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

For this project we will develop an esoteric programming language based on the popular cartoon, Rick and Morty. This programing language, which will support multithreading, will also make use of integrated Linear Temporal logic to ensure correctness. In addition, this language is intended to be accessible enough to attract students and programmers unfamiliar with the concepts of temporal logic and parallelism.

1. Introduction

The popular TV show Rick and Morty often uses the concepts of parallel timelines and alternate universes as a main theme in many episodes. Given the popularity of the show and its relevance in pop culture, we decided that it would be an appropriate medium through which to introduce the concepts of multithreading and temporal logic. The language's syntax will include many references to the show, and thus be attractive to fans of the show. It will also make the use of temporal logic more whimsical and thus increase the topic's appeal to a diverse audience.

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

2. Prior Work

Linear Temoral Logic is a mathematical frameworks that deals with when statements are true over time. It has been used in programming for software verification, specificly in when working with concurrent programs[2, 7]. Temporal Logic has also been used in declarative programming languages to increase the expressivness[6]. As far as we know, we could not find a non declartive programming language which made use of Temporal programming as part of the language.

Additional work that will be imporant to this project is language design. Language design is a field which has had a lot of focus directed towards it. There are currently hundreds of languages that have been created purely for exploration of creating interesting programming lanaguages, and the field of programming research is one which has been growing [1]. It is easy to find advice on creating programming languages, such as that given by Dominic Orchard of the university of Cambridge [9]. With large amount of work that has been done on programming languages, we predict that finding resources to assist us will done be extrodinarily difficult.

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 applicatio n 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. The primary goal of our use of multithreading is not user control; rather it will give a context in which temporal logic can be more usefully applied. While we are unsure of exactly how we will implement this system, threadpools are a particularly attractive option as they will allow easy "blocking" and management of threaded resources.

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 references 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 reminiscent of Haskell's 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, as well as 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 it 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

[1] Language list. URL `https://esolangs.org/wiki/language_list`.

[2] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, Apr. 1986. ISSN 0164-0925. doi: 10.1145/5397.5399. URL `http://doi.acm.org/10.1145/5397.5399`.

[3] M. Coblenz, J. Sunshine, J. Aldrich, B. Myers, S. Weber, and F. Shull. Exploring language support for immutability. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 736–747, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3900-1. doi: 10.1145/2884781.2884798. URL `http://doi.acm.org/10.1145/2884781.2884798`.

[4] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):6195, 1991. doi: 10.1016/0004-3702(91)90006-6.

[5] J. Duregård and P. Jansson. Embedded parser generators. *SIGPLAN Not.*, 46(12):107–117, Sept. 2011. ISSN 0362-1340. doi: 10.1145/2096148.2034689. URL `http://doi.acm.org/10.1145/2096148.2034689`.

[6] J. Gaintzarain and P. Lucio. Logical foundations for more expressive declarative temporal logic programming languages. *ACM Transactions on Computational Logic*, 14(4):141, Jan 2013. doi: 10.1145/2528931.

[7] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. *Protocol Specification, Testing and Verification XV IFIP Advances in Information and Communication Technology*, page 318, 1996. doi: 10.1007/978-0-387-34892-6_1.

[8] P. Jonsson and C. Bckstrm. A unifying approach to temporal constraint reasoning. *Artificial Intelligence*, 102(1):143155, 1998. doi: 10.1016/s0004-3702(98)00031-9.

[9] D. Orchard. The four rs of programming language design. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2011, pages 157–162, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0941-7. doi: 10.1145/2089131.2089138. URL `http://doi.acm.org/10.1145/2089131.2089138`.

[10] D. Orchard. The four rs of programming language design. *Proceedings of the 10th SIGPLAN symposium on New ideas,new paradigms, and reflections on programming and software - ONWARD '11*, 2011. doi: 10.1145/2089131.2089138.