Pfff part 2: Analyzing PHP

Programmer's manual and Implementation

Yoann Padioleau
yoann.padioleau@facebook.com

February 23, 2010

Short Contents

1 Introduction 9

I Using pfff 18

2 Examples of Use 19

3 Analysis Building Blocks 20

4 The Code Database 30

5 Global Analysis 36

6 Static Analysis 42

7 Dynamic Analysis 45

8 Finding Code 46

9 Checking Code 48

10 Transforming Code 49

11 Software Metrics 50

12 Other Services 52

II Internals 53

13 Implementation overview 54

14 Control Flow Graph 56

15 Data Flow Analysis 74

16 Typing 76

17 Xdebug traces 78

18 Code Rank 86

19 Cyclomatic Complexity 91

20 Dead code 93

21 Tainted string analysis 109

22 PHP Built-in Functions, Classes, and Globals 111

23 PHPUnit 121

24 Auxillary Code 122

Conclusion 123

A Testing sample code 124

B Extra code 129

C Indexes 135

D References 138

Contents

1 Introduction **9**
- 1.1 Why another PHP analyzer ? . . . 9
- 1.2 Features . . . 10
- 1.3 Copyright . . . 11
- 1.4 Getting started . . . 12
  - 1.4.1 Requirements . . . 12
  - 1.4.2 Compiling . . . 13
  - 1.4.3 Quick example of use . . . 13
  - 1.4.4 The `pfff_db` command-line tool . . . 14
  - 1.4.5 The `pfff_browser` GUI . . . 14
- 1.5 Source organization . . . 14
- 1.6 API organization . . . 14
- 1.7 Plan . . . 17
- 1.8 About this document . . . 17

I Using pfff **18**

2 Examples of Use **19**

3 Analysis Building Blocks **20**
- 3.1 Control flow graph . . . 20
  - 3.1.1 Introduction . . . 20
  - 3.1.2 Interface . . . 21
  - 3.1.3 `pfff_db -cfg_php` . . . 23
  - 3.1.4 The CFG type . . . 23
  - 3.1.5 Other functions . . . 26
- 3.2 Data flow fixpoint . . . 26
- 3.3 Scope annotation . . . 27
  - 3.3.1 Local/Globals . . . 27
  - 3.3.2 Free variables . . . 27
  - 3.3.3 Namespace . . . 27
- 3.4 Types . . . 28
- 3.5 Xdebug traces . . . 28

3.6 Comment extraction . . . 28
3.7 Tag extraction . . . 29
3.8 AST simplication . . . 29
3.9 Visitors . . . 29

4 The Code Database **30**
4.1 Indexing the code . . . 30
4.2 Entities, the database primary keys . . . 34
4.3 Defs . . . 35
4.4 Uses . . . 35
4.5 Misc . . . 35

5 Global Analysis **36**
5.1 Call graph . . . 36
5.2 Include graph . . . 39
5.3 Bottom-up analysis . . . 39
5.4 Handling built-in functions . . . 39
5.5 Call graph for dynamic calls, function aliasing . . . 41
5.6 Call graph for method calls, object aliasing . . . 41

6 Static Analysis **42**
6.1 Dead code . . . 42
6.2 Type inference . . . 43
6.3 Tainted analysis . . . 44
6.4 Static test coverage . . . 44

7 Dynamic Analysis **45**
7.1 Type extractions . . . 45
7.2 Value extractions . . . 45
7.3 PHPUnit . . . 45
7.4 Test coverage . . . 45

8 Finding Code **46**
8.1 Finding files . . . 46
8.2 Finding functions . . . 47
8.3 Finding expression positions . . . 47
8.4 `sgrep_php`, syntactical grep . . . 47

9 Checking Code **48**
9.1 Statistical bug finding . . . 48

10 Transforming Code **49**

11 Software Metrics **50**
11.1 Basic metrics . . . 50
11.2 Code rank . . . 50
11.3 Cyclomatic complexity . . . 50
11.3.1 Interface . . . 50
11.3.2 `pfff_db -cyclomatic_php` . . . 51

12 Other Services **52**
12.1 Statistics . . . 52
12.2 Testing . . . 52
12.3 Debugging . . . 52

II Internals **53**

13 Implementation overview **54**

14 Control Flow Graph **56**
14.1 Overview . . . 56
14.2 Types . . . 58
14.3 Main entry point . . . 59
14.4 Algorithm . . . 60
14.5 Debugging information . . . 70
14.6 Extra code . . . 71

15 Data Flow Analysis **74**

16 Typing **76**
16.1 Trivial typing . . . 76

17 Xdebug traces **78**

18 Code Rank **86**

19 Cyclomatic Complexity **91**

20 Dead code **93**

21 Tainted string analysis **109**

22 PHP Built-in Functions, Classes, and Globals **111**

23 PHPUnit **121**

24 Auxillary Code **122**

Conclusion **123**

A Testing sample code **124**

B Extra code **129**
B.1 aliasing_function_php.mli . . . 129
B.2 analysis_dynamic_php.mli . . . 129
B.3 analysis_static_php.mli . . . 129
B.4 annotation_php.mli . . . 129
B.5 ast_entity_php.mli . . . 129
B.6 bottomup_analysis_php.mli . . . 130
B.7 builtins_php.mli . . . 130
B.8 callgraph_php.mli . . . 130
B.9 checking_php.mli . . . 130
B.10 code_rank_php.mli . . . 130
B.11 comment_annotater_php.mli . . . 130
B.12 controlflow_php.mli . . . 130
B.13 controlflow_build_php.mli . . . 131
B.14 database_php.mli . . . 131
B.15 database_php_build.mli . . . 131
B.16 database_php_query.mli . . . 131
B.17 database_php_statistics.mli . . . 131
B.18 dataflow_php.mli . . . 131
B.19 deadcode_php.mli . . . 131
B.20 dependencies_php.mli . . . 131
B.21 entities_php.mli . . . 131
B.22 entity_php.mli . . . 132
B.23 finder_php.mli . . . 132
B.24 freevars_php.mli . . . 132
B.25 graph_php.mli . . . 132
B.26 include_require_php.mli . . . 132
B.27 info_annotater_php.mli . . . 132
B.28 lib_analyze_php.mli . . . 132
B.29 namespace_php.mli . . . 132
B.30 normalize_php.mli . . . 132
B.31 scoping_php.mli . . . 133
B.32 smpl_php.mli . . . 133
B.33 statistics_php.mli . . . 133
B.34 tainted_php.mli . . . 133
B.35 test_analyze_php.mli . . . 133
B.36 type_annotater_php.mli . . . 133
B.37 typing_php.mli . . . 133
B.38 typing_trivial_php.mli . . . 133
B.39 typing_weak_php.mli . . . 133
B.40 visitor2_php.mli . . . 134
B.41 xdebug.mli . . . 134

C Indexes **135**

D References **138**

Chapter 1

Introduction

1.1 Why another PHP analyzer ?

pfff (PHP Frontend For Fun) is mainly an OCaml API to write static analysis, code visualizations, code navigations, or style-preserving source-to-source transformations such as refactorings on PHP source code. I have described in a previous manual [**?**] [1] the parsing frontend part of pfff (corresponding to the code located in `parsing_php/`). Parsing is only one part of a programming language analysis infrastructure. This manual [2] describes the second part with the different analysis, static or dynamic, provided by pfff (with code located in `analysis_php/`).

There are already multiple projects that makes it possible to analyze PHP code. For static analysis there is:

- Etiennes Kneuss type inference [**?**] (written in Scala)
- Patrick Camphuiksen soft typing [**?**]
- Minamide incorrect embeded HTML detector [**?**]
- Pixy cross-scripting attack detector [**?**]
- Sebastian bergmann deadcode detector, clone detector, and various software metrics [**?**]
- HPHP global callgraph analysis and type inference (and of course all its compiler related technology optimisations) [**?**]
- The analysis in PHC [**?**]
- Daniel Corson lex-pass refactorer [**?**]

[1] FACEBOOK: in `~pad/pfff/docs/manual/Parsing_php.pdf`
[2] FACEBOOK: in `~pad/pfff/docs/manual/Analyze_php.pdf`

- Many tools (ab)using the PHP tokenizer to provide bug-finder or coding-style checkers

For dynamic analysis there is:

- xdebug debugger, profiler, and tracer [**?**]
- xhprof profiler [**?**]
- HPHP monitoring for memory leak, lock contention, infinite recursion detection, and cpu [**?**]
- The reflective library of PHP [**?**]
- Many tools which are thin wrapper over xdebug, such as the code coverage in PHP unit [**?**], or phptracer [**?**].

I have decided to write yet another PHP analyzer, in OCaml, because I think OCaml is a better language to write programming language analysis tools (for bugs finding, refactoring assistance, type inference, compilers, IDEs, etc) and because I wanted to **integrate in a single infrastructure both static and dynamic analysis** so they can fruitfully be combined. Moreover, by also integrating information from version control systems, and databases (for instance to know the types of data in database schemas, or how developers are related to each other in a company), we can make certain analysis even more useful. To summarize:

static info + dynamic info + time info + developer info + db info = cool

1.2 Features

Here are the current analysis supported by pfff:

- Control-flow graph (CFG)
- TODO Data-flow analysis
- Callgraph (callers/callees)
- SEMI File/module dependencies
- SEMI Type annotations
- Scope annotations
- Code pattern matcher
- Dead code (functions, TODO classes, methods, statements)
- Software metrics such as CodeRank [**?**] or cyclomatic complexity [**?**].

Here are the tools currently built using the pfff infrastructure:

- `reaper`, a deadcode detector
- `sgrep_php`, a syntactical grep
- TODO An emacs mode with type inference feedback to developer, using both dynamic and static information (could also be used by a compiler to optimize things)
- Feedback from unit test to the developer about concrete values passed or returned by functions. It helps understanding one function.
- `pfff_browser`, a multi-purpose GUI including:
  - A semantic source code visualizer. It helps understanding one file or module.
  - A semantic architecture visualizer using Treemaps [**?**]. It helps understanding the organisation of the whole source code.
  - A code navigator with caller/callees, go to definition a la ctags (but more accurate).

1.3 Copyright

The source code of pfff is governed by the following copyright:

11 ⟨*Facebook copyright* 11⟩≡

```
(* Yoann Padioleau
 *
 * Copyright (C) 2009-2010 Facebook
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License (GPL)
 * version 2 as published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * file license.txt for more details.
 *)
```

1.4 Getting started

1.4.1 Requirements

Compiling the analysis part of pfff requires to install more libraries than what was required for the parsing part, which uses mostly the standard OCaml library. For instance, performing expensive global analysis on millions of lines of PHP code requires to store on the disk the result of intermediate computations and also to store its final result. The same is true for tools such as `ctags` which stores meta-data in `TAGS` files. In the case of pfff, the meta-data are stored in Berkeley DB [**?**] B-trees, and accessed through its OCaml binding (`ocamlbdb` by Tao Stein). In the same way, to use the code visualizer and navigator of pfff, you will need to install GTK.

In some cases the OCaml bindings to those libraries are directly included in the source of pfff ("batteries are included" in `external/`) as they don't require any configuration. In other cases you will have to install them by hand. So, in addition to `ocaml` and `make`[3], you will need to install the runtine and development libraries for:

- Berkeley DB 4.3 (its OCaml binding is in in `external/ocamlbdb`), for instance on CentOS 5.2 with:

  ```
  $ yum install db4-devel
  ```

- Perl posix regular expressions PCRE (its OCaml binding is in `external/ocamlpcre`), for instance with:

  ```
  $ yum install pcre-devel
  ```

- MySQL (its OCaml binding is in `external/ocamlmysql`), for instance with:

  ```
  $ yum install mysql-devel
  ```

- Ghostview and Graphviz if you want to visualize control flow graphs of PHP programs, for instance with:

  ```
  $ yum install gv
  $ yum install graphviz
  ```

- GTK and its LablGTK OCaml binding, for instance with:

[3] FACEBOOK: OCaml is also already installed in `/home/pad/packages/bin` so you just have to `source env.sh` from the pfff source directory. Note that it should work only for CentOS 5.2 machines, so ask ops to upgrade your dev machine

```
$ yum install gtk2-devel
$ yum install libgnomecanvas-devel
$ yum install libglade2-devel
$ yum install libart_lgpl-devel
```

[4]

- TODO The Thrift IDL

1.4.2 Compiling

The source code of pfff is available at http://padator.org/software/project-pfff/. [5] To compile pfff, see the instructions in `install.txt`. It should mainly consists in doing:

```
$ cd <pfff_src_directory>
$ ./configure --with-all
$ make depend
$ make
```

If you don't want the GUI, add `--no-gui` to the end of the `configure` command line (after `-with-all`). This reduced the number of dependencies and make it easier to compile pfff.

If you want to embed the analysis library in your own application, you will need to copy the `commons/`, `parsing_php/`, and `analyze_php/` directories as well as a few external dependencies (in `externals/`) in your own project directory, add a recursive make that goes in those directories, and then link your application with the different `.cma` library files (see also `pfff/demos/Makefile`).

It is also possible to use pfff services without writing OCaml code by:

- Interacting with its different command line tools, TODO as shown by the PHP emacs mode included in `emacs/`
- Using its JSON exporters and importers, SEMI as shown by the Python "binding" to pfff in `meta/python`
- TODO Calling pfff servers through Thrift

1.4.3 Quick example of use

TODO

[4] On CentOS and 64 machines, the `yum` package management tool is not very good and does not always install the 64 bit version of the libary. So even if you think the installation of automatic dependencies are OK, you may have to manually re-install certain dependencies again, for instance: `libgnomecanvas-devel`, `libglade2-devel`, `libart_lgpl-devel`

[5] FACEBOOK: The source of pfff are currently managed by git. to git it just do `git clone /home/engshare/git/projects/pfff`

1.4.4 The `pfff_db` command-line tool

The compilation process, in addition to building the `analyze_php.cma` library, also builds binary programs such as `pfff_db` that can let you evaluate among other things how good the pfff analysis are. For instance, to test pfff on the PhpBB (http://www.phpbb.com/, a popular internet forum package written in PHP) source code, just do:

```
$ cd /tmp
$ wget http://d10xg45o6p6dbl.cloudfront.net/projects/p/phpbb/phpBB-3.0.6.tar.bz2
$ tar xvfj phpBB-3.0.6.tar.bz2
$ cd <pfff_src_directory>
$ ./pfff_db -metapath /tmp/pfff_data    /tmpphpBB3/
```

The `pfff_db` program should then iterate over all PHP source code files (`.php` files), and run the parser on each of those files and index information from those files (the ASTs, caller, callees, types, etc) in `/tmp/pfff_data`. At the end, `pfff_db` will output some statistics showing what pfff was not able to handle.

One can then perform different analysis on the code, using the indexing information built in the previous stage. For instance to perform a deadcode analysis, do:

```
$ ./pfff_db -deadcode_detector   /tmp/pfff_data
```

1.4.5 The `pfff_browser` GUI

```
$ ./pfff_browser /tmp/pfff_data
```

Figure 1.1 shows a screenshot of the `pfff_browser` tool in action.

1.5 Source organization

Table 1.1 presents a short description of the modules in the `analyze_php/` directory of the pfff source distribution as well as the corresponding chapters the module is discussed.

Function	Chapter	Modules
Database entry point	??	`database_php.mli`

Table 1.1: Chapters and modules

1.6 API organization

Figure 1.2 presents the graph of dependencies between `.mli` files.

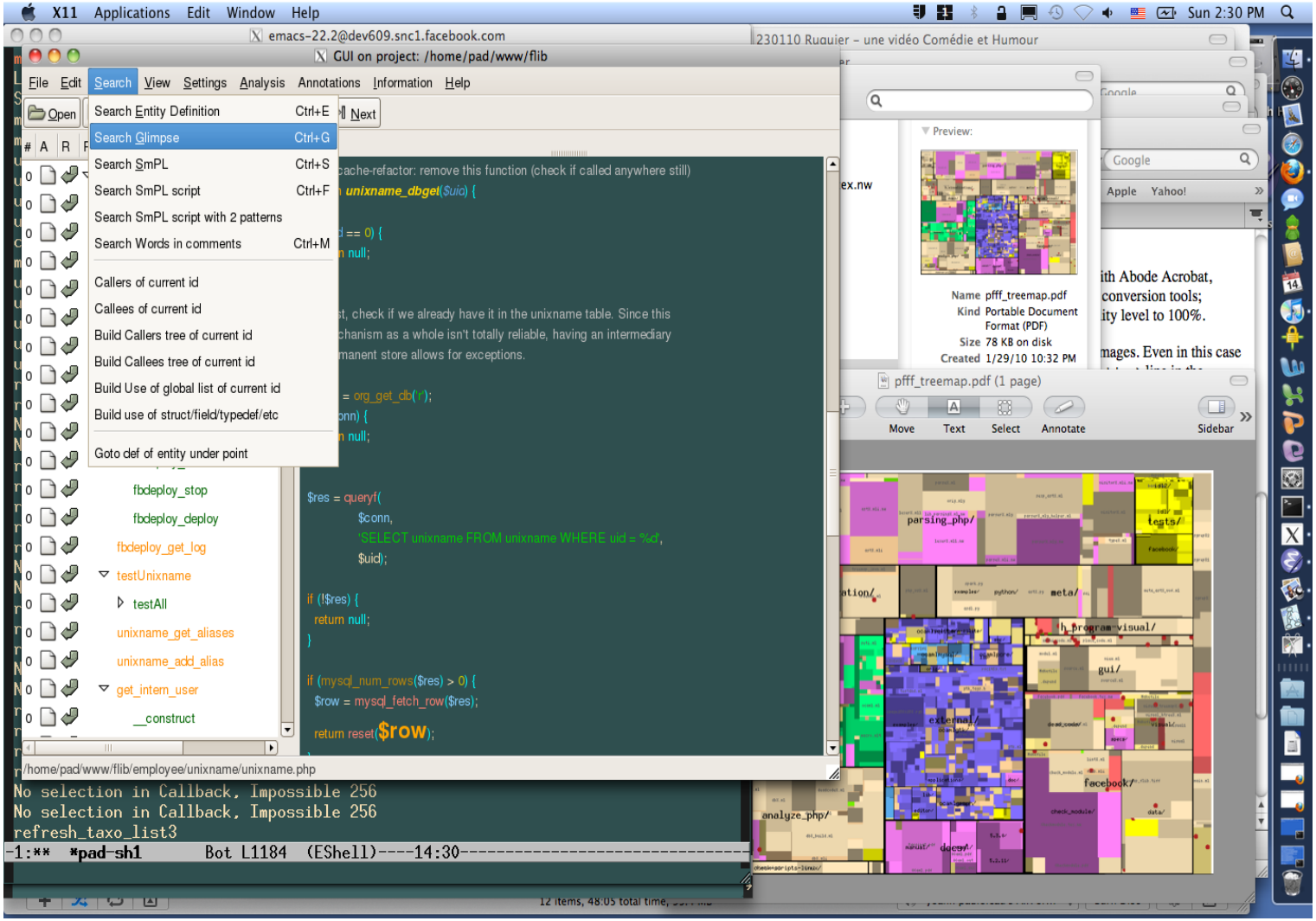

Figure 1.1: GUI browser screenshot

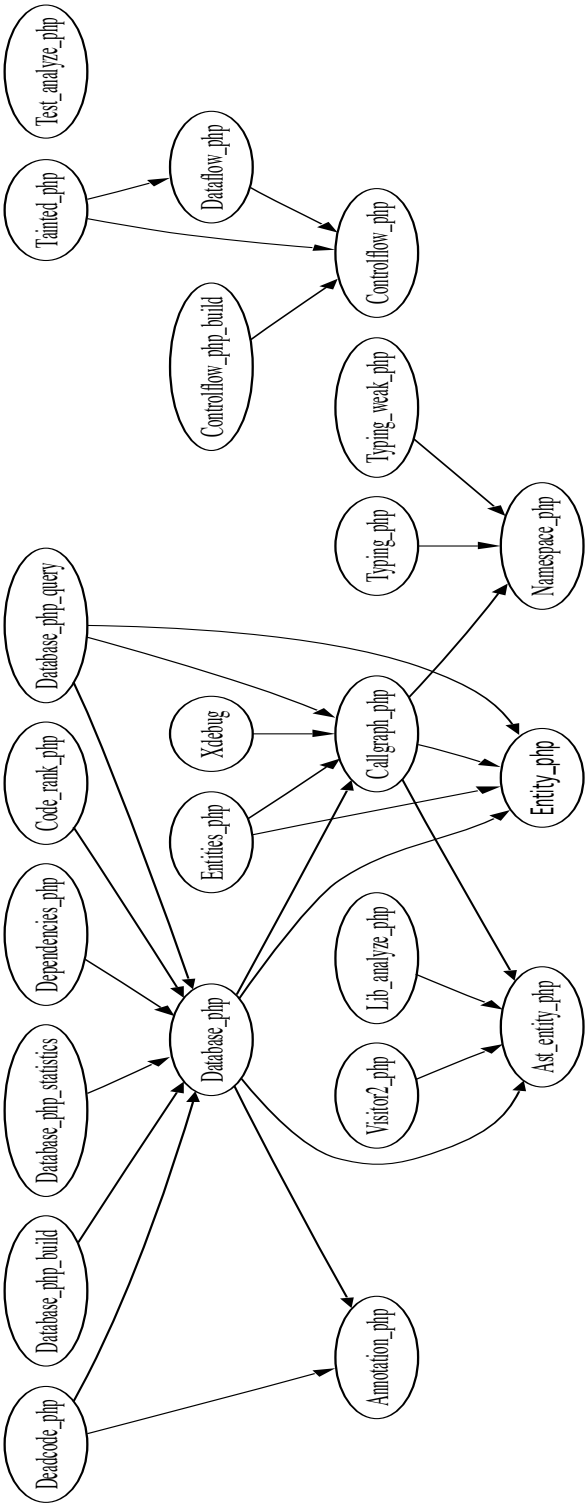

Figure 1.2: API dependency graph between `mli` files

1.7 Plan

Part 1 explains the interface of pfff, that is mainly the `.mli` files. Part 2 explains the code, the `.ml` files.

