameters?


```
function log (&$printer, $prefix, $message) {
   ...
}

$p = new Printer;
log ($p, &$p->pre, &$p->mes);
```

1. Multiple names for the same heap object
2. Very simple to convert into SSA - the references scalars

- Java style

References in PHP

- Java style

1. Multiple names for the same memory location
2. No type declarations or signatures - differs from C++

- Java style
- C++ style

References in PHP

- Java style
- C++ style

1. PHP references are run-time values
2. Symbol table aliases
3. Can be references at some point, and non-refs at another point - again, unlike C++


```
$y = 1;
if (...)
  $x =& $y;
else
  $x = $y;

$x = 5;
print $y;
```

References in PHP cont.

```
$y = 1;
if (...)
  $x =& $y;
else
  $x = $y;

$x = 5;
print $y;
```

1. Call-time pass-by-ref
2. All parameters can be call-clobbered
3. Cant tell absence of aliasing

Aliased parameters?

```php
function log ($printer, $prefix, $message) {
  ...
}

$p = new Printer;
log ($p, &$p->pre, &$p->mes);
```

Aliased parameters?


```php
function log ($printer, $prefix, $message) {
  ...
}

$p = new Printer;
log ($p, &$p->pre, &$p->mes);
```

- What form of SSA to support alias analysis?

SSA + Alias analysis

- What form of SSA to support alias analysis?

- What form of SSA to support alias analysis?

`http://www.cs.man.ac.uk/~jsinger/ssa.html`

SSA + Alias analysis

- What form of SSA to support alias analysis?

`http://www.cs.man.ac.uk/~jsinger/ssa.html`

1. Not what I thought it was

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - Dynamic Single Assignment

Paul Feautrier. Dataflow analysis of array and scalar references. International Journal of Parallel Programming, 1991.

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - Dynamic Single Assignment

Paul Feautrier. Dataflow analysis of array and scalar references. International Journal of Parallel Programming, 1991.

1. Not clear how it works
2. Despite Singer's comment

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - Cytron and Gershbein

Ron Cytron and Reid Gershbein. Efficient accommodation of may-alias information in SSA form. PLDI 1993.

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - Cytron and Gershbein

Ron Cytron and Reid Gershbein. Efficient accommodation of may-alias information in SSA form. PLDI 1993.

1. Unclear how to modify SSA algorithms
2. C++ references? Designed for multi-level pointers

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - ~~Cytron and Gershbein~~
  - Extended SSA Numbering

Christopher Lapkowski and Laurie J. Hendren. Extended SSA numbering: Introducing SSA properties to language with multi-level pointers. Compiler Construction, 1998.

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - ~~Cytron and Gershbein~~
  - Extended SSA Numbering

Christopher Lapkowski and Laurie J. Hendren. Extended SSA numbering: Introducing SSA properties to language with multi-level pointers. Compiler Construction, 1998.

1. Requires strong type information

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - ~~Cytron and Gershbein~~
  - ~~Extended SSA Numbering~~
  - Extended Array SSA

Stephen Fink, Kathleen Knobe, and Vivek Sarkar. Unified analysis of array and object references in strongly typed languages. Static Analysis Symposium, 2000.

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - ~~Cytron and Gershbein~~
  - ~~Extended SSA Numbering~~
  - Extended Array SSA

Stephen Fink, Kathleen Knobe, and Vivek Sarkar. Unified analysis of array and object references in strongly typed languages. Static Analysis Symposium, 2000.

1. Pedigree: SGI, Mono, gcc
2. Worked on gcc
3. solves a lot of problems
4. very readable
5. clear in what problems is solves

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - ~~Cytron and Gershbein~~
  - ~~Extended SSA Numbering~~
  - ~~Extended Array SSA~~
  - Hashed SSA

Fred C. Chow, Sun Chan, Shin-Ming Liu, Raymond Lo, and Mark Streich. Effective representation of aliases and indirect memory operations in SSA form. Compiler Construction, 1996.

SSA + Alias analysis

- What form of SSA to support alias analysis?
  - ~~Dynamic Single Assignment~~
  - ~~Cytron and Gershbein~~
  - ~~Extended SSA Numbering~~
  - ~~Extended Array SSA~~
  - Hashed SSA

Fred C. Chow, Sun Chan, Shin-Ming Liu, Raymond Lo, and Mark Streich. Effective representation of aliases and indirect memory operations in SSA form. Compiler Construction, 1996.

1. Massimiliano Mantione will talk about this tomorrow
2. vars or sets of aliases, or some "name" ie heap node

- Virtual variables

What is HSSA?

- Virtual variables

1. Massimiliano Mantione will talk about this tomorrow
2. Annotates a statement

- Virtual variables
- Mu: may-use

What is HSSA?

- Virtual variables
- Mu: may-use

1. Massimiliano Mantione will talk about this tomorrow

- Virtual variables
- Mu: may-use
- Chi: may-def

What is HSSA?

