Reconciling Synchronous and Asynchronous Execution Models

Francisco Sant'Anna

April 17, 2014

Abstract

Synchronous programming languages...
... .

1 Introduction

Concurrent languages can be classified in two major execution models. In the *asynchronous model*, the program activities (e.g. threads and actors) run independently of one another as result of non-deterministic preemptive scheduling. In order to coordinate at specific points, these activities require explicit use of synchronization primitives (e.g. mutual exclusion and message passing). *C* (the C11 standard [**?**]) and *erlang* [**?**] are examples of asynchronous languages (although they provide antagonistic concurrency primitives [**?**]). In the *synchronous model*, the program activities (e.g. callbacks and coroutines) require explicit control/scheduling primitives (e.g. returning or yielding). For this reason, they are inherently synchronized, as the programmer himself specifies how they execute and transfer control. Esterel [**?**], Céu [**?**], and Functional Reactive Programming languages [**?**, **?**, **?**] (in a much higher level) are examples of TODO.

The synchronous model has been successfully adopted in the field of reactive and control-dominated software for real-time embedded systems [**?**]. Synchronous programs execute under a discrete logical time unit (a *tick*) which continuously compute atomic reactions to the environment (*inputs*). Each tick must execute in negligible time so that... This is the synchronous hypothesis, which states that if TODO [**?**]. is considered to take no time TODO: zero-delay / synchronous hypothesis. This disciplined model simplifies the reasoning about concurrency aspects and enables compile-time verifications that ensure deterministic execution for programs [**?**].

However, the strict temporal order and atomicity between subsequent ticks restricts the expressiveness of synchronous languages:

1. Testing and simulating programs require external tools. Fundamentally, a self generated tick would create a temporal paradox in which the current and future ticks execute concurrently.
2. High priority tasks with hard real-time requirements cannot preempt other tasks. Even if a critical hardware interrupt jumps the external queue, the current reaction still needs to complete to preserve atomicity.
3. Reactions cannot performing heavy calculations (e.g. compression and cryptography), otherwise subsequent reactions would not be handled in real time.
4. Real parallelism in multi-core CPUs is challenging: two ticks cannot be handled at the same time; while intra tick parallelism breaks determinism.

all these limitations are inherent to the synchronous model we believe that, instead of modifying it, we should extend it with asynchronous

We argue that each of these functionalities represents a type of asynchrony in programs and should addressed by different primitives. By doing so, each primitive can have a more clear semantics and restricted side effects, leading to a better integration with the language that minimizes the safety threats of asynchronous execution. We propose three asynchronous extensions to Céu [**?**], a Esterel-based synchronous language:

- Asynchronous Blocks (`async`) to deal with input generation preserve determinism.
- Interrupt Service Routines (`isr`) to deal with memory sharing is restricted and enforced by the compiler
- Thread Blocks (`thread`) to deal with again memory sharing is restricted and enforced by the compiler.

Céu is safety-oriented system language for constrained embedded systems. TODO

- synchronous kernel with asynchronous primitives - instead of the opposite - we think synchronous by default is better - asynchronous is exception

Ortoghonal to the synchronous model each can be disabled without impacting the overall utility of the language.

composable with the synchronous side except for `isr`

Contributions:

- Classification of typical asynchronous functionality required in programs.
- Considerations about safety aspects in asynchronous execution.
- Implementation of these ideas in an existing synchronous language.