Note that the goal of this document is not to explain what is a control-flow graph or how a programming language frontend works, but just how to use the pfff API and how the pfff frontend is concretely implemented. We assume a basic knowledge of the literature on compilers such as [5] or [6].

1.8 About this document

This document is a literate program [1]. It is generated from a set of files that can be processed by tools (Noweb [2] and syncweb [3]) to generate either this manual or the actual source code of the program. So, the code and its documentation are strongly connected.

Part I

Using pfff

Chapter 2

Examples of Use

Chapter 3

Analysis Building Blocks

We now switch to a more systematic presentation of the pfff API, starting with the building blocks on which most subsequent analysis are built. We recommend two books to understand most of the terminology used in this manual:

- Modern Compiler Implementation in ML [6], by A. Appel
- Principles of Program Analysis [**?**], by F. Nielson, H.R. Nielson, and C. Hankin

3.1 Control flow graph

3.1.1 Introduction

Computing the control-flow graph (CFG) of a function is a necessary first step in many static analysis such as:

- Deadcode paths detection
- Data-flow fixpoint analysis (for class analysis, tainted analysis, type inference, etc)
- Certain software metrics such as the cyclomatic complexity of a function

Figure 3.1 presents the CFG of this simple PHP program:

20 ⟨*tests/cfg/while.php* 20⟩≡

```
<?php
function foo() {
  $x = 1;
  while($x < 5) {
    echo "$x\n";
    $x++;
  }
```

```
  $x;
}
foo();
```

3.1.2 Interface

The interface for managing control-flow graphs in pfff is splitted in two files, one containing mostly the types, in `controlflow_php.mli`, and one containing the function to build a CFG from a function or set of statements, in `controlflow_build_php.mli`.

Here is the overview of `controlflow_php.mli`:

21a ⟨*controlflow_php.mli* 21a⟩≡

```
open Ast_php
```

⟨*type node* 24a⟩
⟨*type node_kind* 24c⟩

⟨*type edge* 24b⟩

⟨*type flow* 23d⟩

⟨*controlflow helpers signatures* 26d⟩

⟨*function display_flow signature* 23a⟩

and here is the overview of `controlflow_build_php.mli`:

21b ⟨*controlflow_build_php.mli* 21b⟩≡

⟨*controlflow builders signatures* 21c⟩

⟨*controlflow checkers signatures* 26a⟩

⟨*type Controlflow_build_php.error* 26b⟩

⟨*error exception and report_error signature* 26c⟩

with the main entry points for the user being:

21c ⟨*controlflow builders signatures* 21c⟩≡

```
val control_flow_graph_of_stmts: Ast_php.stmt list -> Controlflow_php.flow

(* alias *)
val cfg_of_stmts: Ast_php.stmt list -> Controlflow_php.flow
```

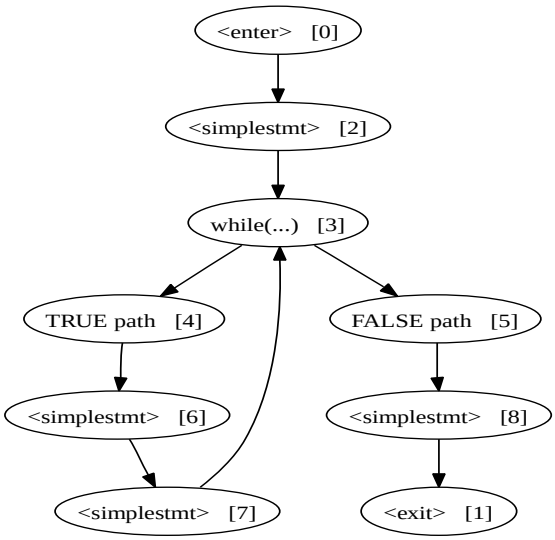

Figure 3.1: CFG of a toy program

```
val cfg_of_func:   Ast_php.func_def   -> Controlflow_php.flow
val cfg_of_method: Ast_php.method_def -> Controlflow_php.flow
```

Another useful function for debugging is `display_flow`:

23a ⟨*function display_flow signature* 23a⟩≡

```
(* using internally graphviz dot and ghostview on X11 *)
val display_flow: flow -> unit
```

3.1.3 pfff_db -cfg_php

You can easily test the CFG service of pfff by using the `-cfg_php` command line action as in:

```
$ ./pfff_db -cfg_php tests/cfg/while.php
```

which should launch ghostview to graphically display the different CFGs of the different functions in the `while.php` file.

23b ⟨*test_analyze_php actions* 23b⟩≡

```
    "-cfg_php",  " <file>",
    Common.mk_action_1_arg test_cfg_php;
```

23c ⟨*test_cfg_php* 23c⟩≡

```
let test_cfg_php file =
  let (ast2,_stat) = Parse_php.parse file in
  let ast = Parse_php.program_of_program2 ast2 in
  ast |> List.iter (function
  | Ast_php.FuncDef def ->
      (try
        let flow = Controlflow_build_php.cfg_of_func def in
        Controlflow_php.display_flow flow;
      with Controlflow_build_php.Error err ->
        Controlflow_build_php.report_error err
      )
  | _ -> ()
  )
```

3.1.4 The CFG type

23d ⟨*type flow* 23d⟩≡

```
type flow = (node, edge) Ograph_extended.ograph_mutable
```

24a ⟨*type node* 24a⟩≡

```
type node = {
  (* For now we just have node_kind, but later if we want to do some data-flow
   * analysis or use temporal logic, we may want to add extra information
   * in each CFG nodes. We could also record such extra
   * information in an external table that maps Ograph_extended.nodei,
   * that is nodeid, to some information.
   *)
  n: node_kind;
}
```

24b ⟨*type edge* 24b⟩≡

```
(* For now there is just one kind of edge. Later we may have more,
 * see the ShadowNode idea of Julia Lawall.
 *)
type edge = Direct
```

24c ⟨*type node_kind* 24c⟩≡

```
and node_kind =
⟨node_kind constructors 24d⟩
⟨node_kind aux types 25f⟩
```

24d ⟨*node_kind constructors* 24d⟩≡

```
  (* special fake cfg nodes *)
  | Enter
  | Exit
```

24e ⟨*node_kind constructors* 24d⟩+≡

```
  (* An alternative is to store such information in the edges, but
   * experience shows it's easier to encode it via regular nodes
   *)
  | TrueNode
  | FalseNode
```

24f ⟨*node_kind constructors* 24d⟩+≡

```
(* not used for now
  | BlockStart of tok (* { *)
  | BlockEnd of tok (* } *)
*)
```

24g ⟨*node_kind constructors* 24d⟩+≡

```
  (* TODO add appropriate info for each of those nodes *)
  | IfHeader
  (* not used for now
  | Else
  | Elsif
  *)
```

25a ⟨*node_kind constructors* 24d⟩+≡

```
| WhileHeader
| DoHeader
| DoWhileTail
| ForHeader
| ForeachHeader
```

25b ⟨*node_kind constructors* 24d⟩+≡

```
| SwitchHeader
| SwitchEnd
| Case
| Default
```

25c ⟨*node_kind constructors* 24d⟩+≡

```
| Return
```

25d ⟨*node_kind constructors* 24d⟩+≡

```
| Break
| Continue

| TryHeader
| CatchStart
| Catch
| TryEnd
| Throw
```

25e ⟨*node_kind constructors* 24d⟩+≡

```
| Join
(* statements without multiple outgoing or ingoing edges, such
 * as echo, expression statements, etc.
 *)
| SimpleStmt of simple_stmt
```

25f ⟨*node_kind aux types* 25f⟩≡

```
and simple_stmt =
    | TodoSimpleStmt
    (* TODO? expr includes Exit, Eval, Include, etc which
     * also have an influence on the control flow ...
     * We may want to uplift those constructors here and have
     * a better expr type
     *)
    (*
    | ExprStmt of expr * tok

    | EmptyStmt of expr * tok
```

```
  | Echo of tok * expr list * tok

  | Globals    of tok * global_var list * tok
  | StaticVars of tok * static_var list * tok

  | InlineHtml of string wrap

  | Use of tok * use_filename * tok
  | Unset of tok * variable list paren * tok
  | Declare of tok * declare list paren * colon_stmt
  *)
```

3.1.5 Other functions

26a ⟨*controlflow checkers signatures* 26a⟩≡

```
val deadcode_detection : Controlflow_php.flow -> unit
```

26b ⟨*type Controlflow_build_php.error* 26b⟩≡

```
type error =
  | DeadCode        of Ast_php.info
  | NoEnclosingLoop of Ast_php.info
  | ColonSyntax     of Ast_php.info
  | NoMethodBody    of Ast_php.info
  | DynamicBreak    of Ast_php.info
```

26c ⟨*error exception and report_error signature* 26c⟩≡

```
exception Error of error

val report_error : error -> unit
```

26d ⟨*controlflow helpers signatures* 26d⟩≡

```
val first_node : flow -> Ograph_extended.nodei
val mk_node: node_kind -> node
```

3.2 Data flow fixpoint

26e ⟨*dataflow_php.mli* 26e⟩≡

```
type 'a env =
  (Ast_php.dname, 'a) Common.assoc

type 'a inout = {
  in_env: 'a env;
  out_env: 'a env;
}
```

```
type 'a mapping =
  (Ograph_extended.nodei, 'a inout) Common.assoc


val fixpoint:
  Controlflow_php.flow ->
  initial:('a mapping) ->
  transfer:('a mapping list (* in edges *) -> Ograph_extended.nodei ->
            'a mapping) ->
  'a mapping



val display_dflow:
  Controlflow_php.flow -> 'a mapping -> ('a -> string) -> unit
```

3.3 Scope annotation

3.3.1 Local/Globals

27a ⟨*scoping_php.mli* 27a⟩≡

```
val globals_builtins : string list

(* !!Annotate via side effects!!. Fill in the scope information that
 * was put to None during parsing. I return a program, but really it
 * works by side effect.
 *)
val annotate_toplevel:
  Ast_php.toplevel -> Ast_php.toplevel
```

3.3.2 Free variables

27b ⟨*freevars_php.mli* 27b⟩≡

3.3.3 Namespace

27c ⟨*namespace_php.mli* 27c⟩≡

```
type nameS =
  | NameS of string
```

```
type dnameS =
  | DNameS of string (* without the dollar *)


val name_to_nameS_wrap: Ast_php.name -> nameS Ast_php.wrap

val dnameS_of_dname: Ast_php.dname -> dnameS

val nameS: nameS -> string
val dnameS: dnameS -> string
```

3.4 Types

28a ⟨*typing_trivial_php.mli* 28a⟩≡

```
val type_of_expr: Ast_php.expr -> Type_php.phptype
```

3.5 Xdebug traces

28b ⟨*xdebug.mli* 28b⟩≡

```
type call_trace = {
  f_call: Callgraph_php.kind_call;
  f_file: Common.filename;
  f_line: int;
  f_params: Ast_php.expr list;
  f_return: Ast_php.expr option;

  (* f_type: *)
}

val iter_dumpfile:
  (call_trace -> unit) -> Common.filename -> unit

val xdebug_main_name: string
```

3.6 Comment extraction

28c ⟨*comment_annotater_php.mli* 28c⟩≡

28d ⟨*info_annotater_php.mli* 28d⟩≡

3.7 Tag extraction

29a ⟨*annotation_php.mli* 29a⟩≡

```
type annotation =
  | CalledFromPhpsh
  | CalledOutsideTfb
  | NotDeadCode

  | Have_THIS_FUNCTION_EXPIRES_ON
  | Other of string

val extract_annotations: string -> annotation list
```

3.8 AST simplication

29b ⟨*normalize_php.mli* 29b⟩≡

```
(*
val normalize:
  Ast_php.toplevel list -> Ast_php.toplevel list
*)
```

3.9 Visitors

29c ⟨*visitor2_php.mli* 29c⟩≡

```
open Ast_php

type visitor_out = {
  vorigin: Visitor_php.visitor_out;
  vid_ast: Ast_entity_php.id_ast -> unit;
}

val default_visitor : Visitor_php.visitor_in

val mk_visitor: Visitor_php.visitor_in -> visitor_out

val do_visit_with_ref:
  ('a list ref -> Visitor_php.visitor_in) -> (visitor_out -> unit) -> 'a list
```

Chapter 4

The Code Database

4.1 Indexing the code

30 ⟨*database_php.mli* 30⟩≡

```
type project =
  Project of Common.dirname * string option (* name of project *)

type id     = Entity_php.id
type fullid = Entity_php.fullid
type id_kind   = Entity_php.id_kind
type id_string = string

(* can be useful in gui to highlight part of text *)
type range_pos = int * int (* charpos x charpos *)

type db_support =
  | Disk of Common.dirname (* the path to the Berkeley DB meta data *)
  | Mem

type database = {
  project: project;
  metapath: Common.dirname;

  mutable next_free_id: int;

  (* opti: *)
  fullid_of_id: (id, fullid) Oassoc.oassoc;
  id_of_fullid: (fullid, id) Oassoc.oassoc;

  file_to_topids: (Common.filename, id list) Oassoc.oassoc;
```

```
  (* return Not_found for toplevel ids *)
  enclosing_id: (id, id) Oassoc.oassoc;
  children_ids: (id, id list) Oassoc.oassoc;

  defs: database_defs;
  uses: database_uses;

  (* symbols, to build completion list in gui for instance (faster than defs) *)
  symbols: (string, unit) Oassoc.oassoc;

  strings: (string, unit) Oassoc.oassoc;

  (* IO *)
  flush_db: unit -> unit;
  close_hook: unit -> unit;
}
 and database_defs = {
   (* asts *)
   toplevels:   (id, Ast_php.toplevel) Oassoc.oassoc;

   (* NEVER USE THIS FIELD directly. Use the ast_of_id helper function! *)
   asts: (id, Ast_entity_php.id_ast) Oassoc.oassoc;

   (* consider also using toks_of_topid_of_id wrapper func *)
   str_of_topid:    (id, string)         Oassoc.oassoc;
   tokens_of_topid: (id, Parser_php.token list) Oassoc.oassoc;
   range_of_topid: (id, (Common.parse_info * Common.parse_info)) Oassoc.oassoc;

   (* Not all ids have a name; for instance StmtList ASTs
    * really dont have one (well for now I gave them a __TOPSTMT__ name).
    * So names is a partial assoc. Same for kinds.
    *
    * for .names, see also name_of_id helper.
    *)
   name_defs: (id_string, id list) Oassoc.oassoc;
   id_kind: (id, id_kind)   Oassoc.oassoc;

   (* computer statically or dynamically *)
   id_type: (id, Type_php.phptype) Oassoc.oassoc;

   (* shortcut, to avoid getting the ast to get the name of the entity *)
   names: (id, id_string) Oassoc.oassoc;

   extra: (id, extra_id_info) Oassoc.oassoc;
 }
```

```
   and database_uses = {
    callees_of_f: (id, Callgraph_php.callsites_opt list) Oassoc.oassoc;
    callers_of_f: (id, Callgraph_php.callersinfo_opt list) Oassoc.oassoc;
   }

  and extra_id_info = {
    tags: Annotation_php.annotation list;
    partial_callers: bool;
    partial_callees: bool;
    todo: unit;
  }

type error =
  | ErrorDb of string
val report_error: error -> string

exception Error of error


exception DatabaseAlreadyLocked

(* note: create_db is in database_c_build.mli *)

(* @Effect: acquire and release file lock *)
val open_db : metapath:Common.dirname -> database
val close_db: database -> unit
val with_db:  metapath:Common.dirname -> (database -> unit) -> unit

val check_db: database -> unit

(* ------------------------------------------------------------------------ *)
val name_of_id:  id -> database -> string
val ast_of_id:   id -> database -> Ast_entity_php.id_ast

val is_top_id: id -> database -> bool
val toks_of_topid_of_id: id -> database -> Parser_php.token list

val filename_of_id: id -> database -> Common.filename
val filename_in_project_of_id: id -> database -> Common.filename

(* for debugging *)
val str_of_id:      id -> database -> string

val ids_with_kind__of_string: string -> database -> (id * id_kind) list

(* entity accessor *)
```

```
val filter_ids_of_string: string -> id_kind -> database -> id list

val function_ids__of_string: string -> database -> id list
val method_ids_of_string:    string -> database -> id list
val class_ids_of_string:     string -> database -> id list

(* get all entities *)
val filter_ids_in_db: id_kind -> database -> (string, id list) Common.assoc

val functions_in_db: database -> (string, id list) Common.assoc
val classes_in_db: database -> (string, id list) Common.assoc
val methods_in_db: database -> (string, id list) Common.assoc

val is_function_id: id -> database -> bool

val id_of_function: string -> database -> id

val id_of_kind_call:
  ?file_disambiguator:Common.filename ->
  Callgraph_php.kind_call -> database -> id

(* works for function and methods *)
val callees_of_id: id -> database -> Callgraph_php.callsite list
val callers_of_id: id -> database -> Callgraph_php.callerinfo list

(* ------------------------------------------------------------------------ *)
val path_of_project:             project -> Common.dirname
val glimpse_metapath_of_database: database -> Common.dirname
val normalize_project: project -> project
val default_metapath_of_project:         project -> Common.dirname
val database_tag_filename: string

val default_extra_id_info : extra_id_info

(* ------------------------------------------------------------------------ *)
val actions: unit -> Common.cmdline_actions
```

