- 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

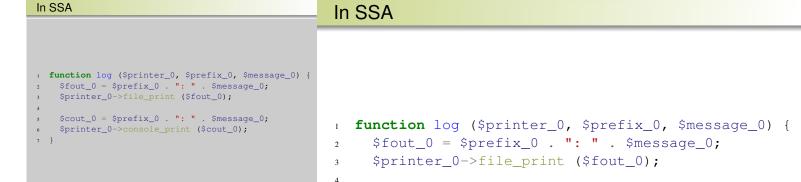
On the use of SSA with Scripting Languages

Paul Biggar and David Gregg

Department of Computer Science and Statistics Trinity College Dublin

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

	Motivating Example	Motivating Example
e PHP snippet be 'intuitively' ed	<pre>function log (\$printer, \$prefix, \$message) { \$fout = "\$prefix: \$message"; \$printer->file_print (\$fout); \$cout = "\$prefix: \$message" \$printer->console_print (\$cout); } </pre>	<pre>function log (\$printer, \$prefix, \$message) { \$fout = "\$prefix: \$message"; \$printer->file_print (\$fout); \$cout = "\$prefix: \$message" \$printer->console_print (\$cout); }</pre>

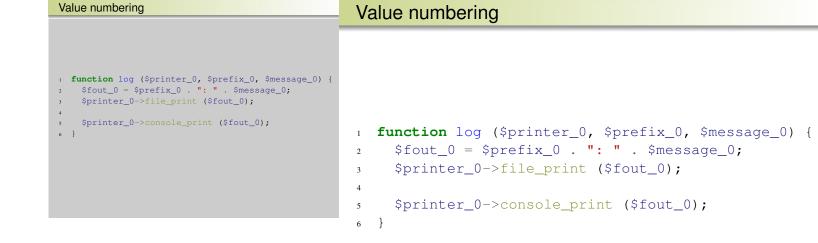


6 7

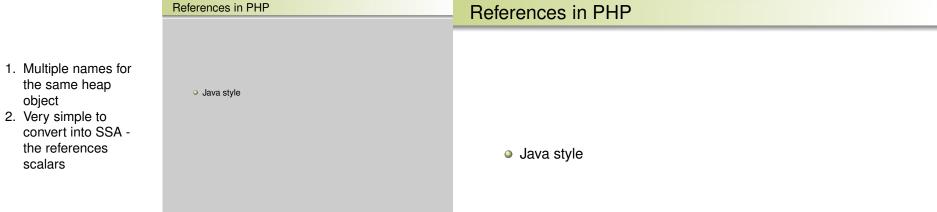
```
s $cout_0 = $prefix_0 . ": " . $message_0;
```

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

 Already in SSA only 1 assignment to each var



Aliased parameters?	Aliased parameters?
<pre>function log (\$printer, \$prefix, \$message) { }</pre>	<pre>1 function log (\$printer, \$prefix, \$message) { 2 3 } 4 5 \$p = new Printer; 6 log (\$p, &\$p->pre, &\$p->mes);</pre>



object

scalars

	References in PHP	References in PHP
 Multiple names for the same memory location No type declarations or signatures - differs from C++ 	• Java style • C++ style	Java styleC++ style

1. Multiple names

location 2. No type

References in PHP cont.

- 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 = \$y; \$ \$x = 5; \$print \$y;

References in PHP cont.

 $_{1}$ \$y = 1; ² if (...) $\$_X = \& \$_V;$ 3 4 else $_{5}$ \$x = \$y;6 $_{7}$ \$x = 5; » print \$y;

, \$prefix, \$message) { ->mes);

```
    Call-time 
pass-by-ref
```

- 2. All parameters can be call-clobbered
- 3. Cant tell absence of aliasing

SSA + Alias analysis	SSA + Alias analysis
• What form of SSA to support 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

• 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.

Not what I thought it was

SSA + Alias analysis

• What form of SSA to support alias analysis?

- Dynamic Single Assignment
- Cytron and Gershbein

1. Not clear how it works

2. Despite Singer's comment

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

• What form of SSA to support alias analysis?

- Dynamic Single Assignment
- Oytron and Gershbein

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

What form of SSA to support alias analysis?

- Dynamic Single Assignment
- Gytron 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.

What form of SSA to support alias analysis?

- Dynamic Single Assignment
- Gytron 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. 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
- Overage Contract of Contrac
- 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.

What form of SSA to support alias analysis?

- Dynamic Single Assignment
- Gytron 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.

Requires strong type information

- 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

What form of SSA to support alias analysis?

- Dynamic Single Assignment
- Or 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
- Gytron 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.

What is HSSA?

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

Virtual variables

Virtual variables

What is HSSA?

Virtual variables

1. Massimiliano

about this

tomorrow 2. Annotates a

statement

Mantione will talk

Mu: may-use

Mu: may-use

Virtual variables

What is HSSA?

