

Re: Higher Order Instructions

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.clr/2007-04/msg00024.html>

- *From:* Barry Kelly <barry.j.kelly@xxxxxxxxxx>
 - *Date:* Sun, 08 Apr 2007 23:03:20 +0100
-

Christopher Diggins wrote:

I was wondering if there has been any research into adding higher-order instruction to the CIL? In other words instructions that either push or pop instructions on the evaluation stack.

There are only a few core instructions that would be necessary to build others :

- constantly : pop a value, push an instruction on the stack that returns that value
- compose : pop two instructions, push a new instruction that that evaluates the first, then the second.
- eval : pop an instruction and evaluate

CIL maps linearly to machine code. What you're describing here doesn't. However, if your instructions aren't first class (i.e. can't be passed or returned to / from methods, or stored / loaded from variables), then this scheme amounts to macro expansion since e.g. your compose operation can be statically expanded to its constituents.

And if your instructions are first class, verification would not be easy for e.g. constrained devices, and performance analysis would not be trivial. It could have similar problems by analogy to e.g. call by name from Algol, where evaluating an argument inside a function might be as simple as a variable read or as complex as a network call.

Also, as it exists, CIL can be trivially interpreted, in a pinch (type info added to stack values or to instructions after single-pass analysis). What your suggesting seems to me to be more like a kind of graph reduction machine, which would (naively, from 30 seconds analysis) suggest to me continuous dynamic allocation, quite unlike CIL.

This functionality would make it easier for me to compile functional languages to the CIL, and make them much more efficient.

Re: Higher Order Instructions

We have been discussing the topic on Lambda-the-Ultimate (<http://lambda-the-ultimate.org/node/2177>). The first response from many people is that they believe that this functionality has a huge performance hit, and loses the effect of statically verifiable type safety. This is untrue.

I've developed a type-system for stack-based languages with higher-order functions and written a paper about it at : <http://www.cat-language.com/paper.html>. I believe the work to be novel, and I would be interested in discussing it further.

I'll look into what you write when I have more time (it's late now :))

But it does look interesting, from a research perspective.

-- Barry

--

<http://barrkel.blogspot.com/>

.