33 ⟨*database_php_build.mli* 33⟩≡

```
(* main entry point, does the whole job *)
val create_db :
  ?metapath:Common.dirname ->
  ?phase:int ->
  ?filter_files_hook:(Common.filename list -> Common.filename list) ->
  ?use_glimpse:bool ->
  Database_php.project ->
```

```
  Database_php.database

val actions: unit -> Common.cmdline_actions

val max_phase: int

(* helpers used internally that can be useful to other *)
val iter_files_and_topids :
  Database_php.database -> string ->
  (Database_php.id -> Common.filename -> unit) ->
  unit

val iter_files_and_ids :
  Database_php.database -> string ->
  (Database_php.id -> Common.filename -> unit) ->
  unit
```

4.2 Entities, the database primary keys

34 ⟨*entity_php.mli* 34⟩≡

```
type id = Id of int

type fullid = filepos
 and filepos = {
  file: Common.filename;
  line: int;
  column: int;
}

type id_kind =
  (* toplevels, which can also be nested *)
  | Function
  | Class
  | Interface
  | StmtList

  (* only at nested level, inside a class *)
  | Method
  | ClassConstants
  | ClassVariables

  | IdMisc
```

```
val str_of_id: id -> string
val str_of_fullid: fullid -> string

val fullid_regexp: string
val fullid_of_string: string -> fullid

val filepos_of_parse_info: Common.parse_info -> filepos

val string_of_id_kind: id_kind -> string
```

35a ⟨*ast_entity_php.mli* 35a⟩≡

4.3 Defs

4.4 Uses

4.5 Misc

35b ⟨*entities_php.mli* 35b⟩≡

```
type idtree = Callgraph_php.idtree

val tree_of_ids: ?sort:bool ->
  Entity_php.id list -> (Entity_php.id -> string) -> idtree

val first_id_in_tree: idtree -> Entity_php.id
```

Chapter 5

Global Analysis

5.1 Call graph

36 ⟨*callgraph_php.mli* 36⟩≡

```
type kind_call =
    | FunCall of string
    | ObjectCall of string (* class *) * string (* method *)
    | ClassCall of string (* module *) * string

type call = Call of Entity_php.id (* from *) *
                    Entity_php.id (* to *) *
                    kind_call_and_pos
   and kind_call_and_pos =
     | Direct     of Namespace_php.nameS Ast_php.wrap
     | MethodCall of Namespace_php.nameS Ast_php.wrap * call_extra_info
     | IndirectTodo (* of Ast.expression * call_extra_info *)

     and call_extra_info = {
       todo: unit;
     }

(* specialized types used in database, to avoid storing each time one
 * of the id as this id will already be the key of the table.
 * For function call it also factorize things. We define two types
 * because depending on the direction we can or not factorize things.
 *)
type callsites_opt =
  | DirectCallToOpt of
      Namespace_php.nameS * Ast_php.info list (* instances *) *
        Entity_php.id list (* candidates *)
```

```
  (* for now we dont factorize method calls. could *)
  | MethodCallToOpt of
      Entity_php.id * Namespace_php.nameS Ast_php.wrap * call_extra_info
  | IndirectFuncPtCallToOptTodo
      (* of Entity.id * Ast.expression * call_extra_info *)

type callersinfo_opt =
  | DirectCallerIsOpt of
      Entity_php.id * int (* nb instances *)
  | MethodCallerIsOpt of
      Entity_php.id * Namespace_php.nameS Ast_php.wrap * call_extra_info
  | IndirectFuncPtCallerIsOptTodo
      (* of Entity.id * Ast.expression * call_extra_info *)


(* types used mostly by callees_of_id and callers_of_id wrappers in
 * database_php.ml *)
type callsite =
  CallSite of Entity_php.id * kind_call_and_pos
type callerinfo =
  CallerInfo of Entity_php.id * kind_call_and_pos

(* ------------------------------------------------------------------------ *)
val default_call_extra_info : call_extra_info

type analysis_confidence = int
val no_info_confidence: analysis_confidence

type node = {
  name: string;
  id: Entity_php.id; (* can be interpreted as the caller or callee *)
  extra: call option;
  confidence: analysis_confidence;
  gray: bool;
}

type idtree = node Common.treeref
type calltree = node Common.treeref


(* as argument for the building functions *)
type calltree_preferences = {
  squeeze_duplicate: bool; (* when want abstract about each instance call *)
  squeeze_duplicate_graph: bool;
  filter_id: Entity_php.id -> bool; (* for instance to filter non-x86 entities *)
```

```
  filter_confidence: analysis_confidence -> bool;
  put_at_end_filtered_and_gray_ize: bool;
}
val default_calltree_preferences: calltree_preferences

(* ------------------------------------------------------------------------ *)
val s_of_kind_call: kind_call -> string

(* ------------------------------------------------------------------------ *)
val callerinfo_to_call: callerinfo -> Entity_php.id -> call
val callsite_to_call: Entity_php.id -> callsite -> call

val callsites_opt_to_callsites:
  callsites_opt -> callsite list
val callersinfo_opt_to_callersinfo:
  callees_of_id:(Entity_php.id -> callsites_opt list) ->
  Entity_php.id ->
  callersinfo_opt -> callerinfo list

val id_of_callerinfo:  callerinfo -> Entity_php.id
val id_of_callsite:    callsite   -> Entity_php.id

(* ------------------------------------------------------------------------ *)

val calltree_callers_of_f:
  Entity_php.id ->
  depth:int ->
  parent_and_extra_opt:
    (Entity_php.id * kind_call_and_pos * analysis_confidence * bool) option ->
  namefunc:(Entity_php.id -> string) ->
  callersfunc:(Entity_php.id -> callerinfo list) ->
  fullid_info:(Entity_php.id -> Entity_php.fullid) ->
  preferences:calltree_preferences ->
  calltree

val calltree_callees_of_f:
  Entity_php.id ->
  depth:int ->
  parent_and_extra_opt:
    (Entity_php.id * kind_call_and_pos * analysis_confidence * bool) option ->
  namefunc:(Entity_php.id -> string) ->
  calleesfunc:(Entity_php.id -> callsite list) ->
  fullid_info:(Entity_php.id -> Entity_php.fullid) ->
  preferences:calltree_preferences ->
  calltree
```

```
(* ------------------------------------------------------------------------ *)
(* The function here returns only local information. For a full
 * caller/callee analysis then need to use this local information and
 * perform a global analysis. cf database.ml and functions below.
 *)
val callees_of_ast:
  Ast_entity_php.id_ast -> Namespace_php.nameS Ast_php.wrap list

val methodcallees_of_ast:
  Ast_entity_php.id_ast -> Namespace_php.nameS Ast_php.wrap list
```

5.2 Include graph

39a ⟨*dependencies_php.mli* 39a⟩≡

```
val dir_to_dir_dependencies:
  Database_php.database -> unit
```

39b ⟨*include_require_php.mli* 39b⟩≡

5.3 Bottom-up analysis

39c ⟨*graph_php.mli* 39c⟩≡

39d ⟨*bottomup_analysis_php.mli* 39d⟩≡

5.4 Handling built-in functions

39e ⟨*builtins_php.mli* 39e⟩≡

```
type idl_type =
  | Boolean
  | Byte
  | Int16
  | Int32
  | Int64
  | Double
  | String
  | Int64Vec
  | StringVec
```

```
  | VariantVec
  | Int64Map
  | StringMap
  | VariantMap
  | Object
  | Resource
  | Variant
  | Numeric
  | Primitive
  | PlusOperand
  | Sequence
  | Any
  | NULL
  | Void


type idl_param = {
  p_name: string;
  p_type: idl_type;
  p_isref: bool;
  p_default_val: string option;
}


type idl_entry =
  | Global of string * idl_type
  | Function of
      string * idl_type * idl_param list * bool (* has variable arguments *)


val ast_php_to_idl:
  Ast_php.program -> idl_entry list

val idl_entry_to_php_fake_code:
  idl_entry -> string


val generate_php_stdlib:
  Common.dirname (* dir with .idl.php files *) ->
  Common.dirname (* dest, e.g. ~/www/php_stdlib/ *) ->
  unit

val actions: unit -> Common.cmdline_actions
```

5.5 Call graph for dynamic calls, function aliasing

41 ⟨*aliasing_function_php.mli* 41⟩≡

```
val finding_function_pointer_prefix:
  string -> Ast_php.toplevel -> string list
```

5.6 Call graph for method calls, object aliasing

Chapter 6

Static Analysis

42a ⟨*analysis_static_php.mli* 42a⟩≡

6.1 Dead code

42b ⟨*deadcode_php.mli* 42b⟩≡

```
type hooks = {
  (* to remove certain false positives *)
  is_probable_dynamic_funcname: string -> bool;

  (* to avoid generating patches for code which does not have a valid
   * git owner anymore
   *)
  is_valid_author: string -> bool;

  (* to avoid generating patches for certain files, such as code in
   * third party libraries or auto generated code
   *)
  is_valid_file: Common.filename -> bool;

  false_positive_deadcode_annotations: Annotation_php.annotation list;

  (* config *)
  print_diff: bool;
  with_blame: bool;
  cache_git_blame: bool;
  (* place where we would put the generated patches *)
  patches_path: Common.dirname;
}
val default_hooks: hooks
```

```
type deadcode_patch_info = {
  file     : Common.filename; (* relative to the project *)
  reviewer : string option;
  cc       : string option;
  date     : Common.date_dmy;
}


(* main entry point. Will generate data in hooks.patches_path *)
val deadcode_analysis: hooks -> Database_php.database -> unit



(* deadcode_analysis may generate some .git_annot files in your
 * source directory *)
val cleanup_cache_files: Common.dirname -> unit

(* path where resides all deacode patches *)
val deadcode_stat: Common.dirname -> unit

val deadcode_patch_info: Common.filename -> deadcode_patch_info


(* internal analysis functions *)
val finding_dead_functions:
  hooks -> Database_php.database -> (string * Database_php.id) list

type dead_ids_by_file =
  Common.filename * (string * Database_php.fullid * Database_php.id) list

val deadcode_fixpoint_per_file:
 dead_ids_by_file list ->
  Database_php.id list (* original set of dead ids *) ->
 hooks -> Database_php.database ->
 dead_ids_by_file list
```

6.2 Type inference

43a ⟨*type_annotater_php.mli* 43a⟩≡

43b ⟨*typing_php.mli* 43b⟩≡

```
(* cf also namespace.ml *)

type environment = (Namespace_php.dnameS, Type_php.phptype) Hashtbl.t


(* !!Annotate via side effects!!. Fill in the type information that
 * was put to None during parsing. I return a program, but really it
 * works by side effect.
 *)
val annotate_toplevel:
  environment ref -> Ast_php.toplevel -> Ast_php.toplevel
```

44a ⟨*typing_weak_php.mli* 44a⟩≡

```
val extract_fields_per_var:
  Ast_php.toplevel -> (Namespace_php.dnameS * string list) list
```

6.3 Tainted analysis

44b ⟨*tainted_php.mli* 44b⟩≡

```
type tainted = bool

val tainted_analysis:
  Controlflow_php.flow -> tainted Dataflow_php.mapping

val check_bad_echo:
  Controlflow_php.flow -> tainted Dataflow_php.mapping -> unit

val display_tainted_flow:
  Controlflow_php.flow -> tainted Dataflow_php.mapping -> unit
```

6.4 Static test coverage

Chapter 7

Dynamic Analysis

45a ⟨*analysis_dynamic_php.mli* 45a⟩≡

7.1 Type extractions

7.2 Value extractions

7.3 PHPUnit

45b ⟨*phpunit.mli* 45b⟩≡

7.4 Test coverage

Chapter 8

Finding Code

8.1 Finding files

46a ⟨*lib_analyze_php.mli* 46a⟩≡

```
val find_php_files: Common.dirname -> Common.filename list

val ii_of_id_ast: Ast_entity_php.id_ast -> Ast_php.info list
```

46b ⟨*database_php_query.mli* 46b⟩≡

```
open Database_php

(*
val get_functions_ids__of_string:
  string -> database -> id list
*)

(* general queries, get files or ids *)
val glimpse_get_matching_files:
  string -> database -> Common.filename list

(* ------------------------------------------------------------------------ *)
(* wrappers around callgraph functions *)
val calltree_callers_of_f:
  depth:int -> preferences:Callgraph_php.calltree_preferences ->
  Entity_php.id -> Database_php.database ->
  Callgraph_php.calltree
val calltree_callees_of_f:
  depth:int -> preferences:Callgraph_php.calltree_preferences ->
  Entity_php.id -> Database_php.database ->
  Callgraph_php.calltree
```

```
(* ------------------------------------------------------------------------ *)
val actions: unit -> Common.cmdline_actions
```

8.2 Finding functions

8.3 Finding expression positions

47a ⟨*finder_php.mli* 47a⟩≡

```
(* position based finder *)
val info_at_pos: int -> Ast_php.toplevel -> Ast_php.info
val expr_at_pos: int -> Ast_php.toplevel -> Ast_php.expr

val info_at_pos_in_full_program: int -> Ast_php.program -> Ast_php.info
```

8.4 sgrep_php, syntactical grep

47b ⟨*smpl_php.mli* 47b⟩≡

Chapter 9

Checking Code

48 ⟨*checking_php.mli* 48⟩≡

```
val check_program: Ast_php.program -> unit

type error =
  | TooManyArguments of (Common.parse_info * Ast_php.name (* def *))
  | NotEnoughArguments of (Common.parse_info * Ast_php.name (* def *))
exception Error of error

val report_error : error -> unit
```

9.1 Statistical bug finding

Chapter 10

Transforming Code

Chapter 11

Software Metrics

11.1 Basic metrics

50a ⟨*statistics_php.mli* 50a⟩≡

11.2 Code rank

50b ⟨*code_rank_php.mli* 50b⟩≡

```
type code_ranks = {
  function_ranks: (Database_php.id, float) Oassoc.oassoc;
}


val build_code_ranks:
  Database_php.database -> code_ranks

val build_naive_caller_ranks:
  Database_php.database -> code_ranks
```

11.3 Cyclomatic complexity

11.3.1 Interface

50c ⟨*cyclomatic_php.mli* 50c⟩≡

```
val cyclomatic_complexity_func:
  ?verbose:bool ->
  Ast_php.func_def -> int

val cyclomatic_complexity_method:
```

```
    ?verbose:bool ->
    Ast_php.method_def -> int

  (* internal *)
  val cyclomatic_complexity_flow:
    ?verbose:bool ->
    Controlflow_php.flow -> int
```

11.3.2 pfff_db -cyclomatic_php

51a ⟨*test_analyze_php actions* 23b⟩+≡

```
    "-cyclomatic_php", " <file>",
    Common.mk_action_1_arg test_cyclomatic_php;
```

51b ⟨*test_cyclomatic_php* 51b⟩≡

```
  let test_cyclomatic_php file =
    let (ast2,_stat) = Parse_php.parse file in
    let ast = Parse_php.program_of_program2 ast2 in
    ast |> List.iter (function
    | Ast_php.FuncDef def ->
        let name = Ast_php.name def.Ast_php.f_name in
        let n = Cyclomatic_php.cyclomatic_complexity_func ~verbose:true def in
        pr2 (spf "cyclomatic complexity for function %s is %d" name n);
    | Ast_php.ClassDef def ->
        let class_stmts = Ast_php.unbrace def.Ast_php.c_body in
        let class_name = Ast_php.name def.Ast_php.c_name in
        class_stmts |> List.iter (function
        | Ast_php.Method def ->
            let method_name = Ast_php.name def.Ast_php.m_name in
            let n = Cyclomatic_php.cyclomatic_complexity_method ~verbose:true def
            in
            pr2 (spf "cyclomatic complexity for method %s::%s is %d"
                    class_name method_name n);
        | Ast_php.ClassConstants _ | Ast_php.ClassVariables _ ->
            ()
        )
    | _ -> ()
    )
```

Chapter 12

Other Services

12.1 Statistics

52a ⟨*database_php_statistics.mli* 52a⟩≡

```
val parsing_stat_db:   Database_php.database -> unit
val typing_stat_db:    Database_php.database -> unit
(*
val callgraph_stat_db: Database_php.database -> unit
val extra_stat_db:     Database_php.database -> unit
*)
val fields_stat_db: Database_php.database -> unit

val all_stat_db:      Database_php.database -> unit

val actions: unit -> Common.cmdline_actions
```

12.2 Testing

52b ⟨*test_analyze_php.mli* 52b⟩≡

```
val common_parse_and_type: Common.filename -> Ast_php.program

val actions: unit -> Common.cmdline_actions
```

12.3 Debugging

Part II

Internals

Chapter 13

Implementation overview

Figure 13.1 presents the graph of dependencies between `.ml` files.

