MLRTEMS: Embedded Standard-ML

Nathan Burgers

May 3, 2013

1 Summary

Functional programming is often ignored in embedded and real-time systems because of its perceived runtime overhead. This project aims to provide a bridge between the MLton optimizing Standard-ML compiler, and the RTEMS real-time systems executive.

2 Motivation

C is currently the language of choice for embedded systems, and is the target language for RTEMS. Historically, C has proven itself to be insecure due to problems with dangling pointers, lack of true type safety, unbounded arrays, and ease of accidentally leaking memory. Embedded systems frequently demand high code stability and may be long-running processes where memory leaks are a serious issue.

In contrast, Standard-ML provides true type safety, immutable data types, mutable references, bounded arrays, and garbage collection, among other features. I argue that embedded systems stand to benefit from functional programming concepts. Standard-ML is the language of choice because it has extensive specification and several mature implementations that emit C as their target language.

3 Project Overview

MLton translates Standard-ML code to optimized C and RTEMS is provided as a series of C libraries that are conditionally compiled to meet the

needs of the system. A basic RTEMS embedded system using Standard-ML requires bridging the two technologies using MLton's foreign function interface, and an associated toolchain to build MLton, RTEMS, and the embedded software product.

The primary focus of this project will be on tooling and will proceed in three phases:

1. Provide a build automation suite to make the process of contributing to the MLton and RTEMS projects simpler. RTEMS is notorious for being difficult to compile, and the barrier of entry for contributing to either of these projects is currently high.

2. Develop a set of benchmarks suitable for a real-time system alongside MLton's benchmark suite and run them against real hardware, locating problem areas. This project currently has access to phsyical leon3 development boards.

3. Target the areas of the MLton compiler that pose problems for real-time applications. It is expected that garbage collection will be the primary factor here. Methods of improving garbage collection speed can range from providing different functional data structure primitives, to altering the garbage collection scheme to reference counting or memory regions, etc.

4 Success Criteria

1. An automated build suite can update and compile the MLton and RTEMS sources for producing binaries on RTEMS supported hardware.

2. Get a suitable benchmark suite running, and the benchmarks online.

3. Run Standard-ML code compiled with MLton and RTEMS on a physical device.

5 Far Reaching Goals

1. Investigate options for and begin forking the MLton memory management scheme to provide real-time guarantees.

6 Roadmap

I would like to work out the roadmap for this project closely with a mentor given discussion about where to take the project once MLton can build its own benchmarks on RTEMS with real hardware.

First 2 weeks Meet with mentors and complete work on build automation suite.

Weeks 3 to 7 Compile MLton benchmark suite and get it running under RTEMS, addressing any issues preventing interfacing with RTEMS or cross-compilation.

Weeks 8 to 13 Address issues affecting real-time capability, like memory management. Possibly exploring memory regions or reference counting, etc.

7 Qualifications

My resume can be downloaded in pdf form from `resume.nateb.me`

- Over six years of general programming experience.
- Over one year of experience developing tools at Refulgent Software LLC.
- Currently working on an automated build suite for RTEMS by scraping its latest versions, dependencies, and tool patches from the RTEMS Git repository. Written in Clojure.
- A vigorous interest in programming langauges and robotics.