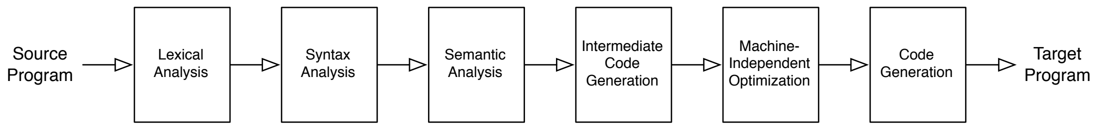
CS 301 Spring 2016
Meetings April 25

Runtime Systems

Plan



This week, we will read and discuss a variety of papers about runtime systems: code and systems that provide services to a program as it runs, but are neither part of the source code of that program nor the operating system of the machine. We will explore garbage collection, just-in-time compilation, adaptive optimization, the general idea of language virtual machines, and more. As you read this week, *focus on big ideas and perspective. Do not get too caught up in details.* I do not expect you to have a deep grasp of each paper. They are listed from more general to more specific (extras follow the exercises if you are really curious). Aim to get some background on GC and language VMs with the first overview papers in those sections. Pick at least one or two of the research papers to check out in some detail, but do try to skim at least the intro from all to get an idea of their goals.

Readings

URLs below should be clickable.

- Any optimization topics of interest from last week's reading.
- GC:
  - Paul Wilson, *Uniprocessor Garbage Collection Techniques.*
    `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.2438&rep=rep1&type=pdf`

    Focus on Sections 1–2, 3–3.2, 4. (You read this in 251 if you took it with me.)
- EC 8.7.1: Inlining
- JITs, Dynamic Optimization, VMs:
  - Matthew Arnold, *et al.*. *A Survey of Adaptive Optimization in Virtual Machines.* In Proceedings of the IEEE Vol. 93 Issue 2, February 2005.
    `http://www.ittc.ku.edu/~kulkarni/teaching/archieve/EECS800-Spring-2008/survey_adaptive_optimization.pdf`

    An overview of techniques for optimizing programs as they run. Read/skim for background and perspective, not specific system details.
  - Andreas Gal, *et al.*, *Trace-Based Just-in-Time Type Specialization for Dynamic Languages.* ACM Conference on Programming Language Design and Implementation (PLDI), 2009.
    `http://cdn.mozilla.net/pdfjs/tracemonkey.pdf`

    Focus on Sections 1–3; skim the rest. Much interesting detail about compiling *very* dynamic (dynamically typed) languages like JavaScript. EC 12.4 covers similar ideas for static compilation. Note: Firefox has since abandoned TraceMonkey, since other newer optimizations (*e.g.*, type inference) made it unneeded.[1] At the time, it was a huge and early step in taking JavaScript performance from terrible (simple interpreters) to surprisingly good (smart JITs) over the past 10 years.

[1] `https://blog.mozilla.org/nnethercote/2011/11/01/spidermonkey-is-on-a-diet/`

  - Thomas Würthinger, *et al.*, *One VM to rule them all.* In Onward!, 2013.

    Focus on Sections 1–3; skim the rest if interested. Could we build a VM for everything (dynamic) and make languages easily interoperable? Here's one attempt.
    `http://lafo.ssw.uni-linz.ac.at/papers/2013_Onward_OneVMToRuleThemAll.pdf`
    These slides might give some good background/context:
    `http://janvitek.org/events/PBD13/slides/MarioWolczko.pdf`
    `http://www.oracle.com/technetwork/oracle-labs/program-languages/overview/index.html`

- Dynamic Analysis:
  - Michael D. Bond, *et al.*, *Tracking Bad Apples: Reporting the Origin of Null and Undefined Value Errors.* International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2007.
    `http://web.cse.ohio-state.edu/~mikebond/origin-tracking-oopsla-2007.pdf`

    Focus on Sections 1–2 for the key idea; skim the rest, which is also interesting. This paper describes a kind of *dynamic analysis*, an analysis of the true run-time behavior of programs, to track where problematic nulls originate.