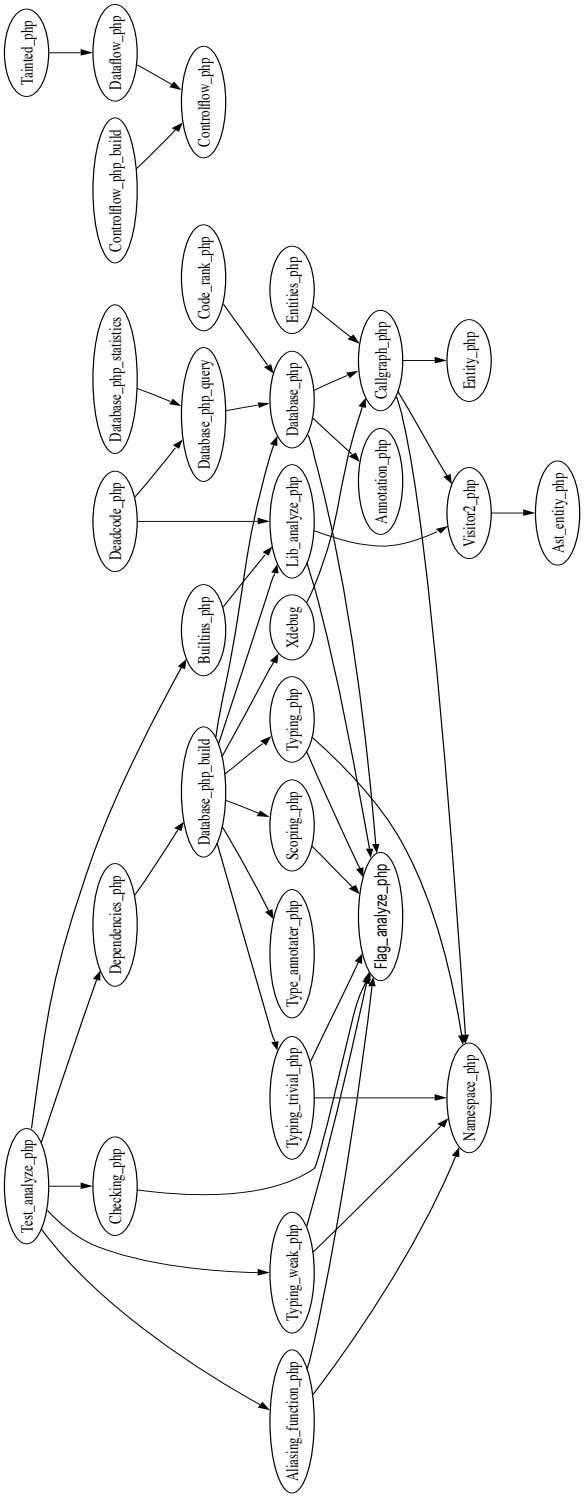

Figure 13.1: Dependency graph between `ml` files

Chapter 14

Control Flow Graph

14.1 Overview

Here is the overview of `controlflow_php.ml`:

56 ⟨*controlflow_php.ml* 56⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

open Ast_php

module Ast = Ast_php

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*****************************************************************************)
(* Types *)
(*****************************************************************************)
```

⟨*type node* 24a⟩
⟨*type node_kind* 24c⟩

⟨*type edge* 24b⟩

⟨*type flow* 23d⟩

```
(*****************************************************************************)
(* String of *)
(*****************************************************************************)
```

⟨*function short_string_of_node* 71a⟩

```
(*****************************************************************************)
(* Accessors *)
(*****************************************************************************)
```

⟨*controlflow_php accessors* 73d⟩

⟨*function display_flow* 70⟩

and here is the overview of `controlflow_build_php.ml`:

57 ⟨*controlflow_build_php.ml* 57⟩≡

⟨*Facebook copyright* 11⟩

```
open Common

open Ast_php

module Ast = Ast_php
module F = Controlflow_php

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*****************************************************************************)
(* Types *)
(*****************************************************************************)
```

⟨*type nodei* 58a⟩

⟨*type state* 58b⟩

⟨*type Controlflow_build_php.error* 26b⟩

```
exception Error of error

(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)
```

⟨*controlflow_php helpers* 71b⟩

```
(*****************************************************************************)
(* Algorithm *)
```

```
(*****************************************************************************)
```

⟨*controlflow_php main algorithm* 60⟩

```
(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)
```

⟨*controlflow builders* 59a⟩

```
(*****************************************************************************)
(* Deadcode stmts detection. See also deadcode_php.ml *)
(*****************************************************************************)
```

⟨*function deadcode_detection* 73b⟩

```
(*****************************************************************************)
(* Error management *)
(*****************************************************************************)
```

⟨*function Controlflow_build_php.report_error* 73c⟩

14.2 Types

The CFG types have already been described in Section 3.1.4. Here we present additional types that are used only internally by the CFG builder algorithm.

58a ⟨*type nodei* 58a⟩≡

```
(* an int representing the index of a node in the graph *)
type nodei = Ograph_extended.nodei
```

58b ⟨*type state* 58b⟩≡

```
(* Information passed recursively in cfg_stmt or cfg_stmt_list below.
 * The graph g is mutable, so most of the work is done by side effects on it.
 * No need to return a new state.
 *)
type state = {
  g: F.flow;

  (* When there is a 'return' we need to know the exit node to link to *)
  exiti: nodei;

  (* Sometimes when there is a 'continue' or 'break' we must know where
   * to jump and so we must know the node index for the end of the loop.
   * The same kind of information is needed for 'switch' or 'try/throw'.
```

```
   *
   * Because loops can be inside switch or try, and vice versa, you need
   * a stack of context.
   *)
  ctx: context Common.stack;
}
 and context =
  | NoCtx
  | LoopCtx   of nodei (* head *) * nodei (* end *)
  | SwitchCtx of nodei (* end *)
  | TryCtx    of nodei (* the first catch *)
```

14.3 Main entry point

59a ⟨*controlflow builders* 59a⟩≡

```
let (control_flow_graph_of_stmts: stmt list -> F.flow) = fun xs ->

  (* yes, I sometimes use objects, and even mutable objects in OCaml ... *)
  let g = new Ograph_extended.ograph_mutable in

  let enteri = g#add_node { F.n = F.Enter; } in
  let exiti = g#add_node { F.n = F.Exit; } in

  let state = {
    g = g;
    exiti = exiti;
    ctx = [NoCtx]; (* could also remove NoCtx and use an empty list *)
  }
  in
  let last_node_opt =
    cfg_stmt_list state (Some enteri) xs
  in
  (* maybe the body does not contain a single 'return', so by default
   * connect last stmt to the exit node
   *)
  g |> add_arc_opt (last_node_opt, exiti);
  g
```

59b ⟨*controlflow builders* 59a⟩+≡

```
let (cfg_of_func: func_def -> F.flow) = fun def ->
  let stmts = stmts_of_stmt_or_defs (Ast.unbrace def.f_body) in
  (* todo? could create a node with function name ? *)
  control_flow_graph_of_stmts stmts
```

```
let (cfg_of_method: method_def -> F.flow) = fun def ->
  match def.m_body with
  | AbstractMethod _ ->
      raise (Error (NoMethodBody (def.m_tok)))
  | MethodBody body ->
      let stmts = stmts_of_stmt_or_defs (Ast.unbrace body) in
      (* todo? could create a node with method name ? *)
      control_flow_graph_of_stmts stmts
```

14.4 Algorithm

60 ⟨*controlflow_php main algorithm* 60⟩≡

```
(*
 * The CFG building algorithm works by iteratively visiting the
 * statements in the AST of a function. At each statement,
 * the cfg_stmt function is called, and passed the index of the
 * previous node (if there is one), and returns the index of
 * the created node (if there is one).
 *
 * history:
 *
 * ver1: old code was returning a nodei, but break has no end, so
 * cfg_stmt should return a nodei option.
 *
 * ver2: old code was taking a nodei, but should also take a nodei
 * option. There can be deadcode in the function.
 *
 * subtle: try/throw. The current algo is not very precise, but
 * it's probably good enough for many analysis.
 *)
let rec (cfg_stmt: state -> nodei option -> stmt -> nodei option) =
 fun state previ stmt ->

   match stmt with
   | ExprStmt _
   | EmptyStmt _
   | Echo (_, _, _)
   | InlineHtml _
      ->
       let simple_stmt = F.TodoSimpleStmt in
       let newi = state.g#add_node { F.n = F.SimpleStmt simple_stmt; } in
       state.g |> add_arc_opt (previ, newi);
       Some newi

   | Block xs ->
```

```
    let stmts = stmts_of_stmt_or_defs (Ast.unbrace xs) in
    cfg_stmt_list state previ stmts

| For _ | Foreach _  | While _ ->
  (* previ -> newi ---> newfakethen -> ... -> finalthen -
   *             |---|-----------------------------------|
   *                 |-> newfakelse
   *)
    let node, colon_stmt =
      (match stmt with
      | While (t1, e, colon_stmt) ->
          F.WhileHeader, colon_stmt
      | For (t1, t2, e1, t3, e2, t4, e5, t6, colon_stmt) ->
          F.ForHeader, colon_stmt
      | Foreach (t1, t2, e1, t3, v, arrow_opt, t4, colon_stmt) ->
          F.ForeachHeader, colon_stmt
      | _ -> raise Impossible
      )
    in

    let newi = state.g#add_node { F.n = node } in
    state.g |> add_arc_opt (previ, newi);

    let newfakethen = state.g#add_node { F.n = F.TrueNode; } in
    let newfakeelse = state.g#add_node { F.n = F.FalseNode; } in
    state.g |> add_arc (newi, newfakethen);
    state.g |> add_arc (newi, newfakeelse);

    let state = { state with
      ctx = LoopCtx (newi, newfakeelse)::state.ctx;
    }
    in
    let finalthen =
      cfg_colon_stmt state (Some newfakethen) colon_stmt
    in
    state.g |> add_arc_opt (finalthen, newi);
    Some newfakeelse

(* This time, may return None, for instance if return in body of dowhile
 * (whereas While cant return None). But if return None, certainly
 * sign of buggy code.
 *)
| Do (t1, st, t2, e, t3) ->
  (* previ -> doi ---> ... ---> finalthen (opt) ---> taili
   *             |--------- newfakethen ---------------|  |---> newfakelse
   *)
```

```
    let doi = state.g#add_node { F.n = F.DoHeader; } in
    state.g |> add_arc_opt (previ, doi);
    let taili = state.g#add_node { F.n = F.DoWhileTail; } in

    let newfakethen = state.g#add_node { F.n = F.TrueNode; } in
    let newfakeelse = state.g#add_node { F.n = F.FalseNode; } in
    state.g |> add_arc (taili, newfakethen);
    state.g |> add_arc (taili, newfakeelse);
    state.g |> add_arc (newfakethen, doi);

    let state = { state with
      ctx = LoopCtx (taili, newfakeelse)::state.ctx;
    }
    in
    let finalthen =
      cfg_stmt state (Some doi) st
    in
    (match finalthen with
    | None ->
        (* weird, probably wrong code *)
        None
    | Some finalthen ->
        state.g |> add_arc (finalthen, taili);
        Some newfakeelse
    )


| IfColon (_t1, _e, tok, _st, _elseifs, _else, _t2, _t3)  ->
    raise (Error (ColonSyntax tok))

| If (t, e, st_then, st_elseifs, st_else_opt) ->
  (* previ  -> newi --->   newfakethen -> ... -> finalthen --> lasti
   *                 |                                          |
   *                 |->   newfakeelse -> ... -> finalelse -|
   *
   * Can generate either special nodes for elseif, or just consider
   * elseif as syntactic sugar that translates into regular ifs, which
   * is what I do for now.
   *)
    let newi = state.g#add_node { F.n = F.IfHeader; } in
    state.g |> add_arc_opt (previ, newi);

    let newfakethen = state.g#add_node { F.n = F.TrueNode; } in
    let newfakeelse = state.g#add_node { F.n = F.FalseNode; } in
    state.g |> add_arc (newi, newfakethen);
    state.g |> add_arc (newi, newfakeelse);
```

```
    let finalthen = cfg_stmt state (Some newfakethen) st_then in

    let finalelse =
      (match st_elseifs, st_else_opt with
      | [], None ->
          Some newfakeelse
      | [], Some (tok, st_else) ->
          cfg_stmt state (Some newfakeelse) st_else
      | (t', e', st_then')::xs, else_opt ->
          (* syntactic unsugaring  *)
          cfg_stmt state (Some newfakeelse)
            (If (t', e', st_then', xs, else_opt))
      )
    in
    (match finalthen, finalelse with
    | None, None ->
        (* probably a return in both branches *)
        None
    | Some nodei, None
    | None, Some nodei ->
        Some nodei
    | Some n1, Some n2 ->
        let lasti = state.g#add_node { F.n = F.Join } in
        state.g |> add_arc (n1, lasti);
        state.g |> add_arc (n2, lasti);
        Some lasti
    )

| Return (t1, eopt, t2) ->
    let newi = state.g#add_node { F.n = F.Return } in
    state.g |> add_arc_opt (previ, newi);
    state.g |> add_arc (newi, state.exiti);

    (* the next statement if there is one will not be linked to
     * this new node *)
    None

| Continue (t1, e, t2) | Break (t1, e, t2) ->

    let is_continue, node =
      match stmt with
      | Continue _ -> true, F.Continue
      | Break _ -> false, F.Break
      | _ -> raise Impossible
    in
```

```
    let depth =
      match e with
      | None -> 1
      | Some e ->
          (match intvalue_of_expr e with
          | Some i -> i
          | None ->
              (* a dynamic variable ? *)
              raise (Error (DynamicBreak t1))
          )
    in

    let newi = state.g#add_node { F.n = node } in
    state.g |> add_arc_opt (previ, newi);

    let nodei_to_jump_to =
      state.ctx +> lookup_some_ctx
        ~level:depth
        ~ctx_filter:(function
        | LoopCtx (headi, endi) ->
            if is_continue
            then Some (headi)
            else Some (endi)

        | SwitchCtx (endi) ->
            (* it's ugly but PHP allows to 'continue' inside 'switch' (even
             * when the switch is not inside a loop) in which case
             * it has the same semantic than 'break'.
             *)
            Some endi
        | TryCtx _ | NoCtx  -> None
        )
    in
    (match nodei_to_jump_to with
    | Some nodei ->
        state.g |> add_arc (newi, nodei);
    | None ->
        raise (Error (NoEnclosingLoop t1))
    );
    None

| Switch (t1, e, cases) ->
    (match cases with
    | CaseList (_obrace, _colon_opt, cases, _cbrace) ->
        let newi = state.g#add_node { F.n = F.SwitchHeader } in
        state.g |> add_arc_opt (previ, newi);
```

```
        (* note that if all cases have return, then we will remove
         * this endswitch node later.
         *)
        let endi = state.g#add_node { F.n = F.SwitchEnd } in

        (* if no default: then must add path from start to end directly
         * todo? except if the cases cover the full spectrum ?
         *)
        if (not (cases +> List.exists
                    (function Ast.Default _ -> true | _ -> false)))
        then begin
          state.g |> add_arc (newi, endi);
        end;
        (* let's process all cases *)
        let last_stmt_opt =
          cfg_cases (newi, endi) state (None) cases
        in
        state.g |> add_arc_opt (last_stmt_opt, endi);

        (* remove endi if for instance all branches contained a return *)
        if (state.g#predecessors endi)#null then begin
          state.g#del_node endi;
          None
        end else
          Some endi

    | CaseColonList (tok, _, _, _, _) ->
        raise (Error (ColonSyntax tok))
    )

(*
 * Handling try part 1. See the case for Throw below and the
 * cfg_catches function for the second part.
 *
 * Any function call in the body of the try could potentially raise
 * an exception, so should we add edges to the catch nodes ?
 * In the same way any function call could potentially raise
 * a divide by zero or call exit().
 * For now we don't add all those edges. We do it only for explicit throw.
 *
 * todo? Maybe later the CFG could be extended with information
 * computed by a global bottom-up analysis (so that we would add certain
 * edges)
 *
 * todo? Maybe better to just add edges for all the nodes in the body
```

```
 * of the try to all the catches ?
 *
 * So for now, we mostly consider catches as a serie of elseifs,
 * and add some goto to be conservative at a few places. For instance
 *
 *   try {
 *     ...;
 *   } catch (E1 $x) {
 *     throw $x;
 *   } catch (E2 $x) {
 *     ...
 *   }
 *   ...
 *
 * is rougly considered as this code:
 *
 *   <tryheader> {
 *    if(true) goto catchstart;
 *    else {
 *      ...;
 *      goto tryend;
 *    }
 *   }
 *   <catchstart>
 *   if (is E1) {
 *     goto exit; /* or next handler if nested try */
 *   } elseif (is E2) {
 *     ...
 *     goto tryend;
 *   } else {
 *     goto exit; /* or next handler if nested try */
 *   }
 *
 *   <tryend>
 *)

| Try(t1, body, catch, other_catches) ->
    let newi = state.g#add_node { F.n = F.TryHeader } in
    let catchi = state.g#add_node { F.n = F.CatchStart } in
    state.g |> add_arc_opt (previ, newi);

    (* may have to delete it later if nobody connected to it *)
    let endi = state.g#add_node { F.n = F.TryEnd } in

    (* for now we add a direct edge between the try and catch,
     * as even the first statement in the body of the try could
```

```
 * be a function raising internally an exception.
 *
 * I just don't want certain analysis like the deadcode-path
 * to report that the code in catch are never executed. I want
 * the catch nodes to have at least one parent. So I am
 * kind of conservative.
 *)
state.g |> add_arc (newi, catchi);

let state' = { state with
  ctx = TryCtx (catchi)::state.ctx;
}
in

let stmts = stmts_of_stmt_or_defs (Ast.unbrace body) in
let last_stmt_opt = cfg_stmt_list state' (Some newi) stmts in
state.g |> add_arc_opt (last_stmt_opt, endi);

(* note that we use state, not state' here, as we want the possible
 * throws inside catches to be themselves link to a possible surrounding
 * try.
 *)
let last_false_node =
  cfg_catches state catchi endi (catch::other_catches) in

(* we want to connect the end of the catch list with
 * the next handler, if try are nested, or to the exit if
 * there is no more handler in this context
 *)
let nodei_to_jump_to =
  state.ctx +> lookup_some_ctx ~ctx_filter:(function
  | TryCtx (nextcatchi) -> Some nextcatchi
  | LoopCtx _ | SwitchCtx _ | NoCtx -> None
  )
in
(match nodei_to_jump_to with
| Some nextcatchi ->
    state.g |> add_arc (last_false_node, nextcatchi)
| None ->
    state.g |> add_arc (last_false_node, state.exiti)
);

(* todo? if nobody connected to endi ? erase the node ? for instance
 * if have only return in the try body ?
 *)
Some endi
```

```
    (*
     * For now we don't do any fancy analysis to statically detect
     * which exn handler a throw should go to. The argument of throw can
     * be static as in 'throw new ExnXXX' but it could also be dynamic. So for
     * now we just branch to the first catch and make edges between
     * the different catches in cfg_catches below
     * (which is probably what is done at runtime by the PHP interpreter).
     *
     * todo? Again maybe later the CFG could be sharpened with
     * path sensitive analysis to be more precise (so that we would remove
     * certain edges)
     *)
    | Throw (t1, e, t2) ->
        let newi = state.g#add_node { F.n = F.Throw } in
        state.g |> add_arc_opt (previ, newi);

        let nodei_to_jump_to =
          state.ctx +> lookup_some_ctx
            ~ctx_filter:(function
            | TryCtx (catchi) ->
                Some catchi
            | LoopCtx _ | SwitchCtx _ | NoCtx ->
                None
            )
        in
        (match nodei_to_jump_to with
        | Some catchi ->
            state.g |> add_arc (newi, catchi)
        | None ->
            (* no enclosing handler, branch to exit node of the function *)
            state.g |> add_arc (newi, state.exiti)
        );
        None

    |
      (Declare (_, _, _)|Unset (_, _, _)|Use (_, _, _)|
      StaticVars (_, _, _)|Globals (_, _, _)
      )
        -> raise Todo


and cfg_stmt_list state previ xs =
  xs +> List.fold_left (fun previ stmt ->
    cfg_stmt state previ stmt
  ) previ
```

```
and cfg_colon_stmt state previ colon =
  let stmts = stmts_of_colon_stmt colon in
  cfg_stmt_list state previ stmts

