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
1. This PHP snippet can be 'intuitively' typed:

```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
function log ($printer_0, $prefix_0, $message_0) {
    $fout_0 = $prefix_0 . "\": " . $message_0;
    $printer_0->file_print ($fout_0);
    $printer_0->console_print ($fout_0);
}
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
<table>
<thead>
<tr>
<th>References in PHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Multiple names for the same memory location</td>
</tr>
<tr>
<td>2. No type declarations or signatures - differs from C++</td>
</tr>
</tbody>
</table>

- Java style
- C++ style

- 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++

```php
$y = 1;
if (...) {
    $x = & $y;
} else {
    $x = $y;
}
$x = 5;
print $y;
```
Aliased parameters?

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

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

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

```
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?

http://www.cs.man.ac.uk/~jsinger/ssa.html
SSA + Alias analysis

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


1. Not what I thought it was
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. Not clear how it works
2. Despite Singer's comment
### SSA + Alias analysis

<table>
<thead>
<tr>
<th>What form of SSA to support alias analysis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Single Assignment</td>
</tr>
<tr>
<td>Cytron and Gershbein</td>
</tr>
<tr>
<td>Extended SSA Numbering</td>
</tr>
</tbody>
</table>


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

---

Trinity College Dublin
SSA + Alias analysis

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

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
- Hashed SSA

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

<table>
<thead>
<tr>
<th>What is HSSA?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual variables</td>
</tr>
<tr>
<td>Mu: may-use</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What is HSSA?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual variables</td>
</tr>
<tr>
<td>Mu: may-use</td>
</tr>
</tbody>
</table>
1. Massimiliano Mantione will talk about this tomorrow

<table>
<thead>
<tr>
<th>What is HSSA?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Virtual variables</td>
</tr>
<tr>
<td>- Mu: may-use</td>
</tr>
<tr>
<td>- Chi: may-def</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What is HSSA?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Virtual variables</td>
</tr>
<tr>
<td>- Mu: may-use</td>
</tr>
<tr>
<td>- Chi: may-def</td>
</tr>
</tbody>
</table>
1. Massimiliano Mantione will talk about this tomorrow
2. GVN and zero variables

What is HSSA?

- 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
- Drop indices to get out of SSA

---

1. Massimiliano Mantione will talk about this tomorrow
2. just drop chis, so might lose info
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. Massimiliano Mantione will talk about this tomorrow
2. ie during copy propagation
Aliased parameters in SSA

```c
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);
    ...
}
```

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?
Implication

Conservative SSA form is very pessimistic
Simpler?

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

Simpler?

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

```c
function bastardized_mandel ($n$) {
    $y = 0;
    while (1) {
        if ($y > $n)
            break;
        $imc = 0.28 * ($y - 12);
        ...
        $y++;
    }
    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
Unknown types propagate

1. Single unknown type - may as well give up

- local symbol table
- global symbol table
- return values
- reference parameters
- callee parameters
Def-use chains cannot be trivially obtained without analysis

*even for scalars!!*
1. Perhaps with TBAA or ATAA

- Intra-procedural (only) analysis

- Intra-procedural (only) analysis
SSA in phc

- 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

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

1. Fabrice mentioned this earlier RE book

- End-to-end compiler IR

- End-to-end compiler IR
1. Have to do them first

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

- End-to-end compiler IR
- Sparse propagation framework
Benefits of SSA

- 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)
- Sparse representation (memory usage)
Perform analyses on “SSA” while building SSA
  - Integrate SSA building into the abstract execution
  - Intuitively might be possible.
**Misc**

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
SSA is hard in scripting languages
Perform propagation algorithm and alias analysis before SSA construction
Can still use SSA for other analyses
Q. What else am I an expert in?

A. Um, I suppose, maybe, scripting languages?
- Compiler research landscape
- (Informal) Semantics
- Optimization and analysis techniques