Linear Temporal programming in an Esoteric Programming Language

Contact: [email]@grinnell.edu

Tristan Knoth
Grinnell College
[knothtri17]

Reilly Grant
Grinnell College
[grantrei]

Chris Kottke
Grinnell College
[kottkech17]

Abstract

This project attempts to create an esoteric programming language deigned based on the popular cartoon, Rick and Morty. This programing language will make use of integrated Linear Temporal logic to ensure correctness, and advanced multithreading. In addition, this language is intended to be interseting enough to attract students and programmers unfamiliar with these concepts under the veneer of an interesting language and thus introduce possibly non acadmemically oriented students to the concepts of temporal logic, and multithreading.

1. Introduction

The popular TV show Rick and Morty often uses the concepts of Parallel timlines and alternative universes as a main theme in may episodes. Given the popularity of the show and its relevence in pop culture, it was decided that it would be a interesting medium over which to introduce the concepts of multithreading, and temporal logic. This language will include many refrences to the show in the syntax, and thus be attractive to fan of the show, and also make the use of temporal logic more whimsical.

Our final goal is to create a turing complete language which satisfies the above conditions of refrence, and also makes serious of the concepts of linear temporal programming.

2. Prior Work

Linear Temporal Logic:

Principles of languages design: Language design is a field which has had a lot of focus directed towards it. Language features have been divided up into features that deal with Reading, writing, running and reasoning, and have

There are also currently hundreds of esoteric languages that have been created that

3. Proposed Work

Temporal logic can be an extremely useful concept in various computing problems, particularly related to program verification. In order to ensure, for example, that a certain state is eventually reached, we need temporal logic. Similarly, we can use temporal logic to ensure that certain eventualities are never reached. A final important application is to ensure ”fairness” in multi-threaded systems. A program may, for example, want to ensure that if a process makes a certain request often enough, that request is eventually fulfilled.

Thus, we propose to use Haskell to develop a simple Turing-complete computer language featuring temporal logic and some simple multi-threading utilities. Without multi-threading, we cannot use the temporal logic capabilities to their full potential. GHC already supports implicit parallelism. Thus, we can simply use GHC's implementation as the foundation of our multi-threading capabilities.

Our language will also support various temporal logic statements. This will allow programmers to properly verify non-terminating applications.

4. Timeline

1. Friday 11/4: project checkpoint 1 due:

   By Friday 11/4, we intend to have a basic Turing complete language with many of the jokes and refrences of Rick and Morty Implemented.

2. Friday 11/18: project checkpoint 2 due:

   By Friday 11/18 we intend to have implemented singly threaded temporally logical system integrated into the language. This will be reminicent of Haskells lazy execution due to it's singly threaded nature.

3. Friday 12/2: project checkpoint 3 due:

   By Friday 12/2 we intend to have easily implemented a system which allows the user to simply handle multithreading, and serial execution.

4. Monday 12/5 and Wednesday 12/7: project presentations:

   By this point, we intend to have finished the project, and mostly be cleaning and them presenting to our peers.

5. Friday 12/9: final project deliverables due:

   By this point, we will have the basic all implemented, or at least some versions of all of them. We may have remaining features that we were unable to implement due to the scope of the project, but complete basics implementation will be done.

References