- Virtual variables
- Mu: may-use
- Chi: may-def

1. Massimiliano Mantione will talk about this tomorrow
2. GVN and zero variables

- Virtual variables
- Mu: may-use
- Chi: may-def
- Space efficient representation

What is HSSA?

- Virtual variables
- Mu: may-use
- Chi: may-def
- Space efficient representation

1. Massimiliano Mantione will talk about this tomorrow
2. just drop chis, so might lose info

- Virtual variables
- Mu: may-use
- Chi: may-def
- Space efficient representation
- Drop indices to get out of SSA

What is HSSA?

- Virtual variables
- Mu: may-use
- Chi: may-def
- Space efficient representation
- Drop indices to get out of SSA

1. Massimiliano Mantione will talk about this tomorrow
2. ie during copy propagation

What is HSSA?

- Virtual variables
- Mu: may-use
- Chi: may-def
- Space efficient representation
- Drop indices to get out of SSA
- Must be careful not to move copies across live ranges

What is HSSA?

- Virtual variables
- Mu: may-use
- Chi: may-def
- Space efficient representation
- Drop indices to get out of SSA
- Must be careful not to move copies across live ranges

1. No longer able to due value numbering optimization from before
2. If we want to do any kind of value propagation, we have to be very conservative
3. But, maybe we can do something. fout and cout are touched in this example, but there will be others right?

Aliased parameters in SSA

```
function log ($printer_0, $prefix_0, $message_0) {
  MU ($printer_0)
  $fout_0 = $prefix_0 . ": " . $message_0;

  $printer_0->file_print ($fout_0);
  $printer_1 = CHI ($printer_0);
  $prefix_1 = CHI ($prefix_0);
  $message_1 = CHI ($message_0);
  $fout_1 = CHI ($fout_0);

  MU ($printer_1)
  MU ($fout_1)
  $cout_0 = $prefix_1 . ": " . $message_1;

  $printer_0->console_print ($cout_0);
  ...
}
```

Aliased parameters in SSA


```
function log ($printer_0, $prefix_0, $message_0) {
  MU ($printer_0)
  $fout_0 = $prefix_0 . ": " . $message_0;

  $printer_0->file_print ($fout_0);
  $printer_1 = CHI ($printer_0);
  $prefix_1 = CHI ($prefix_0);
  $message_1 = CHI ($message_0);
  $fout_1 = CHI ($fout_0);

  MU ($printer_1)
  MU ($fout_1)
  $cout_0 = $prefix_1 . ": " . $message_1;

  $printer_0->console_print ($cout_0);
  ...
}
```

Conservative SSA form is very pessimistic

Implication

Conservative SSA form is very pessimistic

1. Outer loop of something involving mandelbrot
2. everything is dead!!

Simpler?

```
function bastardized_mandel ($n)
{
  for ($y = 0; $y <= $n; $y++)
  {
    $imc = 0.28 * ($y - 12);
    for ($x = 0; $x <= 150; $x++)
    {
      $rec = 0.28 * ($x - 40) - 0.45;
      $re = $rec;
      $im = $imc;
      $color = 10;
      $re2 = $re * $re;
      $im2 = $im * $im;
    }
  }
}
```

Simpler?

```
function bastardized_mandel ($n)
{
  for ($y = 0; $y <= $n; $y++)
  {
    $imc = 0.28 * ($y - 12);
    for ($x = 0; $x <= 150; $x++)
    {
      $rec = 0.28 * ($x - 40) - 0.45;
      $re = $rec;
      $im = $imc;
      $color = 10;
      $re2 = $re * $re;
      $im2 = $im * $im;
    }
  }
}
```

1. get and set are called when reading or writing values
2. Complete access to interpreter internals
3. No longer know anything about uses and defs
4. Completely opaque to source-level compiler

C API handlers

- read_property
- read_dimension
- get
- set
- cast_object
- has_property
- unset_property
- ...

C API handlers

- read_property
- read_dimension
- get
- set
- cast_object
- has_property
- unset_property
- ...

1. simplified further
2. read $n on line 7: get handler!!
3. $y might not even be zero on first iteration

Mandelbrot again

```
1  function bastardized_mandel ($n)
2  {
3    $y = 0;
4
5    while (1)
6    {
7      if ($y > $n)
8        break;
9
10     $imc = 0.28 * ($y - 12);
11     ...
12     $y++;
13   }
14 }
15
16 bastardized_mandel (extension_function ());
```

Mandelbrot again


```
1  function bastardized_mandel ($n)
2  {
3    $y = 0;
4
5    while (1)
6    {
7      if ($y > $n)
8        break;
9
10     $imc = 0.28 * ($y - 12);
11     ...
12     $y++;
13   }
14 }
15
16 bastardized_mandel (extension_function ());
```

1. simplified further
2. read $n on line 7: get handler!!
3. $y might not even be zero on first iteration
4. CHI must now go between the read of $n and the read of $y
5. Even working out the example is head-wrecking
6. Cant kill anything

