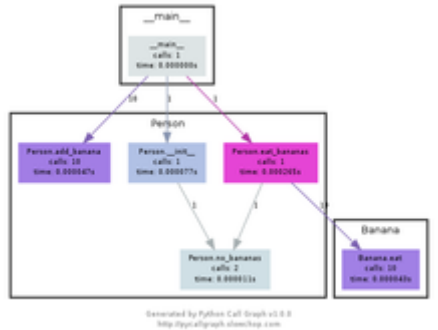
WIKIPEDIA

Call graph

A **call graph** (also known as a **call multigraph**[1]) is a control flow graph,[2] which represents calling relationships between subroutines in a computer program. Each node represents a procedure and each edge *(f, g)* indicates that procedure *f* calls procedure *g*. Thus, a cycle in the graph indicates recursive procedure calls.



A call graph generated for a simple computer program in Python.

Call graphs are a basic program analysis result that can be used for human understanding of programs, or as a basis for further analyses, such as an analysis that tracks the flow of values between procedures. One simple application of call graphs is finding procedures that are never called.

Call graphs can be dynamic or static. A dynamic call graph is a record of an execution of the program, for example as output by a profiler. Thus, a dynamic call graph can be exact, but only describes one run of the program. A static call graph is a call graph intended to represent every possible run of the program. The exact static call graph is an undecidable problem, so static call graph algorithms are generally overapproximations. That is, every call relationship that occurs is represented in the graph, and possibly also some call relationships that would never occur in actual runs of the program.

Call graphs can be defined to represent varying degrees of precision. A more precise call graph more precisely approximates the behavior of the real program, at the cost of taking longer to compute and more memory to store. The most precise call graph is fully *context-sensitive*, which means that for each procedure, the graph contains a separate node for each call stack that procedure can be activated with. A fully context-sensitive call graph is called a calling context tree. This can be computed dynamically easily, although it may take up a large amount of memory. Calling context trees are usually not computed statically, because it would take too long for a large program. The least precise call graph is *context-insensitive*, which means that there is only one node for each procedure.

With languages that feature dynamic dispatch, such as Java and C++, computing a static call graph precisely requires alias analysis results. Conversely, computing precise aliasing requires a call graph. Many static analysis systems solve the apparent infinite regress by computing both simultaneously.

This term is frequently used in the compiler and binary translation community. By tracking a call graph, it may be possible to detect anomalies of program execution or code injection attacks.

Contents

Software

Free software call-graph generators

Run-time call-graph (most of tools listed are profilers with callgraph functionality)

- gprof : included in BSD or part of the GNU Binary Utilities
- callgrind : part of Valgrind
- KCachegrind (https://kcachegrind.github.io/html/Home.html) : powerful tool to generate and analyze call graphs based on data generated by callgrind
- Mac OS X Activity Monitor : Apple GUI process monitor Activity Monitor has a built-in call graph generator that can sample processes and return a call graph. This function is only available in Mac OS X Leopard
- OpenPAT : includes the control_flow tool which automatically creates a Graphviz call-graph picture from runtime measurements.
- pprof (https://github.com/google/pprof), open source tool for visualization and analysis of profile data, to be used in conjunction with gperftools (https://gperftools.googlecode.com/git/doc/cpuprofile.html).
- CodeAnalyst from AMD (released under GPL)
- makeppgraph (http://makepp.sourceforge.net/gallery/) is a dependency graph generator (at module level) for builds performed with makepp.
- Intel(R) Single Event API (https://github.com/01org/IntelSEAPI/wiki) (free, open-source)

Static (for C language), for getting call graphs without running of application

- doxygen : Uses graphviz to generate static call/inheritance diagrams
- cflow : GNU cflow is able to generate the direct and inverted call graph of a C program
- egypt (http://www.gson.org/egypt/) : a small Perl script that uses gcc and Graphviz to generate the static call graph of a C program.
- CCTree (http://www.vim.org/scripts/script.php?script_id=2368) : Native Vim plugin that can display static call graphs by reading a cscope database. Works for C programs.
- codeviz (https://github.com/petersenna/codeviz) : a static call graph generator (the program is *not* run). Implemented as a patch to gcc; works for C and C++ programs.
- Cppdepend :is a static analysis tool for C/C++ code. This tool supports a large number of code metrics, allows for visualization of dependencies using directed graphs and dependency matrix.
- calltree.sh (http://toolchainguru.blogspot.com/2011/03/c-calltrees-in-bash-revisited.html) : Bash shell functions that glue together cscope, graphviz, and a sampling of dot-rendering tools to display "caller" and "callee" relationships above, below, and/or between the C functions you specify.
- tceetree (http://sourceforge.net/projects/tceetree/) : like calltree.sh, it connects Cscope and Graphviz, but it is an executable rather than a bash script.

Go

- go-callvis (https://github.com/TrueFurby/go-callvis) : a call graph generator for Go programs whose output can be drawn with Graphviz

.Net

- NDepend :is a static analysis tool for .Net code. This tool supports a large number of code metrics, allows for visualization of dependencies using directed graphs and dependency matrix.

