

Static Analysis for PHP

Semester Project
Fall 2009

Etienne Kneuss
EPFL

PHP

- Weak & Dynamic Typing
- Compiler optimized for speed, not safety
- Large internal API (> 2500 functions)
- All kinds of dynamic features
 - “If it can be done, it usually can be done dynamically”

The problem

- Lot of room for run-time errors
 - Nearly all possible errors happen at run-time
- Most of those errors will be non-fatal
- Until recently, PHP was shipped to ignore lots of errors by default
 - Lots of broken or badly written scripts

Previous Work

- **PHP-Sat**
 - Mostly (only?) structure based
- **PHPMD** (PHP Mess Detector)
 - Program metrics
 - Some semantic checks
- **Pixy**
 - Taint-analysis, inter-procedural, for PHP4

This solution

- Structural checks
- Semantic checks
- Data/Type flow analysis
- Based on the grammar of PHP5.3

Why do types matter?

- PHP does type juggling

- switch

- What about ctype_digit ?

→ Relying on it is a problem waiting to happen:

[#50696](#), [#49057](#), [#34772](#), [#25763](#), [#24905](#), ...

```
<?php
switch(0) {
    case "true":
        ...
```

- Non-scalar types

Analysis phases

- *Lexing (Jflex)*
- *Parsing (modified CUP)*
- *ST \rightarrow AST*
- AST Transformations
- Structural checks
- API
- Semantic analysis
- *AST \rightarrow CFG*
- Flow analysis

AST Transformations

- Allows to freely transform the AST
- Used for:
 - Include resolver
 - Annotations

Include Resolver

- Link ASTs
- Support some expressions:
 - `dirname()`, constants, string literals, concatenation

`include dirname(__DIR__).'./path/to/file.php';`

- One problem: `include` is an expression!

Annotations

- Comments preceding declarations
- Compatible with phpDocumentor

```
/**  
 * My super concat function  
 * @param string $foo  
 * @param string $bar  
 * @return string  
 */  
function concat ($foo, $bar) {  
    return $foo.$bar;  
}
```

Structural Checks

- Looks for common mistakes
 - Conditional class/functions declarations
 - Call-time pass by reference
 - Variable variables (\$\$a)

```
function foo($a) {  
    return ++$a;  
}  
$b = 2;  
foo(&$b);  
echo $b;
```

API

- Support for API importation
 - e.g. `defined(name: String): Boolean`

```
<function name="defined">  
  <return><type name="bool"></type></return>  
  <args>  
    <arg opt="0"><type name="string"></type></arg>  
  </args>  
</function>
```

Semantic analysis

- Attaches symbols to identifiers
 - Double declarations
 - Inheritance cycles
- Define variable scopes

Data types flow analysis

- Model the flow of types for each values in a series of statements
- The result is a model of the possible types for each values at each program point
 - We can typecheck the result

Examples

Multiple verbosity

```
<?php  
$a = 2;
```

```
if ($a = 3) {  
    $b = 2;  
}
```



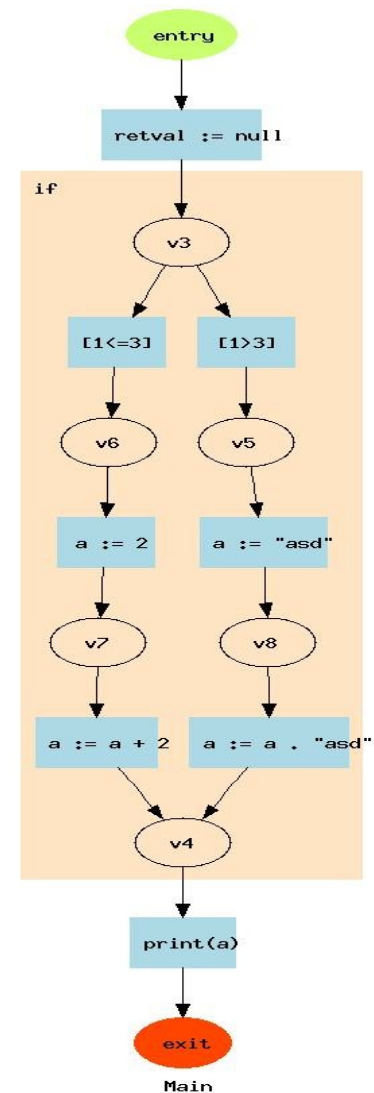
```
colder@mig9 ~/git/phpanalysis $ ./phpanalysis test.php  
colder@mig9 ~/git/phpanalysis $ ./phpanalysis --verbose test.php  
test.php:4 Notice: Potential mistake: assignation used in an if condition  
if ($a = 3) {  
    ^  
  
1 notice occurred.
```