(*
 * Creating the CFG nodes and edges for the cases of a switch.
 *
 * PHP allows to write code like  case X: case Y: ... This is
 * parsed as a [Case (X, []); Case (Y, ...)] which means
 * the statement list of the X case is empty. In this situation we just
 * want to link the node for X directly to the node for Y.
 *
 * So cfg_cases works like cfg_stmt by optionally taking the index of
 * the previous node (here for instance the node of X), and optionally
 * returning a node (if the case contains a break, then this will be
 * None)
 *)

and (cfg_cases:
    (nodei * nodei) -> state ->
    nodei option -> Ast_php.case list -> nodei option) =
 fun (switchi, endswitchi) state previ cases ->

   let state = { state with
     ctx = SwitchCtx (endswitchi)::state.ctx;
   }
   in

   cases +> List.fold_left (fun previ case ->
     let node, stmt_or_defs =
       match case with
       | Case (t1, e, t2, stmt_or_defs) ->
           F.Case, stmt_or_defs
       | Default (t1, t2, stmt_or_defs) ->
           F.Default, stmt_or_defs
     in
     let newi = state.g#add_node { F.n = node } in
     state.g |> add_arc_opt (previ, newi);
     (* connect SwitchHeader to Case node *)
     state.g |> add_arc (switchi, newi);

     let stmts = stmts_of_stmt_or_defs stmt_or_defs in
     (* the stmts can contain 'break' that will be linked to the endswitch *)
     cfg_stmt_list state (Some newi) stmts
   ) previ
```

```
(*
 * Creating the CFG nodes and edges for the catches of a try.
 *
 * We will conside catch(Exn $e) as a kind of if, with a TrueNode for
 * the case the thrown exn matched the specified class,
 * and FalseNode otherwise.
 *
 * cfg_catches takes the nodei of the previous catch nodes (or false node
 * of the previous catch node), process the catch body, and return
 * a new False Node.
 *)

and (cfg_catches: state -> nodei -> nodei -> Ast_php.catch list -> nodei) =
 fun state previ tryendi catches ->
   catches +> List.fold_left (fun previ catch ->
     let (t, e_paren, stmt_or_defs) = catch in
     let newi = state.g#add_node { F.n = F.Catch } in
     let truei = state.g#add_node { F.n = F.TrueNode } in
     let falsei = state.g#add_node { F.n = F.FalseNode } in
     state.g |> add_arc (previ, newi);
     state.g |> add_arc (newi, truei);
     state.g |> add_arc (newi, falsei);

     let stmts = stmts_of_stmt_or_defs (Ast.unbrace stmt_or_defs) in
     (* the stmts can contain 'throw' that will be linked to an upper try or
      * exit node *)
     let last_stmt_opt = cfg_stmt_list state (Some truei) stmts in
     state.g |> add_arc_opt (last_stmt_opt, tryendi);

     (* we chain the catches together, like elseifs *)
     falsei
   ) previ
```

14.5 Debugging information

70 ⟨*function display_flow* 70⟩≡

```
(* using internally graphviz dot and ghostview on X11 *)
let (display_flow: flow -> unit) = fun flow ->
  flow +> Ograph_extended.print_ograph_mutable_generic
    ~s_of_node:(fun (nodei, node) ->
      short_string_of_node node, None, None
    )
```

71a ⟨*function short_string_of_node* 71a⟩≡

```
let short_string_of_node node =
  match node.n with
  | Enter -> "<enter>"
  | Exit -> "<exit>"

  | SimpleStmt _ -> "<simplestmt>"

  | WhileHeader _ -> "while(...)"

  | TrueNode -> "TRUE path"
  | FalseNode -> "FALSE path"

  | IfHeader _ -> "if(...)"
  | Join _ -> "<join>"

  | Return _ -> "return ...;"

  | DoHeader -> "do"
  | DoWhileTail -> "while(...);"

  | Continue -> "continue;"
  | Break -> "break;"

  | ForHeader -> "for(...)"
  | ForeachHeader -> "foreach(...)"

  | SwitchHeader -> "switch(...)"
  | SwitchEnd -> "<endswitch>"

  | Case -> "case: ..."
  | Default -> "default:"

  | TryHeader -> "try"
  | CatchStart -> "<catchstart>"
  | Catch -> "catch(...)"
  | TryEnd -> "<endtry>"

  | Throw -> "throw ...;"
```

14.6 Extra code

71b ⟨*controlflow_php helpers* 71b⟩≡

```
let stmts_of_stmt_or_defs xs =
  xs |> Common.map_filter (fun stmt_or_def ->
```

```
    match stmt_or_def with
    | Stmt st -> Some st
    | FuncDefNested _ | ClassDefNested _ | InterfaceDefNested _ ->
        pr2_once ("ignoring nested func/class/interface in CFG");
        None
  )

let stmts_of_colon_stmt colon =
  match colon with
  | SingleStmt stmt -> [stmt]
  | ColonStmt (tok, _, _, _) -> raise (Error (ColonSyntax tok))
```

72 ⟨*controlflow_php helpers* 71b⟩+≡

```
let add_arc (starti, nodei) g =
  g#add_arc ((starti, nodei), F.Direct)

let add_arc_opt (starti_opt, nodei) g =
  starti_opt |> Common.do_option (fun starti ->
    g#add_arc ((starti, nodei), F.Direct)
  )

(*
 * When there is a 'break', 'continue', or 'throw', we need to look up in the
 * stack of contexts whether there is an appropriate one. In the case
 * of 'break/continue', because PHP allows statements like 'break 2;', we also
 * need to know how many upper contexts we need to look for.
 *)
let rec (lookup_some_ctx:
  ?level:int ->
  ctx_filter:(context -> 'a option) ->
  context list -> 'a option) =
 fun ?(level=1) ~ctx_filter xs ->

   let rec aux depth xs =
     match xs with
     | [] -> None
     | x::xs ->
         (match ctx_filter x with
         | None -> aux depth xs
         | Some a ->
             if depth = level
             then (Some a)
             else
               aux (depth+1) xs
         )
   in
```

```
    aux 1 xs

let intvalue_of_expr e =
  match Ast.untype e with
  | (Scalar (Constant (Int (i_str, _)))) ->
      Some (s_to_i i_str)
  | _ -> None
```

73a ⟨*controlflow builders* 59a⟩+≡

```
(* alias *)
let cfg_of_stmts = control_flow_graph_of_stmts
```

73b ⟨*function deadcode_detection* 73b⟩≡

```
let (deadcode_detection : F.flow -> unit) = fun flow ->
  raise Todo
```

73c ⟨*function Controlflow_build_php.report_error* 73c⟩≡

```
let (report_error : error -> unit) = fun err ->
  let error_from_info info =
    let pinfo = Ast.parse_info_of_info info in
    Common.error_message_short pinfo.Common.file ("", pinfo.Common.charpos)
  in

  match err with
  | ColonSyntax info ->
      pr2 ("FLOW: dude, don't use the old PHP colon syntax: " ^
        error_from_info info)
  | DeadCode info ->
      pr2 ("FLOW: deadcode path detected at: " ^ error_from_info info)
  | NoEnclosingLoop info ->
      pr2 ("FLOW: no enclosing loop found for break or continue at: "
            ^ error_from_info info)
  | NoMethodBody info ->
      pr2 ("FLOW: can't compute CFG of an abstract method at: "
            ^ error_from_info info)
  | DynamicBreak info ->
      pr2 ("FLOW: dynamic break/continue are not supported at: "
            ^ error_from_info info)
```

73d ⟨*controlflow_php accessors* 73d⟩≡

```
let (first_node : flow -> Ograph_extended.nodei) = fun flow ->
  raise Todo

let (mk_node: node_kind -> node) = fun nk ->
  raise Todo
```

Chapter 15

Data Flow Analysis

74 ⟨*dataflow_php.ml* 74⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

open Ast_php

module Ast = Ast_php

module F = Controlflow_php

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * The goal of a dataflow analysis is to store information about each
 * variable at each program point, that is each node in a CFG.
 * As you may want different kind of information, the types below
 * are polymorphic. But each take as a key a variable name (dname, for
 * dollar name, the type of variables in Ast_php).
 *
 * less: could use a functor, so would not have all those 'a.
 *)

(*****************************************************************************)
(* Types *)
(*****************************************************************************)

(* Information about each variable *)
type 'a env =
```

```
  (Ast_php.dname, 'a) Common.assoc

(* Values of this type will be associated to each CFG nodes and computed
 * through a classical fixpoint analysis on the CFG. For instance see
 * tainted_php.ml
 *)
type 'a inout = {
  in_env: 'a env;
  out_env: 'a env;
}

type 'a mapping =
  (Ograph_extended.nodei, 'a inout) Common.assoc

(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)

(* TODO? having only a transfer function is enough ? do we need to pass
 * extra information to it ? maybe only the mapping is not enough. For
 * instance if in the code there is $x = &$g, a reference, then
 * we may want later to have access to this information. Maybe we
 * should pass an extra env argument ? Or maybe can encode this
 * sharing of reference in the 'a, so that when one update the
 * value associated to a var, its reference variable get also
 * the update.
 *)

let (fixpoint:
      F.flow ->
      initial:('a mapping) ->
      transfer:('a mapping list (* in edges *) -> Ograph_extended.nodei ->
                'a mapping) ->
      'a mapping) =
 fun flow ~initial ~transfer ->
  raise Todo




let (display_dflow: F.flow -> 'a mapping -> ('a -> string) -> unit) =
 fun flow mapping string_of_val ->
   raise Todo
```

Chapter 16

Typing

16.1 Trivial typing

76 ⟨*typing_trivial_php.ml* 76⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

open Ast_php

module Flag = Flag_analyze_php
module Ast = Ast_php

module V = Visitor_php
module T = Type_php
module N = Namespace_php


(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * Typing using information from xdebug traces. Mostly trivial :)
 *)

(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)

let rec type_of_constant x =
```

```
  match x with
  | Int _ -> T.Basic T.Int
  | Double _ -> T.Basic T.Float
  | String _ -> T.Basic T.String

  | CName name ->
      (match Ast.name name with
      | "TRUE" -> T.Basic T.Bool
      | "FALSE" -> T.Basic T.Bool

      | "NULL" -> T.Null
      | _ -> raise Todo
      )
  | PreProcess _ ->
      T.Basic T.String
  | XdebugClass (name, xs) ->
      (* TODO xs *)
      T.Object (Some (Ast.name name))
  | XdebugResource ->
      T.Resource

let rec type_of_scalar x =
  match x with
  | Constant cst ->
      type_of_constant cst


  | _ -> raise Todo

let rec (type_of_expr: Ast_php.expr -> Type_php.phptype) = fun e ->
  match Ast.untype e with
  | Scalar scalar ->
      [type_of_scalar scalar]

  | ConsArray (_t, xs_paren) ->
      (* TODO more precise *)
      [T.ArrayFamily (T.Hash ([T.Unknown]))]

  | Unary ((UnMinus, _), e) -> type_of_expr e

  (* xdebug use sometimes ???, which in xdebug.ml is replaced by ... *)
  | EDots _ ->
      [T.Unknown]
  | _ -> raise Todo
```

Chapter 17

Xdebug traces

78 ⟨*xdebug.ml* 78⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

open Ast_php

module Ast = Ast_php

module CG = Callgraph_php

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * Wrappers around the really good debugger/profiler/tracer for PHP:
 *
 *  http://xdebug.org/
 *
 * See also dynamic_analysis.ml
 *
 * php -d xdebug.collect_params=3
 *  => full variable contents with the limits respected by
 *   xdebug.var_display_max_children, max_data, max_depth
 *
 * But the dumped file can be huge:
 *  - for unit tested flib/core/ 1.8 Go
 *  - for unit tested flib/ 5Go ...
 *
 * so had to optimize a few things:
```

```
 *
 * - use compiled regexp,
 * - fast-path parser for expression.
 * - use iterator interface instead of using lists.
 *)

(*****************************************************************************)
(* Wrappers *)
(*****************************************************************************)

(*****************************************************************************)
(* Types *)
(*****************************************************************************)

(* when use the trace=3 format *)
type call_trace = {
  f_call: Callgraph_php.kind_call;
  f_file: Common.filename;
  f_line: int;
  f_params: Ast_php.expr list;
  f_return: Ast_php.expr option;

  (* f_type: *)
}

let xdebug_main_name = "xdebug_main"

(*****************************************************************************)
(* String of *)
(*****************************************************************************)


(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)

let sanitize_xdebug_expr_for_parser2 str =

(* Str is buggy
  let str = Str.global_replace
    (Str.regexp "class\b") "class_xdebug" str
  in
  (* bug in xdebug output *)
  let str = Str.global_replace
    (Str.regexp ")[ \t]*}") ");}" str
  in
```

```
  str
  *)
  let str = Pcre.replace ~pat:"\\?\\?\\?"  ~templ:"..." str in
  let str = Pcre.replace ~pat:"class"  ~templ:"class_xdebug" str in
  let str = Pcre.replace
    ~pat:"resource\\([^)]*\\) of type \\([^)]*\\)"
    ~templ:"resource_xdebug" str
  in

  (* bug in xdebug output *)
  let str = Pcre.replace ~pat:"([^{ \t])[ \t]+}"   ~itempl:(Pcre.subst "$1;}") str in
  str

let sanitize_xdebug_expr_for_parser a =
  Common.profile_code "sanitize" (fun () -> sanitize_xdebug_expr_for_parser2 a)


(* bench:
 * on toy.xt:
 *  - 0.157s with regular expr parser
 *  - 0.002 with in memory parser that dont call xhp, tag file info, etc
 * on flib_core.xt: 44s in native mode
 *)
let parse_xdebug_expr2 s =
  (* Parse_php.expr_of_string s *)
  Parse_php.xdebug_expr_of_string s

let parse_xdebug_expr a =
  Common.profile_code "Xdebug.parse" (fun () -> parse_xdebug_expr2 a)


(*****************************************************************************)
(* Parsing regexps *)
(*****************************************************************************)

(* examples:
 *   0.0009      99800   -> {main}() /home/pad/mobile/project-facebook/pfff/tests/xdebug/b
 *   0.0009      99800     -> main() /home/pad/mobile/project-facebook/pfff/tests/xdebug/b
 *   0.0006      97600     -> A->__construct(4) /home/pad/mobile/project-facebook/pfff/tes
 * TODO               ModuleStack::current()
 *
 *)
let regexp_in =
  Str.regexp
    ("\\(" ^ (* 1, everything before the arrow *)
       "[ \t]+" ^
```

```
      "\\([0-9]+\\.[0-9]+\\)[ \t]+\\([0-9]+\\)" ^ (* 2 3, time and nbcall? *)
      "[ \t]+" ^
      "->[ \t]+" ^
    "\\)" ^ (* end of arrow *)
      "\\([^(]+\\)" ^ (* 4, funcname or methodcall *)
      "(\\(.*\\)) " ^ (* 5, arguments *)
      "\\(/.*\\)" ^ (* 6, filename *)
      ":" ^
      "\\([0-9]+\\)" ^ (* 7, line *)
      "")


(* examples:
 *         >=> 8
 *)

let regexp_out =
  Str.regexp
    ("\\([ \t]+\\)" ^ (* 1, spacing *)
     ">=> " ^
     "\\(.*\\)" (* 2, returned expression *)
    )

(* example:
 *   0.0009      99800   -> {main}() /home/pad/mobile/project-facebook/pfff/tests/xdebug/b
 *)
let regexp_special_main =
  Str.regexp ".*-> {main}()"


let regexp_last_time =
  Str.regexp (
    "^[ \t]+[0-9]+\\.[0-9]+[ \t]+[0-9]+$"
  )

let regexp_meth_call =
   Str.regexp (
   "\\([A-Za-z_0-9]+\\)->\\(.*\\)"
   )

let regexp_class_call =
   Str.regexp (
   "\\([A-Za-z_0-9]+\\)::\\(.*\\)"
   )
let regexp_fun_call =
   Str.regexp (
```

```
   "[A-Za-z_0-9]+$"
   )

(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)

(* Can not have a parse_dumpfile, because it is usually too big *)
let iter_dumpfile2 callback file =

  pr2 (spf "computing number of lines of %s" file);
  let nblines = Common.nblines_with_wc file in

  pr2 (spf "nb lines = %d" nblines);
  let nb_fails = ref 0 in

  Common.execute_and_show_progress nblines (fun k ->
  Common.with_open_infile file (fun (chan) ->

    try
      while true do
        let line = input_line chan in

        k ();

        match line with
        (* most common case first *)
        | _ when line =~ regexp_in ->
            (* pr2 "IN:"; *)
            let (before_arrow, _time, _nbcall, call, args, file, lineno) =
              Common.matched7 line
            in
            let str = sanitize_xdebug_expr_for_parser args in
            let str = "foo(" ^ str ^ ")" in

            let trace_opt =
            (try
              let kind_call =
                match () with
                | _ when call =~ regexp_meth_call ->
                    let (sclass, smethod) = Common.matched2 call in
                    CG.ObjectCall(sclass, smethod)
                | _ when call =~ regexp_class_call ->
                    let (sclass, smethod) = Common.matched2 call in
                    CG.ClassCall(sclass, smethod)
                | _ when call =~ regexp_fun_call ->
```

```
        CG.FunCall(call)
    | _ when call = "{main}" ->
        CG.FunCall(xdebug_main_name)
    | _ ->
        failwith ("not a funcall: " ^ call)
  in
  (match kind_call with
  | CG.FunCall "require_once"
  | CG.FunCall "include_once"
  | CG.FunCall "include"
  | CG.FunCall "require"
    -> None
  | _ ->
      let expr = parse_xdebug_expr str in

      let args =
        match Ast.untype expr with
        | Lvalue var ->
            (match Ast.untype var with
            | FunCallSimple (qu_opt, name, args_paren) ->
                assert(Ast.name name = "foo");
                Ast.unparen args_paren +> List.map (function
                | Arg e -> e
                | _ -> raise Impossible
                )
            | _ -> raise Impossible
            )
        | _ -> raise Impossible
      in

      let trace = {
        f_call = kind_call;
        f_file = file;
        f_line = s_to_i lineno;
        f_params = args;
        f_return = None;
      }
      in
      Some trace
  )
with exn ->
   pr2(spf "php parsing pb: exn = %s, s = %s"
          (Common.exn_to_s exn) str);
   incr nb_fails;
   (* raise exn *)
   None
```

```
          )
          in
          (try
            trace_opt +> Common.do_option callback
          with exn ->
            pr2(spf "callback pb: exn = %s, line = %s"
                    (Common.exn_to_s exn) line);
            raise exn
          )



      | _ when line ==~ regexp_out ->
          (* pr2 "OUT:"; *)
          let (before_arrow, return_expr) = Common.matched2 line in
          let str = sanitize_xdebug_expr_for_parser return_expr in

          (try
            let _expr = parse_xdebug_expr str  in
            ()
          with
           exn ->
             pr2("php parsing pb: " ^ str);
             incr nb_fails;
             (* raise exn *)
          )

      | _ when line =~ "^TRACE" -> ()
      | _ when line ==~ regexp_special_main -> ()
      | _ when line ==~ regexp_last_time -> ()


      | _ when line = "" -> ()
      | _ ->
          (* TODO error recovery ? *)
          pr2 ("parsing pb: " ^ line);
          incr nb_fails;

    done
  with End_of_file -> ()

));
pr2 (spf "nb xdebug parsing fails = %d" !nb_fails);
()
```

```
let iter_dumpfile a b =
  Common.profile_code "Xdebug.iter_dumpfile" (fun () -> iter_dumpfile2 a b)
```

Chapter 18

Code Rank

86 ⟨*code_rank_php.ml* 86⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

module Flag = Flag_analyze_php
module Db   = Database_php
module EC   = Entity_php
module CG   = Callgraph_php

