More discussion questions for *A Runtime System* (Appel)

N. Ramsey, B. LaChance, and K. Cronburg, for COMP 250RTS

September 13, 2017

Reminder: we are postponing sections 5, 6, 7, and 22 until after we will have read Appel's paper on his garbage collector.

Discussion questions, round 2

Data layout: who controls?

(1) The compiler and run-time system have to agree on certain aspects of the representation of data. Consider these two models:

- The compiler controls the representation of data, and the run-time system has to adapt to it.
- The run-time system controls the representation of data, and the compiler has to adapt to it.

Answer these questions:

(a) Which model describes Standard ML of New Jersey (Appel's system)?

(b) Under what circumstances do you imagine that the other model is better? Do you know any systems that use the other model?

Shared secrets

The layout of data is just one example of secret information that is shared by the compiler and run-time system. And while a compiler and its run-time system always share secrets, the fewer secrets are shared, the more modular the system, and the easier it is to evolve. In this context, I think it will be useful to think about share secrets asymmetrically.

(2) In Appel's system, what run-time secrets does the compiler know?

(3) In Appel's system, what compiler secrets does the run-time system know?

(4) Compared with other systems you know, have heard about, or can imagine, how well does Appel's system hide information?

Foundation for the future: known and unknown calls

This question opens a problem we'll see throughout the term, and it also focuses attention on technicalities of sections 11 and 12.

(5) People who write compilers have learned to distinguish between "calls to known functions" and "calls to unknown functions." If you think in C terms, a call to a named function is probably a call to a known function, whereas a call through a function pointer is probably a call to an unknown function. Calls to known functions are ripe with opportunities for code improvement:

- They need not use the standard calling convention but can instead use lighter-weight conventions—or even a one-off convention.
- They can be inlined.
- When the forms of some arguments are known at compile time, those calls can be specialized.

With this idea in mind, let's revisit sections 11 and 12 of Appel's paper:

(a) Using the ideas of "known" and "unknown" functions, how do you explain what Appel is saying in section 11?

(b) Using the ideas of "known" and "unknown" functions, explain the consequences of the implementation of the module system described in section 12.

Bonus questions motivated by Monday's class

(6) Last time we talked about mapping Erlang, Java, or C# onto Appel's system. What implications do those mappings have for performance?

(7) Given that Java has single inheritance and a static type system, how would you design a layout for user-defined objects? The layout must support efficient method dispatch.

(8) Given your answer to the preceding question, what do you do about Java interfaces? More specifically, how do you compile code that exploits Java interfaces, and what support do you want from the run-time system?

(9) Suppose that

- Your language has a mutable string abstraction.
- Strings are rarely mutated.
- Strings are commonly broken into substrings, perhaps by parsing or regular-expression matching.

How do you want strings represented at run time? What run-time support do you want?

(We don't expect to be able to answer this question now, but we certainly ought to be able to answer it by the end of the term.)