[Instr Note] Real world CPS example

COMP 411 on Piazza <no-reply@piazza.com> Tue, Apr 7, 2020 at 1:46 AM
Reply-To: reply@piazza.com
To: cork@rice.edu

Instructor Simran Virk posted a new Note.

Real world CPS example

I wanted to post an old piazza note by Nick Vrvilo, a previous TA of 411 (who also wrote the reference interpreter). It provides some motivation for why the CPS transformation is useful -

Real-world Continuation-Passing Style (CPS) example

In past years, I know one thing students have struggled with is the motivation behind *why* you would want to do the CPS transformation. Basically any time that you can't depend on a "call stack" of some kind to handle your application's control flow, CPS (or some similarly-flavored transformation) is often the answer. Asynchronous computation and/or messages are a common example of code that breaks your ability to rely on the call stack.

C# 5.0 introduced the `async` and `await` constructs to help programmers write asynchronous control flows in a way that resembles synchronous code. They have a bunch of examples on that page of using these constructs to handle asynchronous HTTP communication. There's a similar feature being added to Scala in SIP-22, although it's available as a library right now. The C++20 standard is expected to add support for "coroutines" (formerly known as "resumable functions") to the language, which is another restricted version of the CPS transformation. The official editor of the coroutine proposal is a member of the Microsoft Visual C++ team, and the features are already available in Visual Studio.

The async/await constructs in both C# and Scala make use of the CPS transformation over a delimited section of the code (i.e., delimited continuations) in order to transform the seemingly sequential code into a series of asynchronous function calls. The Scala version actually uses a simplified CPS transform called A-Normal Form (ANF). I'm pretty sure that the CPS transformation we do in Jam is pretty much identical to the ANF transformation. This is not surprising since ANF was invented by a student of Matthias Felleisen—back when he was a professor at Rice—and our current Comp 411 material is rooted in the programming languages class that Dr. Felleisen created at Rice.

I think this quotation from the C# documentation nicely sums up the advantage of having this kind of CPS-transformation support build into the compiler/interpreter for your language:

> *The compiler does the difficult work that the developer used to do, and your application retains a logical structure that resembles synchronous code. As a result, you get all the advantages of asynchronous programming with a fraction of the effort.*

As an example of a language that didn't have a built-in concept of continuations, look at JavaScript and the callback hell that JavaScript developers have to work so hard to avoid. In contrast, TypeScript introduced `async/await` in version 2.1, saving their developers from callback hell by getting the TypeScript compiler to do the CPS transformation automatically on asynchronous control flows.

However, Async functions and futures were finally added to ECMAScript (the JavaScript standard) in 2017 in the 8th edition (i.e., ES8).

async/await in Python was added in 3.5, and they call CPS'd code "coroutine" code:
https://docs.python.org/3/library/asyncio-task.html#coroutines

Search or link to this question with @133.

Sign up for more classes at http://piazza.com/rice.

Thanks,
The Piazza Team
--
Contact us at team@piazza.com