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * reference:
 *   "CodeRank: A New Family of Software Metrics"
 *    B. Neate, W. Irwin,  N. Churcher
 *
 * Thx to sebastien bergmann for pointing out this paper.
 *
 * How it compares to naive metrics which is counting the number of callers ?
 * Do we find more important functions ? Normally if a function is
 * not very often called, but called by an important function, coderank
 * will find it, but our naive metrics will not. Do we have such
 * example of important but small function ?
 *
 *
 *)
(*****************************************************************************)
(* Types *)
```

```
(*****************************************************************************)

type code_ranks = {
  function_ranks: (Database_php.id, float) Oassoc.oassoc;
}
let empty_code_ranks () = {
    function_ranks = new Oassoch.oassoch [];
  }

(* code id. Can be just equal to Database_php.id, which may generate
 * sparse tables, or something else that requires a mapping from it
 * to a Database_php.id
 *)
type cid = int

(* big, indexed by code id *)
type ranks = float array

  (* fast enough ? why not float array ? because later we may want to
   * and cr = {
   *  mutable score: float;
   * mutable cid: cid;
   * }
   *)


type equation_right = (cid * float (* dividing factor *))  list

(* big, indexed by code id *)
type equations = equation_right array

(* side effect on ranks *)
let eval_equation ranks d eq =
  let contribs =
    (eq +> List.fold_left (fun acc (cid,  diviser) ->
      (* bugfix: forgot to add acc :) *)
      acc +. ranks.(cid) /. diviser
    ) 0.0)
  in
  let newv = (1.0 -. d) +. d *. contribs
  in
  newv

(*****************************************************************************)
(* Helper *)
(*****************************************************************************)
```

```
let (initial_value: Database_php.id list -> int -> ranks) = fun ids maxid ->
  let (arr: ranks) = Array.make (maxid + 1) 0.1 in
  arr

let (equations: Database_php.id list -> int -> Database_php.database -> equations) =
 fun ids maxid db ->
  let (eqs: equations) = Array.make (maxid + 1) [] in

  ids +> List.iter (fun id ->
    let (EC.Id this_cid) = id in

    (* note that the callers may be toplevel statement *)
    let callers = Db.callers_of_id id db in
    let eq_right =
      callers +> Common.map_filter (fun callerinfo ->
        let callerid = CG.id_of_callerinfo callerinfo in

        let (EC.Id cid) = callerid in
        (* ignore for now caller which are not functions *)

        if not (Db.is_function_id callerid db)
           || callerid = id (* dont want recursive calls to messup data *)
        then None
        else
          let callees_of_caller =
            Db.callees_of_id callerid db in

          let nbcallees = List.length callees_of_caller in
          Some (cid, float_of_int nbcallees)
      )
    in
    eqs.(this_cid) <- eq_right;
  );
  eqs

let (fixpoint: ranks -> equations -> ranks) = fun ranks equations ->

  let changed = ref true in
  let depth_limit = 500 in
  let step = ref 0 in
  pr2 ("computing code ranks fixpoint");

  while !changed && !step < depth_limit do
    changed := false;
    incr step;
```

```
    pr2 (spf "step = %d" !step);

    for i = 0 to Array.length ranks - 1 do
      let oldv = ranks.(i) in
      let newv = eval_equation ranks 0.85  equations.(i) in
      if oldv <> newv
      then changed := true;
      ranks.(i) <- newv;
    done
  done;
  ranks

let (ranks_to_code_ranks: Database_php.id list -> ranks -> code_ranks) =
 fun ids ranks ->
  let cr = empty_code_ranks () in
  ids +> List.iter (fun (EC.Id cid) ->
    cr.function_ranks#add2 (EC.Id cid, ranks.(cid));
  );
  cr

(*****************************************************************************)
(* Builder *)
(*****************************************************************************)

let build_code_ranks db =

  (* let's focus on functions first *)
  let funcids = Db.functions_in_db db in

  let allids = ref [] in
  let maxid = ref 0 in

  funcids +> List.iter (fun (s, ids) ->
    pr2 s;
    ids +> List.iter (fun (EC.Id id) ->
      Common.push2 (EC.Id id) allids;
      if id > !maxid then maxid := id;
    );
  );

  pr2 (spf "maxid = %d, nb ids = %d" !maxid (List.length !allids));

  let ranks = initial_value !allids !maxid in
  let eqs = equations !allids !maxid db in
  let ranks = fixpoint ranks eqs in
  ranks_to_code_ranks !allids ranks
```

```
let build_naive_caller_ranks db =
  let funcids = Db.functions_in_db db in
  let cr = empty_code_ranks () in
  funcids +> List.iter (fun (s, ids) ->
    ids +> List.iter (fun id ->
      let nbcallers = Db.callers_of_id id db in
      cr.function_ranks#add2 (id, float_of_int (List.length nbcallers));
    )
  );
  cr


(*****************************************************************************)
(* Testing *)
(*****************************************************************************)

(* toy example in paper is in tests/code_rank
*)
```

Chapter 19

Cyclomatic Complexity

91 ⟨*cyclomatic_php.ml* 91⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * References:
 *  - A Complexity Measure, by  Thomas J. McCabe, IEEE Transactions on
 *    software engineering, Dec 1976
 *  - http://en.wikipedia.org/wiki/Cyclomatic_complexity
 *
 * From the wikipedia page:
 * "Cyclomatic complexity (or conditional complexity) is a software metric
 * (measurement). It was developed by Thomas J. McCabe, Sr. in 1976 and
 * is used to indicate the complexity of a program. It directly measures
 * the number of linearly independent paths through a program's source
 * code.
 *
 * The complexity is then defined as
 *
 * M = E - N + 2P
 *
 * where
 *
 * M = cyclomatic complexity
 * E = the number of edges of the graph
 * N = the number of nodes of the graph
```

```
 * P = the number of connected components
 *
 * ...
 * For a single program (or subroutine or method), P is always equal to 1.
 * "
 *
 * The goal of having a set of "linear independent paths"
 * is to have a set of "foundational paths" from which all other paths
 * can be described (by a linear combination of those foundational paths).
 *
 * Note that the cyclomatic complexity does not care about
 * the complexity of expressions or function calls. For instance
 * 'if(e1 || e1 || e3) { ...} will have the same cyclomatic complexity
 * as 'if(e1) { ... }' because they have a similar CFG.
 *)

(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)

let cyclomatic_complexity_flow ?(verbose=false) flow =

  let n = flow#nb_nodes in
  let e = flow#nb_edges in
  let p = 1 in

  let m = e - n +  2 * p  in

  if verbose
  then pr2 (spf "N = %d, E = %d" n e);

  m

let cyclomatic_complexity_func ?verbose func =
  let flow = Controlflow_build_php.cfg_of_func func in
  cyclomatic_complexity_flow flow

let cyclomatic_complexity_method ?verbose meth =
  let flow = Controlflow_build_php.cfg_of_method meth in
  cyclomatic_complexity_flow flow
```

Chapter 20

Dead code

93 ⟨*deadcode_php.ml* 93⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

open Ast_php

module Flag = Flag_analyze_php
module Ast = Ast_php

module V = Visitor_php
module T = Type_php
module N = Namespace_php

module Db = Database_php
module DbQ = Database_php_query

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * For the moment we just look if the function has no callers and is not
 * mentionned in some strings somewhere (but look for exact string).
 * update: We do a few heuristics for function pointer that
 * are passed via the hooks is_probable_dynamic_funcname.
 * See aliasing_function_php.ml which was used to compute
 * some of the value for this hook.
 *
 * After the heuristic suggested by lklots, got now
 * 2421 dead functions. I think it was 3500 at the beginning when
```

```
 * I was not even doing the exact-string-match false positive heuristic.
 *
 *
 * history:
 * - deadcode_patches.tgz:
 * - deadcode_patches-v2.tgz: better heuristic for funcvar
 * - deadcode_patches-v3.tgz: better blame -C, and filter third-party stuff
 * - deadcode_patches-v4.tgz: do not generate patch for files where
 *   tbgs/glimpse would return multiple match on certain deadcode functions
 * - deadcode_patches-v5.tgz: add date information to the patch filename
 *   so can later decide to apply first old dead code patches
 * - deadcode_patches-v6.tgz: do fixpoint. Also do CEs when using
 *   nested ast_ids
 *
 * Some of the code specific to facebook is in facebook/ and in main_db.ml
 *
 * Here is the first commit :)
 *
 *   commit e52575f16c70a4507125a1e3a662456cb36d057d
 *   Author: dcorson <dcorson@2c7ba8d8-a2f7-0310-a573-de162e16dcc7>
 *   Date:   2 weeks ago
 *
 *   initial batch of dead code from ~pad's static analyzer
 *
 *   Summary: i also further deleted a couple files that were now empty
 *   like lib/deprecated.php (removing any includes of them as well).
 *   this is a test run of 30 files.  ~pad has over a thousand more that
 *   he will probably be sending out to ppl that they blame to, after we
 *   see how this initial batch goes.
 *
 *   Reviewed By: epriestley
 *
 *   Test Plan: grep for bunch of dead functions and all dead files,
 *   browse site and use i18n stuff and import a blog.
 *
 *   Revert Plan: ok
 *
 *   git-svn-id: svn+ssh://tubbs/svnroot/tfb/trunk/www@202716 2c7ba8d8-a2f7-0310-a573-de16
 *
 *)

(*****************************************************************************)
(* Types and globals *)
(*****************************************************************************)

type hooks = {
```

```
  (* to remove certain false positives *)
  is_probable_dynamic_funcname: string -> bool;

  (* to avoid generating patches for code which does not have a valid
   * git owner anymore (the guy left the company for instance ...)
   *)
  is_valid_author: string -> bool;

  (* to avoid generating patches for certain files, such as code in
   * third party libraries or auto generated code. Will be called
   * with a filename without leading project.
   *)
  is_valid_file: filename -> bool;

  (* code annotated with @not-dead-code should not be considered *)
  false_positive_deadcode_annotations: Annotation_php.annotation list;

  (* config *)
  print_diff: bool;
  with_blame: bool;
  cache_git_blame: bool;
  (* place where we would put the generated patches *)
  patches_path: Common.dirname;
}

let ext_git_annot_cache = ".git_annot"

let default_hooks = {
  is_probable_dynamic_funcname = (fun s -> false);
  is_valid_author  = (fun s -> true);
  is_valid_file = (fun filename -> true);
  false_positive_deadcode_annotations = [
    Annotation_php.CalledFromPhpsh;
    Annotation_php.CalledOutsideTfb;
    Annotation_php.NotDeadCode;
  ];
  print_diff = true;
  with_blame = false;
  cache_git_blame = true;
  patches_path = "/tmp/deadcode_patches";
}

type removed_lines = {
  just_code_lines: int list;
  code_and_comment_lines: int list;
}
```

```
type deadcode_patch_info = {
  file     : Common.filename; (* path relative to the project *)
  reviewer : string option; (* maybe nobody ... *)
  cc       : string option;
  date     : Common.date_dmy;
}

type dead_ids_by_file =
  (Common.filename * (string * Database_php.fullid * Database_php.id) list)

(*****************************************************************************)
(* Deadcode analysis Helpers *)
(*****************************************************************************)

(* TODO: should also analyze dead methods, dead classes *)

let false_positive fid hooks db =
  let extra = db.Db.defs.Db.extra#assoc fid in
  let s = Db.name_of_id fid db in

  match () with

  | _ when db.Db.strings#haskey s ->
      pr2 ("Probable indirect call (or because not yet parsed): " ^ s);
      true

  | _ when hooks.is_probable_dynamic_funcname s ->
      pr ("Probable dynamic call: " ^ s);
      true
  | _ when hooks.false_positive_deadcode_annotations +>
        List.exists (fun annot -> List.mem annot extra.Db.tags) ->

      pr2("tagged as @called-from-phpsh or related: " ^ s);
      true

  | _ ->
      false


let finding_dead_functions hooks db =
  let dead_ids = ref [] in

  Db.functions_in_db db +> List.iter (fun (idstr, ids) ->
    let s = idstr in
```

```
    if List.length ids > 1
    then pr2 ("Ambiguous name: " ^ s);

    ids +> List.iter (fun id ->

      let file_project = Db.filename_in_project_of_id id db in

      if hooks.is_valid_file file_project then begin

        let is_dead_func =
            (* TODO what if recursive ? or mutually recursive but dead ? *)
          let callers = Db.callers_of_id id db in
          null callers && not (false_positive id hooks db)
        in
        if is_dead_func
        then begin
          pr ("DEAD FUNCTION: no caller for " ^ s);
          Common.push2 (s, id) dead_ids;
        end
      end
    );
  );
  pr2 (spf "number of dead functions: %d" (List.length !dead_ids));
  !dead_ids



(* fixpoint per file. Just remember ids, then reitarate again but on each
 * file, and get all ids in it which are function and look at callers again
 * and see if in current set of dead, and add!
 *)
let deadcode_fixpoint_per_file grouped_by_file all_dead_ids hooks db =
  let hdead = Common.hashset_of_list all_dead_ids in

  grouped_by_file +> List.map (fun (file, funcs_and_ids) ->
    (* TODO right now do fixpoint on the file, but maybe should do
     * it more globally ?
     *)

    let ids_file =
      db.Db.file_to_topids#assoc file in

    let candidates =
      ids_file +> List.filter (fun id ->
        not (Hashtbl.mem hdead id)       &&
        Db.is_function_id id db          &&
```

```
        not (false_positive id hooks db) &&
        true
      )
    in

    let rec fix also_dead remaining =
      let new_dead, remaining' =
        remaining +> List.partition (fun id ->
          let callers = Db.callers_of_id id db in
          let idcallers = callers +> List.map Callgraph_php.id_of_callerinfo in
          let ids_not_dead =
            idcallers +> Common.exclude (fun id2 ->
              (* bugfix: put a && instead of || ... *)
              Hashtbl.mem hdead id2 || id = id2 (* recursive calls *)
            )
          in
          null ids_not_dead
        )
      in
      if null new_dead
      then also_dead (* fixpoint reached *)
      else begin
        new_dead +> List.iter (fun id -> Hashtbl.add hdead id true);
        fix (new_dead ++ also_dead) remaining'
      end
    in
    let new_dead = fix [] candidates in

    let info_new_dead = new_dead +> List.map (fun id ->
        let s = Db.name_of_id id db in
        let fullid = db.Db.fullid_of_id#assoc id in
        pr ("DEAD FUNCTION: no caller when do fixpoint for " ^ s);
        s, fullid, id
    )
    in
    (file, info_new_dead ++ funcs_and_ids)
  )


(*****************************************************************************)
(* Patch generation helpers *)
(*****************************************************************************)

(*
 * Analyze which lines to remove. Try also to remove associated comments.
 *
```

```
 * Before I was eating too much tokens.
 *
 *  bad: let (min, max) = db.Db.defs.Db.range_of_ast#assoc id in
 *
 * This is because some whitespaces like newlines are associated
 * with the entity and they sometimes are on the same line that
 * some '}',
 *
 * So first get ii of ast, then process tokens
 * until first ii, filter space, and look if comment,
 * then take this comment too.
 * check if first column!! or nobody else on its line
 *
 * Update: we dont want to use the lines of comment removed to take
 * a decision about the blamer. So now returns two set of lines,
 * the code lines and code-and-comment lines. For instance sgrimm
 * was blamed for some deadcode where he actually only modified
 * the comment (and in quite mechanical way); better to blame
 * the person who wrote the code when it comes to deadcode removal.
 *
 * bugfix: sometimes in facebook they add 2 comments, as in
 *  /**
 *   * Determine if an actor's whitelist has a specific recipient in it.
 *   */
 *  // memcache-refactor: remove this function (check if called anywhere still)
 *
 * so have to handle this special case too.
 *)
let get_lines_to_remove db ids =
  ids +> List.map (fun (s, fullid, id) ->
    let ast = Db.ast_of_id id db  in

    let toks = Db.toks_of_topid_of_id id db in

    let ii = Lib_analyze_php.ii_of_id_ast ast in
    let (min, max) = Lib_parsing_php.min_max_ii_by_pos ii in

    let min = Ast_php.parse_info_of_info min in
    let max = Ast_php.parse_info_of_info max in

    let toks_before_min =
      toks +> List.filter (fun tok ->
        Token_helpers_php.pos_of_tok tok < min.charpos
      ) +> List.rev
    in
    let min_comment =
```

```
    match toks_before_min with
    | Parser_php.T_WHITESPACE i1::
        (Parser_php.T_COMMENT i2|Parser_php.T_DOC_COMMENT i2)::xs ->
        if Ast_php.col_of_info i2 = 0 &&
           (* bugfix: dont want comment far away *)
           Ast_php.line_of_info i1 = min.Common.line - 1
        then Ast_php.parse_info_of_info i2
        else min

    (* bugfix *)
    | Parser_php.T_COMMENT i1first::
        Parser_php.T_WHITESPACE i1::
        (Parser_php.T_COMMENT i2|Parser_php.T_DOC_COMMENT i2)::xs
      ->
        if Ast_php.col_of_info i1first = 0 &&
           Ast_php.col_of_info i2 = 0
        then Ast_php.parse_info_of_info i2
        else min

    (* for one-liner comment, there is no newline token before *)
    | Parser_php.T_COMMENT i2::xs ->
        if Ast_php.col_of_info i2 = 0
        then Ast_php.parse_info_of_info i2
        else min

    | Parser_php.T_WHITESPACE i1::tok2::xs ->
        let _ltok2 = Token_helpers_php.line_of_tok tok2 in
        let _lwhite = Ast_php.line_of_info i1 in
        if Ast_php.col_of_info i1 = 0 (* buggy: lwhite <> ltok2 *)
        then (* safe to remove the previous newline too *)
          Ast_php.parse_info_of_info i1
        else min
    | _ -> min
  in

  let minline = min.line in
  let maxline = max.line in
  let minline_comment = min_comment.line in
  { just_code_lines = enum minline maxline;
    code_and_comment_lines = enum minline_comment maxline
  }
)
+> (fun couples ->
  let just_code = couples +> List.map (fun x -> x.just_code_lines) in
  let all_lines = couples +> List.map (fun x -> x.code_and_comment_lines) in
  { just_code_lines        = just_code +> List.flatten;
```

```
      code_and_comment_lines = all_lines +> List.flatten;
    })

(*****************************************************************************)
(* VCS related Helpers *)
(*****************************************************************************)

(*
 * Return which person(s) to blame for some deadcode (in fact certain lines).
 * Do majority, except a whitelist, and if nothing found then
 * do majority of file, and if nothing found (because of whitelist)
 * then say "NOBODYTOBLAME"
 *
 * One improvement suggested by sgrimm is to use git annotate -C (or
 * git blame -C) which tries to detect move of code and give a more
 * accurate author. See h_version-control/git.ml.
 *
 * For instance on www/lib/common.php,
 * git annotate -C   vs git annotate gives:
 *
 * 138,147c138,147
 * < 2ea63cc5   (  jwiseman     2007-07-03 01:39:41 +0000       138) *
 * < d6106bdb   (  jwiseman     2007-07-05 21:58:37 +0000       139) * @param  int       $
 * < d6106bdb   (  jwiseman     2007-07-05 21:58:37 +0000       140) * @param  string    $
 * < d6106bdb   (  jwiseman     2007-07-05 21:58:37 +0000       141) * @param  array     $
 * < 2ea63cc5   (  jwiseman     2007-07-03 01:39:41 +0000       142) * @return resource  a
 * < 2ea63cc5   (  jwiseman     2007-07-03 01:39:41 +0000       143) * @author jwiseman
 * < 2ea63cc5   (  jwiseman     2007-07-03 01:39:41 +0000       144) */
 * < d6106bdb   (  jwiseman     2007-07-05 21:58:37 +0000       145)function require_write
 * < 2ea63cc5   (  jwiseman     2007-07-03 01:39:41 +0000       146)  $conn_w = id_get_cor
 * < 2ea63cc5   (  jwiseman     2007-07-03 01:39:41 +0000       147)  if (!$conn_w) {
 * ---
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       138) *
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       139) * @param  int       $
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       140) * @param  string    $
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       141) * @param  array     $
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       142) * @return resource  a
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       143) * @author jwiseman
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       144) */
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       145)function require_write
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       146)  $conn_w = id_get_cor
 * > effa6f73   (    mcslee     2007-10-18 06:43:09 +0000       147)  if (!$conn_w) {
 *
 * It is clear that the first series of blame is better, as
 * it contains multiple commits, and because mcslee was probably just
 * moving code around and not actually modifying the code.
```