PHP, Perl and Python

- Devel::NYTProf (https://metacpan.org/module/Devel::NYTProf) : a perl performance analyser and call chart generator
- phpCallGraph (http://phpcallgraph.sourceforge.net/) : a call graph generator for PHP programs that uses Graphviz. It is written in PHP and requires at least PHP 5.2.
- pycallgraph (http://pycallgraph.slowchop.com/) : a call graph generator for Python programs that uses Graphviz.
- pyan (https://github.com/davidfraser/pyan) : a static call graph generator for Python programs that uses Graphviz.
- gprof2dot (https://github.com/jrfonseca/gprof2dot) : A call graph generator written in Python that converts profiling data for many languages/runtimes to a Graphviz callgraph.
- code2flow (https://github.com/scottrogowski/code2flow): A call graph generator for Python and Javascript programs that uses Graphviz

Proprietary call-graph generators

Project Analyzer
: Static code analyzer and call graph generator for Visual Basic code

Visual Expert
: Static code analyzer and call graph generator for Oracle PL/SQL, SQLServer Transact-SQL, C# and PowerBuilder code

Intel VTune Performance Analyzer
: Instrumenting profiler to show call graph and execution statistics

DMS Software Reengineering Toolkit
: Customizable program analysis tool with static whole-program global call graph extraction for C, Java and COBOL

Other, related tools

Graphviz
: Turns a text representation of any graph (including a call graph) into a picture.

Sample graph

A sample call graph generated from gprof analyzing itself:

```
index  called     name                        |index  called     name
     72384/72384      sym_id_parse [54]       |      1508/1508       cg_dfn [15]
[3]   72384           match [3]               |[13]   1508           pre_visit [13]
----------------------------------------------|----------------------
       4/9052       cg_tally [32]            |      1508/1508       cg_assemble [38]
    3016/9052       hist_print [49]          |[14]   1508           propagate_time [14]
    6032/9052       propagate_flags [52]     |----------------------
[4]    9052         sym_lookup [4]           |         2            cg_dfn [15]
----------------------------------------------|      1507/1507       cg_assemble [38]
    5766/5766       core_create_function_syms [41]|[15]   1507+2         cg_dfn [15]
[5]    5766         core_sym_class [5]       |      1509/1509       is_numbered [9]
----------------------------------------------|      1508/1508       is_busy [11]
       24/1537      parse_spec [19]          |      1508/1508       pre_visit [13]
    1513/1537       core_create_function_syms [41]|      1508/1508       post_visit [12]
[6]    1537         sym_init [6]             |         2            cg_dfn [15]
----------------------------------------------|----------------------
    1511/1511       core_create_function_syms [41]|      1505/1505       hist_print [49]
[7]    1511         get_src_info [7]         |[16]   1505           print_line [16]
----------------------------------------------|         2/9          print_name_only [25]
       2/1510       arc_add [31]             |----------------------
    1508/1510       cg_assemble [38]         |      1430/1430       core_create_function_syms [41]
[8]    1510         arc_lookup [8]           |[17]   1430           source_file_lookup_path [17]
----------------------------------------------|----------------------
    1509/1509       cg_dfn [15]              |        24/24         sym_id_parse [54]
[9]    1509         is_numbered [9]          |[18]    24            parse_id [18]
----------------------------------------------|        24/24         parse_spec [19]
    1508/1508       propagate_flags [52]     |----------------------
[10]   1508         inherit_flags [10]       |        24/24         parse_id [18]
----------------------------------------------|[19]    24            parse_spec [19]
    1508/1508       cg_dfn [15]              |        24/1537       sym_init [6]
[11]   1508         is_busy [11]             |----------------------
----------------------------------------------|        24/24         main [1210]
    1508/1508       cg_dfn [15]              |[20]    24            sym_id_add [20]
[12]   1508         post_visit [12]          |
```

See also

- Dependency graph

References

1. Uday Khedker; Amitabha Sanyal; Bageshri Sathe (2009). *Data Flow Analysis: Theory and Practice*. CRC Press. p. 234. ISBN 978-0-8493-3251-7.
2. Pankaj Jalote (1997). *An Integrated Approach to Software Engineering*. Springer Science & Business Media. p. 372. ISBN 978-0-387-94899-7.

- Ryder, B.G., "Constructing the Call Graph of a Program," Software Engineering, IEEE Transactions on, vol. SE-5, no.3pp. 216– 226, May 1979 [1] (http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=35910&arnumber=1702621&count=17&index=5)
- Grove, D., DeFouw, G., Dean, J., and Chambers, C. 1997. Call graph construction in object-oriented languages. SIGPLAN Not. 32, 10 (Oct. 1997), 108-124. [2] (http://doi.acm.org/10.1145/263700.264352)
- Callahan, D.; Carle, A.; Hall, M.W.; Kennedy, K., "Constructing the procedure call multigraph," Software Engineering, IEEE Transactions on, vol.16, no.4pp.483–487, Apr 1990 [3] (http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=1950&arnumber=54302&count=13&index=12)