ResumableFunctions: C# sharp style generators for Julia.

Ben Lauwens

1 Royal Military Academy, Brussels, Belgium

Summary

C# has a convenient way to create iterators (Microsoft 2017) using the yield return statement. The package (Lauwens 2017a) provides the same functionality for the Julia language (Bezanson et al. 2017) by introducing the @resumable and the @yield macros. These macros can be used to replace the Task switching functions produce and consume which were deprecated in Julia v0.6. Channels are the preferred way for inter-task communication in julia v0.6+, but their performance is subpar for iterator applications.

The macro @resumable transform a function definition into a finite state-machine, i.e. a callable type holding the state and references to the internal variables of the function and a constructor for this new type respecting the method signature of the original function definition. When calling the new type a modified version of the body of the original function definition is executed: - a dispatch mechanism is inserted at the start to allow a non local jump to a label inside the body; - the @yield statement is replaced by a return statement and a label placeholder as endpoint of a non local jump; - for loops are transformed in while loops and - try-catch-finally-end expressions are converted in a sequence of try-catch-end expressions with at the end of the catch part a non local jump to a label that marks the beginning of the expressions in the finally part. The two last transformations are needed to overcome the limitations of the non local jump macros @goto and @label.

Straightforward two-way communication between the caller and the callable type is possible by calling the callable type with an extra argument. The value of this argument is passed to the left side of an arg = @yield ret expression.

The iterator interface is implemented so that a @resumable function can be used transparently.

Benchmarks show that this macro based implementation of semi-coroutines is an order of magnitude faster than both the original Task switching with produce and consume and the newer Channel based approach for inter-task communication. A context switch is more expensive than a function call.

The next generation of process-driven simulations in the discrete-event simulation framework (Lauwens 2017b) is based on this package.

References


Lauwens, Ben. 2017a. “ResumableFunctions: C# Sharp Style Generators A.k.a. Semi-