```
 *
 * Note that by default git blame does already some analysis such as
 * detecting renaming of files. But it does not do more than that. For
 * intra files moves, you want git annotate -C.
 *
 * With -C it takes 130min to run the deadcode analysis on www.
 * Fortunately once it's cached, it takes only 2 minutes.
 *
 *)
let get_blamers ~prj_path ~filename ~filename_in_project ~hooks
    lines_to_remove =

  (* git blame is really slow, so cache its result *)
  let annots =
    Common.cache_computation ~use_cache:hooks.cache_git_blame
      filename ext_git_annot_cache (fun () ->
      Git.annotate ~basedir:prj_path filename_in_project
    ) in

  let toblame =
    lines_to_remove +> List.map (fun i ->
      let (version, Lib_vcs.Author author, date) = annots.(i) in
      author
    )
    +> List.filter hooks.is_valid_author
  in

  let hblame = Common.hashset_of_list toblame in
  let other_authors =
    annots +> Array.to_list +> List.map (fun x ->
      let (version, Lib_vcs.Author author, date) = x in
      author
    )
    +> List.filter (fun x ->
      hooks.is_valid_author x && not (Common.hmem x hblame)
    )
  in

  let counts = Common.count_elements_sorted_highfirst toblame +>
    List.map fst in
  let counts' = Common.count_elements_sorted_highfirst other_authors +>
    List.map fst in

  match counts ++ counts' with
  | a::b::_ -> Common.join "_" [a;b]
  | a::_ ->
```

```
      pr2 "Just found a single author :(";
      a
  | [] -> "NOBODYTOBLAME"


(*
 * We dont want to generate patches for very recent code, because
 * people may actually add code that is not yet called but will in the futur.
 * Moreover recent code changes so our built database and patches
 * may not apply anymore once the reviewer get the email.
 * Finally we would prefer in a first round to generate patches only
 * for old code, as it has more chance to be true deadcode.
 *
 * For all those reasons, it is useful to know the "date" of the patch
 * we will generate.
 *
 * For now we use the date of the lines of code we will remove.
 * todo? we could use instead the lines of the whole file ?
 *
 *)
let get_date ~prj_path ~filename ~filename_in_project ~hooks
    lines_to_remove =

  (* git blame is really slow, so cache its result *)
  let annots =
    Common.cache_computation ~use_cache:hooks.cache_git_blame
      filename ext_git_annot_cache (fun () ->
      Git.annotate ~basedir:prj_path filename_in_project
    ) in

  (* todo? use only the lines_to_remove or the whole file to
   * decide of the "date" of the patch ? *)
  let toblame =
    lines_to_remove +> List.map (fun i ->
      let (version, Lib_vcs.Author author, date) = annots.(i) in
      date
    )
  in
  Common.maximum_dmy toblame


(*****************************************************************************)
(* Deadcode patches *)
(*****************************************************************************)

let generate_deadcode_patch ~prj_path ~filename ~filename_in_project
```

```
    ~hooks ~lines_to_remove xs' =

  let blame = get_blamers ~prj_path ~filename ~filename_in_project
    ~hooks lines_to_remove.just_code_lines
  in
  let date = get_date ~prj_path ~filename ~filename_in_project
    ~hooks lines_to_remove.just_code_lines
  in

  let (dir, base) = Common.db_of_filename filename_in_project in
  let finaldir = Filename.concat hooks.patches_path dir in
  pr2(spf "Blaming %s for file %s" blame filename_in_project);
  Common.command2("mkdir -p " ^ finaldir);

  let patchfile = spf "%s/%s__%s__%s.patch"
    finaldir
    blame
    (Common.string_of_date_dmy date)
    base
  in
  pr2 ("Generating: " ^ patchfile);
  Common.uncat xs' patchfile;
  ()
```

```
(* assume the filename of the patch was generated by function above *)
let deadcode_patch_info patch =
  let base = Filename.basename patch in
  if base =~ "\\(.+\\)__\\(.+\\)__.*"
  then
    let (blame, date) = Common.matched2 base in

    let blamers = Common.split "_" blame in
    let (reviewer, cc) =
      match blamers with
      | ["NOBODYTOBLAME"] -> None, None
      | [x] -> Some x, None
      | [x;y] -> Some x, Some y
      | _ -> raise Impossible
    in
    let first_line = List.hd (Common.cat patch) in
    (* e.g. "--- a/flib/gender/gender.php\t" *)
    if (first_line =~ "--- a/\\([^ \t\n]+\\)")
    then
```

```
      let file = Common.matched1 first_line in
      {
        file = file;
        reviewer = reviewer;
        cc = cc;
        date = Common.date_dmy_of_string date;
      }
    else
      failwith ("WIERD: content of deadcode patch seems invalid: " ^ first_line)
  else
    failwith ("WIERD: the filename does not respect the convention: " ^ patch)

(*****************************************************************************)
(* Extra actions *)
(*****************************************************************************)

(* who has the most deadcode :) *)
let deadcode_stat dir =
  let files =
    Common.files_of_dir_or_files_no_vcs_post_filter "" [dir ^ "/"] in
  let h = Hashtbl.create 101 in

  files +> List.iter (fun file ->

    let info = deadcode_patch_info file in
    let nblines = Common.cat file +> List.length in
    let author = Common.some_or info.reviewer "NOBODYTOBLAME" in
    Common.hupdate_default author (fun old -> old + nblines)
      Common.cst_zero  h

  (* old: Common.command2(spf "mv \"%s\" \"%s.patch\" " file file); *)
  );
  let xs = Common.hash_to_list h in
  let sorted = Common.sort_by_val_highfirst xs in
  sorted +> List.iter (fun (author, count) ->
    pr2(spf "%15s = %d lines of dead code" author count);
  );
  let total = sorted +> List.map snd +> Common.sum_int in
  pr2 (spf "total = %d" total);
  ()



let cleanup_cache_files dir =
  let cache_ext = [ext_git_annot_cache] in
  cache_ext +> List.iter (fun ext ->
```

```
    let files = Common.files_of_dir_or_files_no_vcs ext [dir] in
    files +> List.iter (fun file ->
      assert(Common.filesuffix file = ext);
      pr2 file;
      Common.command2(spf "rm -f %s" file);
    ));
  ()


(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)

let deadcode_analysis hooks db =

  let dead_ids = finding_dead_functions hooks db in

  let prj_path = Db.path_of_project db.Db.project in

  (* grouping, blaming, and generate diffs *)
  let grouped_by_file =
    dead_ids
    +> List.map (fun (s, id) -> s, db.Db.fullid_of_id#assoc id, id)
    +> Common.group_by_mapped_key (fun (s,fullid,id) -> fullid.Entity_php.file)
  in

  (* can remove that code below, because most of the time
   * it's because tbgs/glimpse is not aware of diffs between
   * foocall and $foocall or diffs between foocall and foocall_extra_stuff.
   * Test tbgs on 'jobs_get_mba_schools' and it will return also
   * files having 'jobs_get_mba_schools_names'
   *
   * This reduces the LOC of deadcode patches from 72000 to 52000
   *)
  (*
  let grouped_by_file =
    if false

    grouped_by_file +> List.filter (fun (file, ids) ->
      if ids +> List.exists (fun (s, fullid, id) ->
        let nbmatch = List.length (DbQ.glimpse_get_matching_files s db) in
        if not (nbmatch >= 1)
        then pr2_once "Have you run -index_glimpse ? ";

        if nbmatch > 1
        then pr2("tbgs/glimpse return multiple matches for: " ^ s);
```

```
      nbmatch > 1
    )
    then begin
      pr2("so for the moment ignoring file: " ^ file);
      false
    end
    else
      true
  )
in
*)
let grouped_by_file =
  deadcode_fixpoint_per_file
    grouped_by_file
    (dead_ids +> List.map snd)
    hooks db
in

if hooks.with_blame then begin
  if not (Common.command2_y_or_no("rm -rf " ^ hooks.patches_path))
  then failwith "ok we stop";
  Common.command2("mkdir -p " ^ hooks.patches_path);
end;

grouped_by_file +> List.iter (fun (filename, ids) ->
  let file =
    Common.cat filename +> Common.index_list_1 in
  let filename_in_project =
    Common.filename_without_leading_path prj_path filename in

  let lines_to_remove =
    get_lines_to_remove db ids  in
  let file' =
    file +> Common.exclude (fun (line, idx) ->
      List.mem idx lines_to_remove.code_and_comment_lines)
    +> List.map fst
  in

  (* generating diff *)
  let tmpfile = Common.new_temp_file "pfff" ".php" in
  write_file ~file:tmpfile (Common.unlines file');
  (* pr2 filename_in_project; *)
  let xs =
    Common.cmd_to_list (spf "diff -u -p  \"%s\" \"%s\"" filename tmpfile)
  in
```

```
  (* less: use h_version-control/patch.ml helper ? *)
  let xs' = xs +> List.map (fun s ->
    match () with
    | _ when  s =~ "^\\-\\-\\- \\([^ \t]+\\)\\([ \t]?\\)" ->
        let (_, rest) = matched2 s in
        "--- a/" ^ filename_in_project ^ rest
    | _ when  s =~ "^\\+\\+\\+ \\([^ \t]+\\)\\([ \t]?\\)" ->
        let (_, rest) = matched2 s in
        "+++ b/" ^ filename_in_project ^ rest
    | _ -> s
  )
  in

  (* generating patch *)
  if hooks.with_blame
  then generate_deadcode_patch ~prj_path ~filename ~filename_in_project
    ~hooks ~lines_to_remove xs'
  else
    if hooks.print_diff then xs' +> List.iter pr
)
```

Chapter 21

Tainted string analysis

109 ⟨*tainted_php.ml* 109⟩≡
⟨*Facebook copyright* 11⟩

```
(* Contributions by Alok Menghrajani *)

open Common

open Ast_php

module Ast = Ast_php

module D = Dataflow_php
module F = Controlflow_php

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*****************************************************************************)
(* Types *)
(*****************************************************************************)

type tainted = bool

type env = tainted D.env

type inout = tainted D.inout

(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)
```

```
let (tainted_analysis: F.flow -> tainted D.mapping) = fun flow ->
  raise Todo

(*****************************************************************************)
(* Giving warnings about dangerous code  *)
(*****************************************************************************)

let (check_bad_echo: F.flow -> tainted D.mapping -> unit) =
 fun flow mapping ->
   raise Todo


(*****************************************************************************)
(* Debugging *)
(*****************************************************************************)

let (display_tainted_flow: Controlflow_php.flow -> tainted Dataflow_php.mapping -> unit) =
 fun flow mapping ->
   raise Todo
```

Chapter 22

PHP Built-in Functions, Classes, and Globals

111 ⟨*builtins_php.ml* 111⟩≡
⟨*Facebook copyright* 11⟩

```
open Common

open Ast_php

module Flag = Flag_analyze_php
module Ast = Ast_php

module V = Visitor_php

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * As opposed to OCaml or C++ or Java or most programming languages,
 * there is no source code files where PHP builtin functions
 * and their types are declared (they are defined in the PHP manual only).
 * In OCaml most library functions are written in OCaml itself or are specified
 * via an 'external' declaration as in:
 *
 *   external (=) : 'a -> 'a -> bool = "%equal"
 *
 * This is very convenient for certain tasks such as code browsing where
 * many functions would be seen as 'undefined' otherwise, or for
 * the type inference  where we have info about the basic functions.
```

```
 *
 * Unfortunately, this is not the case for PHP. Fortunately the
 * good guys from HPHP have spent the time to specify in a IDL form
 * the interface of all those builtin PHP functions (including the
 * one in some popular PHP extensions). They used it to
 * generate C++ header files, but we can abuse it to instead generate
 * PHP "header" files that our tool can understand.
 *
 * Here is for instance the content of one such IDL file,
 * hphp/src/idl/math.idl.php:
 *
 *      f('round',   Double,  array('val' => Variant,
 *                          'precision' => array(Int64, '0')));
 *
 * which defines the interface for the 'round' builtin math function.
 * In the manual at http://us3.php.net/round is is defined a:
 *
 *     float round  ( float $val  [, int $precision = 0] )
 *
 * So the only job of this module is to generate from the IDL file
 * some PHP code like:
 *
 *    function round($val, $precision = 0) {
 *       // THIS IS AUTOGENERATED BY builtins_php.ml
 *    }
 *
 * or even better:
 *
 *    function round(Variant $val, int $precision = 0) double {
 *       // THIS IS AUTOGENERATED BY builtins_php.ml
 *    }
 *
 * Alternatives: could also define in PHP a f() function that would
 * do the appropriate things, just like what they do for generating
 * C++ headers, but then you have to know PHP to do that :) Moreover
 * here I have defined a type for idl_entry. Types are good!
 *)

(*****************************************************************************)
(* Types *)
(*****************************************************************************)

(* see hphp/src/idl/base.php. generated mostly via macro *)
type idl_type =
  | Boolean
```

```
  (* maybe not used *)
  | Byte
  | Int16

  | Int32
  | Int64
  | Double
  | String

  | Int64Vec
  | StringVec
  | VariantVec

  | Int64Map
  | StringMap
  | VariantMap

  | Object
  | Resource

  | Variant

  | Numeric
  | Primitive
  | PlusOperand
  | Sequence

  | Any

  (* Added by me *)
  | NULL
  | Void


type idl_param = {
  p_name: string;
  p_type: idl_type;
  p_isref: bool;
  p_default_val: string option;
}

type idl_entry =
  | Global of string * idl_type
  | Function of
      string * idl_type * idl_param list * bool (* has variable arguments *)
```

```
let special_comment =
  "// THIS IS AUTOGENERATED BY builtins_php.ml\n"
let builtins_do_not_give_decl =
  ["die";"eval";"exit";"__halt_compiler";"echo"; "print"]

(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)

let idl_type__str_conv = [
  Boolean     , "Boolean";
  Byte        , "Byte";
  Int16       , "Int16";
  Int32       , "Int32";
  Int64       , "Int64";
  Double      , "Double";
  String      , "String";
  Int64Vec    , "Int64Vec";
  StringVec   , "StringVec";
  VariantVec  , "VariantVec";
  Int64Map    , "Int64Map";
  StringMap   , "StringMap";
  VariantMap  , "VariantMap";
  Object      , "Object";
  Resource    , "Resource";
  Variant     , "Variant";
  Numeric     , "Numeric";
  Primitive   , "Primitive";
  PlusOperand , "PlusOperand";
  Sequence    , "Sequence";
  Any         , "Any";

  NULL        ,  "NULL";
  NULL        ,  "Null";
]

let (idl_type_of_string, str_of_idl_type) =
 Common.mk_str_func_of_assoc_conv idl_type__str_conv

let idl_type_of_string (s, info) =
    try idl_type_of_string s
    with Not_found ->
      let pinfo = Ast.parse_info_of_info info in
      pr2 (Common.error_message_info pinfo);
      failwith ("not a idl type: " ^ s)
```

```
(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)
exception NotValidIdlPhpEntry


let rec idl_type_of_ast (x : expr) =
  match untype x with
  | Scalar (Constant (CName (Name (str_typ)))) ->
      idl_type_of_string str_typ, false
  | Binary (e1, (Arith Or, _), e2)  ->

      let typ, is_ref2 = idl_type_of_ast e1 in
      assert(not is_ref2);
      let is_ref =
      (match untype e2 with
      | (Scalar (Constant (CName (Name ("Reference", _))))) ->
          true
      | _ -> raise NotValidIdlPhpEntry
      )
      in
      typ, is_ref

  | _ -> raise NotValidIdlPhpEntry

let idl_arg_to_arg (arg: array_pair) =
  (* old: "$param" *)
  match arg with
  | ArrayArrowExpr ((Scalar (Constant (Ast.String (str_param, _))), _), _,
                    targ) ->

        (match targ with
        | (ConsArray (_, (_,array_pairs,_)),_) ->
            (match array_pairs with
            |   ArrayExpr (typ)
             :: ArrayExpr (Scalar (Constant (Ast.String (str_expr, _))),_)
             ::[] ->
                  let (typ, isref) = idl_type_of_ast typ in
                  {
                    p_name = str_param;
                    p_type = typ;
                    p_isref = isref;
                    p_default_val = Some str_expr;
                  }
            | _ -> raise NotValidIdlPhpEntry
```

```
            )
        | typ ->

            let (typ, isref) = idl_type_of_ast typ in
            {
              p_name = str_param;
              p_type = typ;
              p_isref = isref;
              p_default_val = None;
            }
        )
  | _ -> raise NotValidIdlPhpEntry



let ast_args_to_idl_entry (args: expr list) =
  match args with
  | [] -> raise NotValidIdlPhpEntry
  | x::xs ->
      (match x with
      | (Scalar (Constant (Ast.String (var_or_func, _))), _) ->
          (match xs with
          | [] ->
              Function(var_or_func, Void, [], false)

          | [typ] ->
              let (typ, isref) = idl_type_of_ast typ in
              assert(not isref);
              Function (var_or_func, typ, [], false)

          | return_type::spec_args ->
              (match spec_args with
              | (ConsArray (_, (_,array_pairs,_)),_)::maybe_variable_args ->

                  let args = array_pairs +> List.map idl_arg_to_arg in
                  (* TODO: func extra info *)
                  let variable_arg =
                    match maybe_variable_args with
                    | [] -> false
                    | [(Scalar (Constant (CName (Name (attr_str,_)))), _)]
                        ->
                        (match attr_str with
                        | "VariableArguments"
                        | "ReferenceVariableArguments"
                            -> true
                        | "NoEffect" -> false (* ignore for now *)
```

```
                        | _ -> raise NotValidIdlPhpEntry
                        )

                    | _ -> raise NotValidIdlPhpEntry
                in
                let (typ, isref) = idl_type_of_ast return_type in
                assert(not isref);
                Function(var_or_func, typ, args, variable_arg)
            | _ -> raise NotValidIdlPhpEntry
            )
        )
    | _ -> raise NotValidIdlPhpEntry
    )



(*****************************************************************************)
(* Main entry points *)
(*****************************************************************************)

let ast_php_to_idl toplevels =
  match toplevels with
  | [StmtList xs; FinalDef _] ->
      xs +> Common.map_filter (fun stmt ->
        try (
          match stmt with
          | ExprStmt ((IncludeOnce _,_), _) ->
              None

          | ExprStmt ((Lvalue ((FunCallSimple (_, (Name ("f",info)), args),_)),_), _) ->
              let args' = unparen args +> List.map (function
                | Arg e -> e
                | ArgRef _ -> raise NotValidIdlPhpEntry
              )
              in
              Some (ast_args_to_idl_entry args')