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

- Virtual variables
- Mu: may-use
- Ohi: may-def
- e

 Massimiliano Mantione will talk about this tomorrow

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

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

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

- Virtual variables
- Mu: may-use
- Ohi: 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

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

- Virtual variables
- Mu: may-use
- Ohi: 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

```
1 function log ($printer_0, $prefix_0, $message_0) {
    MU (Sprinter 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 (Sprinter 1)
    MU ($fout 1)
     $cout_0 = $prefix_1 . ": " . $message_1;
15
    $printer_0->console_print ($cout_0);
17 }
```

Aliased parameters in SSA

```
function log ($printer 0, $prefix 0, $message 0) {
     MU ($printer 0)
2
     $fout 0 = $prefix 0 . ": " . $message 0;
3
     $printer 0->file print ($fout 0);
5
     $printer_1 = CHI ($printer_0);
6
     $prefix_1 = CHI ($prefix_0);
     $message_1 = CHI ($message_0);
8
     fout 1 = CHI (fout 0);
9
10
     MU ($printer 1)
11
     MU ($fout 1)
12
     $cout 0 = $prefix 1 . ": " . $message 1;
13
14
     $printer 0->console print ($cout 0);
15
16
     . . .
17
```

Implication	Implication
Conservative SSA form is very pessimistic	
	Concernative SSA form is very possimistic
	Conservative SSA form is very pessimistic

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

dead!!

function bastardized mandel (\$n) 2 { for (\$y = 0; \$y <= \$n; \$y++) 3 4 simc = 0.28 * (sv - 12);5 for (\$x = 0; \$x <= 150; \$x++) 6 7 srec = 0.28 * (sx - 40) - 0.45;8 \$re = \$rec; \$im = \$imc; 10 color = 10;11 \$re2 = \$re * \$re; 12 sim2 = sim * sim;13 14 15 16 }

Simpler?

Simpler?

3

5

6

7

8

9

11

12

```
function bastardized_mandel ($n)
2
    for ($y = 0; $y <= $n; $y++)
      simc = 0.28 * (sy - 12);
      for (\$x = 0; \$x \le 150; \$x++)
         srec = 0.28 * (sx - 40) - 0.45;
         sre = srec;
         sim = simc;
10
         color = 10;
         $re2 = $re * $re;
         \sin 2 = \sin * \sin;
```

- 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

C API handlers

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

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

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

iteration

Mandelbrot again
<pre>function bastardized_mandel (\$n)</pre>
2 {
$_{3}$ $\$y = 0;$
4
s while (1)
6 {
<pre>1 if (\$y > \$n)</pre>
8 break;
9
10 \$imc = 0.28 * (\$y - 12);
12 \$ ¥ + ;
13 }
14 }
15
<pre>16 bastardized_mandel (extension_function ());</pre>

Mandelbrot again

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

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

M	andelbrot in SSA
1	<pre>function bastardized_mandel (\$n_0)</pre>
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	<pre>if (\$y_2 > \$n_1)</pre>
11	break;
12	
13	\$imc_1 = CHI (\$imc_0);
14	<pre>\$imc_1 = 0.28 * (\$y_2 - 12);</pre>
15	\$y_3 = CHI (\$y_2);
16	\$imc_2 = CHI (\$imc_1);
17	
18	
19	}
20	}
21	
22	<pre>bastardized_mandel (extension_function ());</pre>

Mandelbrot in SSA

```
function bastardized mandel ($n 0)
2
     v 0 = 0;
3
4
     y_1 = PHI (y_0, y_X)
5
     n_1 = PHI (n_0, n_X)
6
     while (1)
7
8
       y_2 = CHI (y_1);
9
       if (\$y_2 > \$n_1)
10
         break;
11
12
       simc 1 = CHI (simc 0);
13
       simc 1 = 0.28 * (sy 2 - 12);
14
       y_3 = CHI (y_2);
15
16
       simc 2 = CHI (simc 1);
17
18
        . . .
19
20
```

 Single unknown type - may as well give up

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

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

Implication	Implication
Def-use chains cannot be trivially obtained without analysis even for scalars!!	
	Def-use chains cannot be trivially obtained without analysis <i>even for scalars</i> !!

SSA in phc	SSA in phc
 Intra-procedural (only) analysis 	
	Intra-procedural (only) analysis

1. Perhaps with TBAA or ATAA

SSA in phc

SSA in phc

• Intra procedural (only) analysis

• Derive def-use chains from whole-program 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
 - 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.

- 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. Simultaneously

Benefits of SSA

End-to-end compiler IR

1. Fabrice mentioned this earlier RE book

• End-to-end compiler IR

• End to end compiler IR

• Sparse propagation framework

1. Have to do them first

- End to end compiler IR
- Sparse propagation framework

- 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)

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

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

Open research problem (I think)

• Perform analyses on "SSA" while building SSA

- Integrate SSA building into the abstract execution
- Intuitively might be possible.

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

• Userspace handlers - syntax hides function calls.

1. Not like operator+ in C++

• Userspace handlers - syntax hides function calls.

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

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

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

Summary

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

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

Thanks Thanks Thanks What else am I an expert in?

Um, I suppose, maybe, scripting languages?

- Compiler research landscape
- Informal) Semantics

Thanks

Optimization and analysis techniques

Q.

What else am I an expert in?

A

Um, I suppose, maybe, scripting languages?

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