Tail recursion in Clojure

Mark Armstrong

November 16, 2020

Contents

1 Introduction **1**

2 Motivation **1**

3 Recursion and the stack **2**

4 Tail recursion **2**

5 Visualising the stack **2**

6 A factorial function that shows its stack usage **2**

7 A tail-recursive factorial function that shows its stack usage **2**

8 A macro to automate showing stack usage **2**

1 Introduction

These notes were created during the tutorials on November 16th and 18th.

2 Motivation

In general, recursion as a control structure is *less efficient* than iterating (or looping) constructs, because each recursive call requires an additional *stack frame*, used to store the parameters and any local memory for the call.

This is why uncontrolled recursion usually results in a *stack overflow*, that is, running out of memory on the stack, whereas an infinite loop may continue to run indefinitely, never causing memory issues.

However, while these additional stack frames are necessary *in general* for recursion, there is a special case in which the stack frame can be *reused*, avoiding this issue. This kind of recursion is known as *tail recursion*, and our goal today is to visualise how it works, as well as come to understand how to write tail recursive functions.

We will use Clojure for this exercise, both because it is a functional language and hence recursion is a very natural control structure to use in it, and because its powerful *macro* system will allow us to provide you with a new way to define functions which automatically output a visualisation of their use of the stack.

3 Recursion and the stack

4 Tail recursion

5 Visualising the stack

6 A factorial function that shows its stack usage

7 A tail-recursive factorial function that shows its stack usage

8 A macro to automate showing stack usage