Exercises

1. Let's talk projects at the end of the meeting. These papers might give you some interesting ideas and I will pitch some others to you if you do not already have ideas.
2. Be prepared to discuss the general ideas behind the papers. The questions below will help target your reading.
3. Garbage collectors (familiar if you have taken 251 with me):
   - What is a limitation that applies to reference-counting, mark-sweep, *and* copying garbage collection?
   - What is a problem in both reference-counting and mark-sweep garbage collection that is addressed by copying collection?
   - What is one limitation of reference-counting that is not a problem for mark-sweep garbage collection?
   - What is the point of incremental garbage collection?
   - What is the key expected behavior of programs for which generational collection is optimized? (This is also called the *generational hypothesis.*) How does generational collection optimizes for this behavior?
4. JITs, Dynamic Compilation, VMs, Case Studies:
   - What are the upsides and downside of optimizing code at run time? Think about this at some length. Can run-time optimization clear some of the limitations of compile-time optimization? Is it it worth the cost? Consider the ideas of profile-guided optimization, feedback-guided optimization, adaptive optimization, etc. (All closely related.)
   - Identify several opportunites for run-time optimization for a language like Java. (First, think back to a paper we read before spring break on Polymorphic Inline Caches.)
   - How do optimization opportunities differ based on properties of the language we consider? Pick a few languages familiar to you, *e.g.*, C, Java, SML, Javascript, Python, Racket.
   - Why is tracing potentially useful for a dynamically typed language like JavaScript? Do you expect it to be useful in statically type languages? Why do you suppose it was eventually abandoned in Firefox?

- What is the goal of *One VM to rule them all* project? What is the key to their design/approach that is motivated by this goal?

5. Extended run-time checking:

   - What do you think of the Null Origin Tracking paper? (Bond, *et al.*)
   - What other properties could you imagine checking at run time?
   - How can a run-time analysis like this be more precise than a static (compile-time analysis)?

Still curious?

Here are some optional extras:

- JavaScript implementation:
  - Firefox's SpiderMonkey JavaScript Engine:
    `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals`
  - Chromium's V8 JavaScript Engine: `https://developers.google.com/v8/under_the_hood`
- Other:
  - Chris Lattner and Vikram Adve, *LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation.* CGO 2004.
    `http://llvm.org/pubs/2004-01-30-CGO-LLVM.pdf`

    LLVM has become widely used over the past several years as a general compiler middle/backend to generate native machine code. It had a C, etc., focus early on, but has generalized over time. It powers everything Apple (*e.g.*, Swift) these days.
  - Thomas Kotzmann, *et al.* *Design of the Java HotSpot^{TM} Client Compiler for Java 6.* ACM Transactions on Architecture and Compiler Opitmization (TACO), Vol. 5, No. 1, May 2008.
    `https://web.stanford.edu/class/cs343/resources/java-hotspot.pdf`

    Overview of the design of the standard JVM implementation as of a handful of years ago. Some things have changed since, but this is still fairly representative.
- GC:
  - Richard Jones, *et al.*, *The Garbage Collection Handbook: The Art of Automatic Memory Management.* CRC Press, 2012. (I have a copy; ask me.)
  - David Detlefs, *et al.*, *Garbage-First Garbage Collection.* International Symposium on Memory Management (ISMM), 2004.
    `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.6386&rep=rep1&type=pdf`

    Deployed in HotSpot JVM, along with Concurrent Mark-Sweep and others.
  - HotSpot VM GC Tuning Guide:
    `https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/index.html`
  - Stephen M. Blackburn and Kathryn S. McKinley, *Immix: A Mark-Region Garbage Collector with Space Efficiency, Fast Collection, and Mutator Performance.* ACM Conference on Programming Language Design and Implementation (PLDI), 2008.
    `http://users.cecs.anu.edu.au/~steveb/downloads/pdf/immix-pldi-2008.pdf`

    More recent algorithm, deployed in Jikes RVM.
  - Steve Fink and Robert Nyman, *Generational Garbage Collection in Firefox.*
    `https://hacks.mozilla.org/2014/09/generational-garbage-collection-in-firefox/`