Branches

```
<?php
```

```
if (1 > 3) {  
    $a = "asd";  
    $a = $a."asd";  
} else {  
    $a = 2;  
    $a = $a+2;  
}
```

```
echo $a;
```



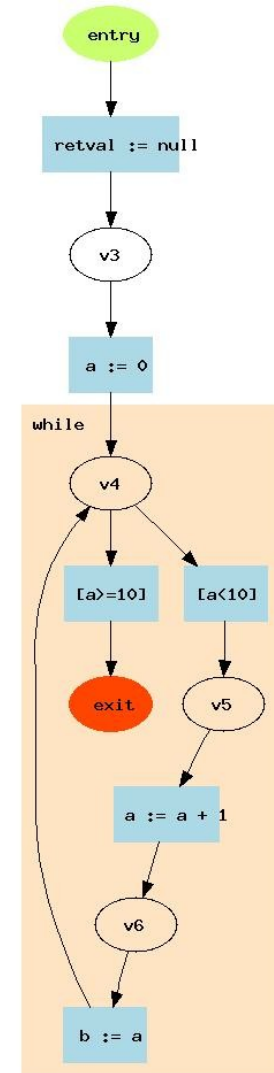
```
colder@mig9 ~/git/phpanalysis $ ./phpanalysis --debug test.php  
node entry has env [ ]  
node exit has env [ retval => TNull; a => {TString,TInt} ]  
node v3 has env [ retval => TNull ]  
node v4 has env [ retval => TNull; a => {TString,TInt} ]  
node v5 has env [ retval => TNull ]  
node v6 has env [ retval => TNull ]  
node v7 has env [ retval => TNull; a => TInt ]  
node v8 has env [ retval => TNull; a => TString ]
```

Loops

```
<?php
$a = 0;
while($a < 10) {
    $a = $a + 1;
    $b = $a;
}
```

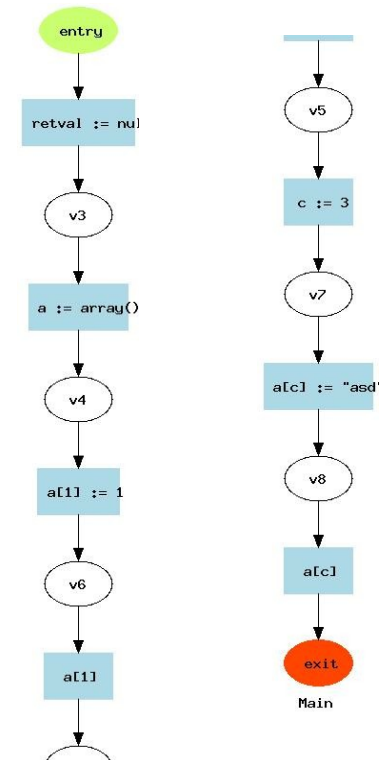


```
colder@mig9 ~/git/phpanalysis $ ./phpanalysis --debug test.php
node entry has env [ ]
node exit has env [ b => {TNull,TInt}; retval => TNull; a => TInt ]
node v3 has env [ retval => TNull ]
node v4 has env [ b => {TNull,TInt}; retval => TNull; a => TInt ]
node v5 has env [ b => {TNull,TInt}; retval => TNull; a => TInt ]
node v6 has env [ b => {TNull,TInt}; retval => TNull; a => TInt ]
```



Array types

```
<?php  
$a = array();  
$a[1] = 1;  
$c = 3;  
$a[$c] = "asd";
```



```
colder@vpn-a-183-154 ~/Documents/git/phpanalysis $ ./phpanalysis --debug test.php  
node entry has env [  ]  
node exit has env [ a => Array[1 => {TInt,TString} (TString)]; c => TInt; retval => TNull ]  
node v3 has env [ retval => TNull ]  
node v4 has env [ a => Array[]; retval => TNull ]  
node v5 has env [ a => Array[1 => TInt]; retval => TNull ]  
node v6 has env [ a => Array[1 => TInt]; retval => TNull ]  
node v7 has env [ a => Array[1 => TInt]; c => TInt; retval => TNull ]  
node v8 has env [ a => Array[1 => {TInt,TString} (TString)]; c => TInt; retval => TNull ]
```

Object Types

```
<?php  
class A {  
    public $foo = 2;  
}
```

```
$a = new A;  
$b = $a;
```



```
colder@mig9 ~/git/phpanalysis $ ./phpanalysis --debug test.php  
node entry has env [ ]  
node exit has env [ retval => TNull; a => (#7->Object(A)[foo => TInt]()); b => (#7->Object(A)[foo => TInt]()) ]  
node v4 has env [ retval => TNull ]  
node v5 has env [ a => (#7->Object(A)[foo => TInt]()); retval => TNull ]
```

Conditional types filtering

- “remembering” checks
- Every values have a boolean value:

TRUE	FALSE
Arrays, integers, floats, strings, resources , objects , true	Arrays, integers, floats, strings, false , null

- Lots of functions possibly return *false* on error
- We don't want *false* to pollute our results, if properly checked!

