Why all programmers want continuations

(but use callbacks instead)

Gabriel Kerneis

12 April 2015

Abstract: Have you ever wondered why callbacks are so pervasive in modern programming languages, and yet so hated that there is such an idiom as "callback hell"? Have you ever scratched your head so hard that you started losing your hair while debugging a maze of twisty little functions all alike? Have you ever wished you could write straightforward, linear, synchronous code, and let your programming language handle concurrency? This is 2015: why isn't your compiler able to link stack frames by itself as soon as you are writing asynchronous code? As it turns out, your compiler can in fact do this for you, and much more. It just needs a gentle push in the right direction.

This talk is a tutorial on escaping callback hell with promises and generators; examples are in Javascript, but should be accessible to any interested programmer. We first build a minimal promise implementation from first principles, discovering how the underlying hidden monad makes continuation-passing style programming easier and safer. Then, we go one step further, and throw generators into the mix to recover a direct, coroutine style, restoring sanity and reaching true enlightenment. We conclude with a brief tour of other popular programming languages, and discover that the essential building blocks are already available in most cases. Educating users about them is left as an exercise to the reader.

Acknowledgments: The author is grateful to Matt Greer and Jake Archibald for their tutorials on promises, heavily reused in this presentation. He also wishes to thank the many callback lovers (and the occasional continuation haters!) he has pitched this talk to in the last few months. Their insightful, if often fairly defensive, feedback has been the main motivation for giving this talk.

Disclaimer: The opinions expressed in this tutorial are those of the author, and do not necessarily reflect the official position of his employer. No callback has been harmed during the preparation of this tutorial.