Mandelbrot in SSA

```
1  function bastardized_mandel ($n_0)
2  {
3    $y_0 = 0;
4
5    $y_1 = PHI ($y_0, $y_X)
6    $n_1 = PHI ($n_0, $n_X)
7    while (1)
8    {
9      $y_2 = CHI ($y_1);
10     if ($y_2 > $n_1)
11       break;
12
13     $imc_1 = CHI ($imc_0);
14     $imc_1 = 0.28 * ($y_2 - 12);
15     $y_3 = CHI ($y_2);
16     $imc_2 = CHI ($imc_1);
17
18     ...
19   }
20 }
21
22 bastardized_mandel (extension_function ());
```

Mandelbrot in SSA

```
1  function bastardized_mandel ($n_0)
2  {
3    $y_0 = 0;
4
5    $y_1 = PHI ($y_0, $y_X)
6    $n_1 = PHI ($n_0, $n_X)
7    while (1)
8    {
9      $y_2 = CHI ($y_1);
10     if ($y_2 > $n_1)
11       break;
12
13     $imc_1 = CHI ($imc_0);
14     $imc_1 = 0.28 * ($y_2 - 12);
15     $y_3 = CHI ($y_2);
16     $imc_2 = CHI ($imc_1);
17
18     ...
19   }
20 }
```

1. Single unknown type - may as well give up

Unknown types propagate

- local symbol table
- global symbol table
- return values
- reference parameters
- callee parameters

Unknown types propagate

- local symbol table
- global symbol table
- return values
- reference parameters
- callee parameters

Def-use chains cannot be trivially obtained without analysis
even for scalars!!

Implication

Def-use chains cannot be trivially obtained without analysis
even for scalars!!

1. Perhaps with TBAA or ATAA

SSA in phc

- Intra-procedural (only) analysis

SSA in phc

- Intra-procedural (only) analysis

- ~~Intra-procedural (only) analysis~~
- Derive def-use chains from whole-program analysis

SSA in phc

- ~~Intra-procedural (only) analysis~~
- Derive def-use chains from whole-program analysis

1. Simultaneously

SSA in phc

- ~~Intra-procedural (only) analysis~~
- Derive def-use chains from whole-program analysis
  - Abstract Execution / Interpretation
  - Points-to analysis
  - Conditional Constant-propagation
  - Type-inference

Conditional Pointer Aliasing and Constant Propagation. Anthony Pioli. MS Thesis, SUNY at New Paltz Technical Report #99-102, January 1999.

SSA in phc

- ~~Intra-procedural (only) analysis~~
- Derive def-use chains from whole-program analysis
  - Abstract Execution / Interpretation
  - Points-to analysis
  - Conditional Constant-propagation
  - Type-inference

Conditional Pointer Aliasing and Constant Propagation. Anthony Pioli. MS Thesis, SUNY at New Paltz Technical Report #99-102, January 1999.

1. Fabrice mentioned this earlier RE book

- End-to-end compiler IR

Benefits of SSA

- End-to-end compiler IR

1. Have to do them first

Benefits of SSA

- ~~End-to-end compiler IR~~
- Sparse propagation framework

Benefits of SSA

- ~~End-to-end compiler IR~~
- Sparse propagation framework

- ~~End-to-end compiler IR~~
- ~~Sparse propagation framework~~
- Sparse analysis framework (execution-time)

Benefits of SSA

- ~~End-to-end compiler IR~~
- ~~Sparse propagation framework~~
- Sparse analysis framework (execution-time)

- ~~End-to-end compiler IR~~
- ~~Sparse propagation framework~~
- Sparse analysis framework (execution-time)
- Sparse representation (memory usage)

Benefits of SSA

- ~~End-to-end compiler IR~~
- ~~Sparse propagation framework~~
- Sparse analysis framework (execution-time)
- Sparse representation (memory usage)

- Perform analyses on “SSA” while building SSA
  - Integrate SSA building into the abstract execution
  - Intuitively might be possible.

Open research problem (I think)

- Perform analyses on “SSA” while building SSA
  - Integrate SSA building into the abstract execution
  - Intuitively might be possible.

1. Not like operator+ in C++

Misc

- Userspace handlers - syntax hides function calls.

Misc

- Userspace handlers - syntax hides function calls.

1. Must drop indices - which is relatively convenient due to HSSA

Misc

- Userspace handlers - syntax hides function calls.
- Renaming not possible

Misc

- Userspace handlers - syntax hides function calls.
- Renaming not possible

- SSA is hard in scripting languages
- Perform propagation algorithm and alias analysis before SSA construction
- Can still use SSA for other analyses

Summary

- SSA is hard in scripting languages
- Perform propagation algorithm and alias analysis before SSA construction
- Can still use SSA for other analyses

Thanks

Q.

What else am I an expert in?

A.

Um, I suppose, maybe, scripting languages?

- Compiler research landscape
- (Informal) Semantics
- Optimization and analysis techniques

Thanks

Thanks

Q.

What else am I an expert in?

A.

Um, I suppose, maybe, scripting languages?

- Compiler research landscape
- (Informal) Semantics
- Optimization and analysis techniques