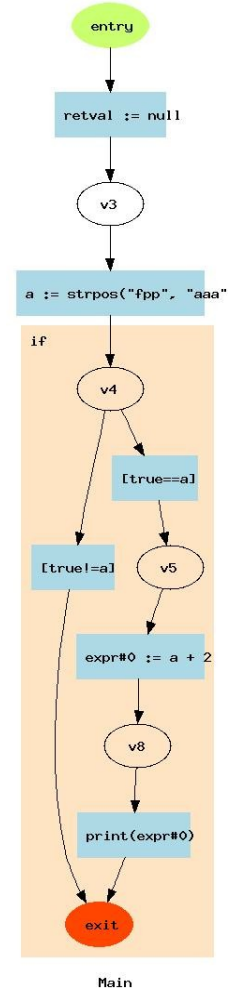
Conditional type filtering

```
<?php
$a = strpos("fpp", "aaa");

if ($a) {
    echo $a+2;
}
```



```
colder@mig9 ~/git/phpanalysis $ ./phpanalysis --debug test.php
node entry has env [ ]
node exit has env [ retval => TNull; a => {TFalse,TInt}; expr#0 => {TInt,TNull} ]
node v3 has env [ retval => TNull ]
node v4 has env [ a => {TFalse,TInt}; retval => TNull ]
node v5 has env [ a => TInt; retval => TNull ]
node v8 has env [ retval => TNull; a => TInt; expr#0 => TInt ]
```



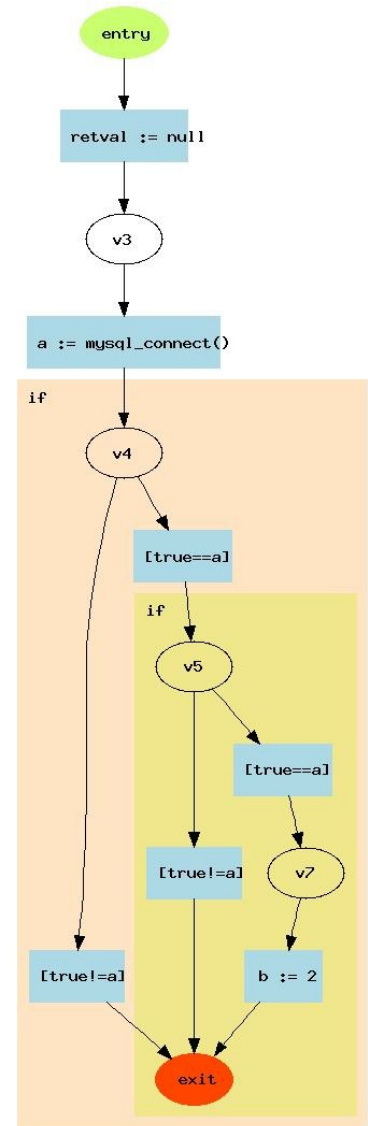
Redundant tests

```
<?php
$a = mysql_connect();

if ($a) {
    if ($a) {
        $b = 2;
    }
}
```



```
colder@mig9 ~/git/phpanalysis $ ./phpanalysis --debug test.php
node entry has env [ ]
node exit has env [ b => {TNull,TInt}; retval => TNull; a => {TResource,TFalse} ]
node v3 has env [ retval => TNull ]
node v4 has env [ retval => TNull; a => {TResource,TFalse} ]
node v5 has env [ retval => TNull; a => TResource ]
node v7 has env [ retval => TNull; a => TResource ]
test.php:5 Notice: Redundant or incompatible check
    if ($a) {
        ^
1 notice occurred.
```



Tests on real code

- Used to find 4 mistakes in 2 scripts written by a colleague!
- Too many false positives in complex, modular, OOP code.
 - **134** false notices found in a 370 lines class containing 9 methods
 - Increasing the signal-to-noise ratio is the next challenge!

Future Work

- Stabilize the support for every PHP5.3 features
- Provide a decent model for references
- Improve annotations for complex arrays
- Add support for in-code annotations?
- ...