          (* TODO class *)
          | ExprStmt ((Lvalue ((FunCallSimple (_, (Name ("c",info)), args),_)),_), _) ->
              pr2 "TODO: Not handling 'c' idl decl";
              None

          (* TODO constant *)
          | ExprStmt ((Lvalue ((FunCallSimple (_, (Name ("k",info)), args),_)),_), _) ->
              pr2 "TODO: Not handling 'k' idl decl";
```

```
              None



            | ExprStmt ((Lvalue ((FunCallSimple (_, (Name ("dyn",info)), args),_)),_), _) ->
                pr2 "TODO: Not handling 'dyn' idl decl";
                None

            | ExprStmt ((Lvalue ((FunCallSimple (_, (Name ("p",info)), args),_)),_), _) ->
                pr2 "TODO: Not handling 'p' idl decl";
                None


            | _ -> raise NotValidIdlPhpEntry
          )
          with NotValidIdlPhpEntry ->
            let ii = Lib_parsing_php.ii_of_stmt stmt in
            let info, _max = Lib_parsing_php.min_max_ii_by_pos ii in
            let pinfo = Ast.parse_info_of_info info in
            pr2 ("PB:" ^ (Common.error_message_info pinfo));
            raise NotValidIdlPhpEntry
        )
    | _ -> failwith "does not look like a idl.php file"



let idl_entry_to_php_fake_code entry =
  match entry with
  | Global (var, _t) ->
      spf "// Global: $%s\n" var

  | Function (func, _typTODO, params, variable_arg) ->
      if List.mem func builtins_do_not_give_decl
      then ""
      else

      let params_str = params +> List.map (fun param ->
        let typ_str =
          str_of_idl_type param.p_type
        in
        let ref_str =
          if param.p_isref then "&" else ""
        in
        let default_str =
          match param.p_default_val with
          | None -> ""
```

```
          | Some s -> spf " = %s" s
        in
        (* cf non_empty_parameter_list grammar rule in parser_php.mly *)
        spf "%s %s$%s%s" typ_str ref_str param.p_name default_str
      )
      in
      let str_variable_arg =
        if variable_arg
        then "$args = func_num_args(); // fake code to say variable #args\n"
        else ""
      in

      spf "function %s(%s) {\n %s %s \n}\n"
        func
        (Common.join ", " params_str)
        special_comment
        str_variable_arg



(*****************************************************************************)
(* Action *)
(*****************************************************************************)

(* Generating stdlib from idl files *)
let generate_php_stdlib src dest =
  let files = Lib_analyze_php.find_php_files src in

  if not (Common.command2_y_or_no("rm -rf " ^ dest))
  then failwith "ok we stop";
  Common.command2("mkdir -p " ^ dest);

  files +> List.iter (fun file ->
    pr2 (spf "processing: %s" file);

    if (not (file =~ ".*.idl.php"))
    then failwith "Files does not end in .idl.php";

    let (ast2,_stat) = Parse_php.parse file in
    let asts = Parse_php.program_of_program2 ast2 in

    let base = Filename.basename file in
    let target = Filename.concat dest ("builtins_" ^ base) in
    Common.with_open_outfile target (fun (pr, chan) ->
      (* bugfix: dont forget that :) thx erling *)
      pr "<?php\n";
```

```
      let idl_entries =
        ast_php_to_idl asts
      in
      idl_entries +> List.iter (fun idl ->
        let s = idl_entry_to_php_fake_code idl in
        pr s
      )
    )
  );
  ()


let actions () = [
  "-generate_php_stdlib", "<src> <dest>",
  Common.mk_action_2_arg generate_php_stdlib;
]
```

Chapter 23

PHPUnit

121 ⟨*phpunit.ml* 121⟩≡
⟨*Facebook copyright* 11⟩

Chapter 24

Auxillary Code

Conclusion

Appendix A

Testing sample code

124a ⟨*function common_parse_and_type* 124a⟩≡

```
let common_parse_and_type file =
  let (ast2,_stat) = Parse_php.parse file in
  let ast = Parse_php.program_of_program2 ast2 in
  (*
  Type_annoter_c.annotate_program
    !Type_annoter_c.initial_env ast +> ignore;
  *)
  ast
```

124b ⟨*test_analyze_php.ml* 124b⟩≡

```
open Common

module Db = Database_php

(*****************************************************************************)
(* Subsystem testing *)
(*****************************************************************************)
```

⟨*function common_parse_and_type* 124a⟩

```
(* ---------------------------------------------------------------------- *)
let test_check_php file =
  let ast = common_parse_and_type file in
  Checking_php.check_program ast

(* ---------------------------------------------------------------------- *)
let test_type_php file =
  let asts = common_parse_and_type file in

  let env = ref (Hashtbl.create 101) in
  let asts = asts +> List.map (fun ast ->
```

```
      Typing_php.annotate_toplevel env ast
    )
  in

  Sexp_ast_php.show_expr_info := true;
  pr (Sexp_ast_php.string_of_program asts);
  ()

(* ---------------------------------------------------------------------- *)
let test_scope_php file =
  let asts = common_parse_and_type file in
  let asts = asts +> List.map Scoping_php.annotate_toplevel in

  Sexp_ast_php.show_expr_info := true;
  pr (Sexp_ast_php.string_of_program asts);
  ()

(* ---------------------------------------------------------------------- *)
let test_typing_weak_php file =
  let asts = common_parse_and_type file in
  asts +> List.iter (fun ast ->
    let xs = Typing_weak_php.extract_fields_per_var ast in
    pr2_gen xs
  )

(* ---------------------------------------------------------------------- *)
let test_topological_sort () =
  raise Todo

(* ---------------------------------------------------------------------- *)
let test_idl_to_php file =
  let asts = common_parse_and_type file in
  let idl_entries =
    Builtins_php.ast_php_to_idl asts
  in
  idl_entries +> List.iter (fun idl ->
    let s = Builtins_php.idl_entry_to_php_fake_code idl in
    pr s
  )


(* ---------------------------------------------------------------------- *)
let test_dependencies_php metapath =
  Database_php.with_db metapath (fun db ->
    Dependencies_php.dir_to_dir_dependencies db
  )
```

```
(* ------------------------------------------------------------------------ *)
let test_function_pointer_analysis metapath =
  Database_php.with_db metapath (fun db ->

    (* move more code in aliasing_function_php.ml ? *)
    let h = Hashtbl.create 101 in

    Database_php_build.iter_files_and_topids db "FPOINTER" (fun id file ->
      let ast = db.Db.defs.Db.toplevels#assoc id in
      let funcvars = Lib_parsing_php.get_all_funcvars_ast ast in
      funcvars +> List.iter (fun dvar ->
        pr2 dvar;
        let prefixes =
          Aliasing_function_php.finding_function_pointer_prefix dvar ast in
        prefixes +> List.iter (fun s ->
          pr2(spf " '%s'" s);
          Hashtbl.replace h s true;
        );
      )
    );
    pr2 "dangerous prefixes:";
    h +> Common.hashset_to_list +> List.iter pr2;
  )


(* ------------------------------------------------------------------------ *)
let test_visit2_php file =
  let (ast2,_stat) = Parse_php.parse file in
  let ast = Parse_php.program_of_program2 ast2 in

  let hooks = { Visitor2_php.default_visitor with

    Visitor_php.klvalue = (fun (k, vx) e ->
      match fst e with
      | Ast_php.FunCallSimple (qu_opt, callname, args) ->
          let s = Ast_php.name callname in
          pr2 ("calling: " ^ s);

      | _ -> k e
    );
  } in
  let visitor = Visitor2_php.mk_visitor hooks in
  ast +> List.iter visitor.Visitor2_php.vorigin.Visitor_php.vtop

(* ------------------------------------------------------------------------ *)
```

```
⟨test_cfg_php 23c⟩
(* ------------------------------------------------------------------------ *)
⟨test_cyclomatic_php 51b⟩

(* ------------------------------------------------------------------------ *)
let test_tainted_php file =
  raise Todo

(* ------------------------------------------------------------------------ *)
let test_xdebug_dumpfile file =
  file +> Xdebug.iter_dumpfile (fun acall ->
    (* pr2 s *)
    ()
  )

(*****************************************************************************)
(* Main entry for Arg *)
(*****************************************************************************)

(* Note that other files in this directory define some cmdline actions:
 *  - database_php_build.ml
 *
 *)

let actions () = [
  ⟨test_analyze_php actions 23b⟩

  "-check_php", " <file>",
  Common.mk_action_1_arg test_check_php;
  "-type_php", " <file>",
  Common.mk_action_1_arg test_type_php;
  "-scope_php", " <file>",
  Common.mk_action_1_arg test_scope_php;

  "-tainted_php", " <file>",
  Common.mk_action_1_arg test_tainted_php;

  "-visit2_php", "   <file>",
    Common.mk_action_1_arg test_visit2_php;

  "-weak_php", " <file>",
  Common.mk_action_1_arg test_typing_weak_php;

  "-idl_to_php", " <file>",
  Common.mk_action_1_arg test_idl_to_php;
```

```
  "-dependencies_php", " <metapath>",
  Common.mk_action_1_arg test_dependencies_php;

  "-function_pointer_analysis", "<db>",
  Common.mk_action_1_arg (test_function_pointer_analysis);

  "-parse_xdebug_dumpfile", " <dumpfile>",
  Common.mk_action_1_arg test_xdebug_dumpfile;

]
```

Appendix B

Extra code

B.1 aliasing_function_php.mli

129a ⟨*aliasing_function_php.mli* 41⟩+≡

B.2 analysis_dynamic_php.mli

129b ⟨*analysis_dynamic_php.mli* 45a⟩+≡

B.3 analysis_static_php.mli

129c ⟨*analysis_static_php.mli* 42a⟩+≡

B.4 annotation_php.mli

129d ⟨*annotation_php.mli* 29a⟩+≡

B.5 ast_entity_php.mli

129e ⟨*ast_entity_php.mli* 35a⟩+≡

```
open Ast_php

type id_ast =
  | Function of func_def
  | Class of class_def
  | Interface of interface_def
  | StmtList of stmt list
```

```
  | Method of method_def
  | ClassConstants of tok * class_constant list * tok
  | ClassVariables of
      class_var_modifier * class_variable list * tok

  | Misc of info list
 (* with tarzan *)

val toplevel_to_idast: toplevel -> id_ast
```

B.6 bottomup_analysis_php.mli

130a ⟨*bottomup_analysis_php.mli* 39d⟩+≡

B.7 builtins_php.mli

130b ⟨*builtins_php.mli* 39e⟩+≡

B.8 callgraph_php.mli

130c ⟨*callgraph_php.mli* 36⟩+≡

B.9 checking_php.mli

130d ⟨*checking_php.mli* 48⟩+≡

B.10 code_rank_php.mli

130e ⟨*code_rank_php.mli* 50b⟩+≡

B.11 comment_annotater_php.mli

130f ⟨*comment_annotater_php.mli* 28c⟩+≡

B.12 controlflow_php.mli

130g ⟨*controlflow_php.mli* 21a⟩+≡

B.13 controlflow_build_php.mli

131a ⟨*controlflow_build_php.mli* 21b⟩+≡

B.14 database_php.mli

131b ⟨*database_php.mli* 30⟩+≡

B.15 database_php_build.mli

131c ⟨*database_php_build.mli* 33⟩+≡

B.16 database_php_query.mli

131d ⟨*database_php_query.mli* 46b⟩+≡

B.17 database_php_statistics.mli

131e ⟨*database_php_statistics.mli* 52a⟩+≡

B.18 dataflow_php.mli

131f ⟨*dataflow_php.mli* 26e⟩+≡

B.19 deadcode_php.mli

131g ⟨*deadcode_php.mli* 42b⟩+≡

B.20 dependencies_php.mli

131h ⟨*dependencies_php.mli* 39a⟩+≡

B.21 entities_php.mli

131i ⟨*entities_php.mli* 35b⟩+≡

B.22 entity_php.mli

132a ⟨*entity_php.mli* 34⟩+≡

B.23 finder_php.mli

132b ⟨*finder_php.mli* 47a⟩+≡

B.24 freevars_php.mli

132c ⟨*freevars_php.mli* 27b⟩+≡

B.25 graph_php.mli

132d ⟨*graph_php.mli* 39c⟩+≡

B.26 include_require_php.mli

132e ⟨*include_require_php.mli* 39b⟩+≡

B.27 info_annotater_php.mli

132f ⟨*info_annotater_php.mli* 28d⟩+≡

B.28 lib_analyze_php.mli

132g ⟨*lib_analyze_php.mli* 46a⟩+≡

B.29 namespace_php.mli

132h ⟨*namespace_php.mli* 27c⟩+≡

B.30 normalize_php.mli

132i ⟨*normalize_php.mli* 29b⟩+≡

B.31 scoping_php.mli

133a ⟨*scoping_php.mli* 27a⟩+≡

B.32 smpl_php.mli

133b ⟨*smpl_php.mli* 47b⟩+≡

B.33 statistics_php.mli

133c ⟨*statistics_php.mli* 50a⟩+≡

B.34 tainted_php.mli

133d ⟨*tainted_php.mli* 44b⟩+≡

B.35 test_analyze_php.mli

133e ⟨*test_analyze_php.mli* 52b⟩+≡

B.36 type_annotater_php.mli

133f ⟨*type_annotater_php.mli* 43a⟩+≡

B.37 typing_php.mli

133g ⟨*typing_php.mli* 43b⟩+≡

B.38 typing_trivial_php.mli

133h ⟨*typing_trivial_php.mli* 28a⟩+≡

B.39 typing_weak_php.mli

133i ⟨*typing_weak_php.mli* 44a⟩+≡

B.40 visitor2_php.mli

134a ⟨*visitor2_php.mli* 29c⟩+≡

B.41 xdebug.mli

134b ⟨*xdebug.mli* 28b⟩+≡

Appendix C

Indexes

⟨*Facebook copyright* 11⟩
⟨*aliasing_function_php.mli* 41⟩
⟨*analysis_dynamic_php.mli* 45a⟩
⟨*analysis_static_php.mli* 42a⟩
⟨*annotation_php.mli* 29a⟩
⟨*ast_entity_php.mli* 35a⟩
⟨*bottomup_analysis_php.mli* 39d⟩
⟨*builtins_php.ml* 111⟩
⟨*builtins_php.mli* 39e⟩
⟨*callgraph_php.mli* 36⟩
⟨*checking_php.mli* 48⟩
⟨*code_rank_php.ml* 86⟩
⟨*code_rank_php.mli* 50b⟩
⟨*comment_annotater_php.mli* 28c⟩
⟨*controlflow builders* 59a⟩
⟨*controlflow builders signatures* 21c⟩
⟨*controlflow checkers signatures* 26a⟩
⟨*controlflow helpers signatures* 26d⟩
⟨*controlflow_build_php.ml* 57⟩
⟨*controlflow_build_php.mli* 21b⟩
⟨*controlflow_php accessors* 73d⟩
⟨*controlflow_php helpers* 71b⟩
⟨*controlflow_php main algorithm* 60⟩
⟨*controlflow_php.ml* 56⟩
⟨*controlflow_php.mli* 21a⟩
⟨*cyclomatic_php.ml* 91⟩
⟨*cyclomatic_php.mli* 50c⟩
⟨*database_php_build.mli* 33⟩
⟨*database_php.mli* 30⟩
⟨*database_php_query.mli* 46b⟩
⟨*database_php_statistics.mli* 52a⟩

⟨*dataflow_php.ml* 74⟩
⟨*dataflow_php.mli* 26e⟩
⟨*deadcode_php.ml* 93⟩
⟨*deadcode_php.mli* 42b⟩
⟨*dependencies_php.mli* 39a⟩
⟨*entities_php.mli* 35b⟩
⟨*entity_php.mli* 34⟩
⟨*error exception and report_error signature* 26c⟩
⟨*finder_php.mli* 47a⟩
⟨*freevars_php.mli* 27b⟩
⟨*function Controlflow_build_php.report_error* 73c⟩
⟨*function common_parse_and_type* 124a⟩
⟨*function deadcode_detection* 73b⟩
⟨*function display_flow* 70⟩
⟨*function display_flow signature* 23a⟩
⟨*function short_string_of_node* 71a⟩
⟨*graph_php.mli* 39c⟩
⟨*include_require_php.mli* 39b⟩
⟨*info_annotater_php.mli* 28d⟩
⟨*lib_analyze_php.mli* 46a⟩
⟨*namespace_php.mli* 27c⟩
⟨*node_kind aux types* 25f⟩
⟨*node_kind constructors* 24d⟩
⟨*normalize_php.mli* 29b⟩
⟨*phpunit.ml* 121⟩
⟨*phpunit.mli* 45b⟩
⟨*scoping_php.mli* 27a⟩
⟨*smpl_php.mli* 47b⟩
⟨*statistics_php.mli* 50a⟩
⟨*tainted_php.ml* 109⟩
⟨*tainted_php.mli* 44b⟩
⟨*test_analyze_php actions* 23b⟩
⟨*test_analyze_php.ml* 124b⟩
⟨*test_analyze_php.mli* 52b⟩
⟨*test_cfg_php* 23c⟩
⟨*test_cyclomatic_php* 51b⟩
⟨*tests/cfg/while.php* 20⟩
⟨*type Controlflow_build_php.error* 26b⟩
⟨*type edge* 24b⟩
⟨*type flow* 23d⟩
⟨*type node* 24a⟩
⟨*type nodei* 58a⟩
⟨*type node_kind* 24c⟩
⟨*type state* 58b⟩
⟨*type_annotater_php.mli* 43a⟩
⟨*typing_php.mli* 43b⟩

⟨*typing_trivial_php.ml* 76⟩
⟨*typing_trivial_php.mli* 28a⟩
⟨*typing_weak_php.mli* 44a⟩
⟨*visitor2_php.mli* 29c⟩
⟨*xdebug.ml* 78⟩
⟨*xdebug.mli* 28b⟩

Appendix D

References

Bibliography

[1] Donald Knuth,, *Literate Programming*, http://en.wikipedia.org/wiki/Literate_Program cited page(s) 17

[2] Norman Ramsey, *Noweb*, http://www.cs.tufts.edu/~nr/noweb/ cited page(s) 17

[3] Yoann Padioleau, *Syncweb, literate programming meets unison*, http://padator.org/software/project-syncweb/readme.txt cited page(s) 17

[4] Hannes Magnusson et al, *PHP Manual*, http://php.net/manual/en/index.php cited page(s)

[5] Alfred Aho et al, *Compilers, Principles, Techniques, and tools*, http://en.wikipedia.org/wiki/Dragon_Book_(computer_science) cited page(s) 17

[6] Andrew Appel, *Modern Compilers in ML*, Cambridge University Press cited page(s) 17, 20

[7] Yoann Padioleau, *Commons Pad OCaml Library*, http://padator.org/docs/Commons.pdf cited page(s)

[8] Yoann Padioleau, *OCamltarzan, code generation with and without camlp4*, http://padator.org/ocaml/ocamltarzan-0.1.tgz cited page(s)

[22] George Necula, *CIL*, CC. http://manju.cs.berkeley.edu/cil/ cited page(s)