Proposal Addendum

Justin R. Wilson

1 Implementation: Translation vs. Library

The goal of the implementation is to allow developers to encode reactive systems using reactive components. The first approach is to specify reactive components directly by designing a new programming language. This approach necessitates the development of a translator that takes reactive components and produces a form that can ultimately be executed. A viable approach to the translator may be to output another programming language such as C or Java. The second approach is to specify reactive components indirectly in an existing programming language via a library.

Both approaches rest on a set of semantic checks that enforce the atomicity and determinism of transitions and composed transitions. The atomicity and determinism guarantees are needed to avoid subtle data races and therefore are a necessary part of any implementation. These checks are performed at compile-time in the translation approach and either at load-time or run-time in the library approach.

In addition to the semantic checks and run-time library of the library approach, the translation approach requires the development of a translator. In the library approach, the burden of translation is shifted to the developer as reactive components and transitions must be defined programmatically.

The main issue with the library approach is the difference in semantics between the host language and the semantics of reactive components. Lee describes how introducing concurrency via library significantly alters the semantics of the language [1]. Most likely, the host language will be sequential imperative programming language like C or Java. Reactive components on the other hand are based on non-deterministic sequencing. If the target of the translator is a language like C or Java, then both approaches must ultimately express reactive components using the semantics of a sequential imperative language. The advantage of the translator approach is that the introduction of reactive component semantics is automatic and not subject to programmer error.

References

[1] E.A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.