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 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[1, 6]. Temporal Logic has also been used

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. Dominic Orchard of the university of Cambridge recomends thinking of programming languages in terms of how well the implement 4 facets, Reading, Writing, Running, and Reasoning [8]. Writing deals with how the language handles abstraction, how easy it is to do low level work, and conversely how easy high level

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

3. Proposed Work

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

[1] 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`.

[2] 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`.

[3] 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.

[4] 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`.

[5] 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.

[6] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. *Protocol Specification, Test-*

ing and Verification XV IFIP Advances in Information and Communication Technology, page 318, 1996. doi: 10.1007/978-0-387-34892-6_1.

[7] 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.

[8] 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`.

[9] 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.