Turing Project Report

Group Turing
Jonatan H. Hindersland, Didrik K. Drøsdal, Andreas Grimsmo
https://github.com/uiano/Turing-MPS

IKT445
December 2022

Abstract

The Turing language is a programming language specification made in Jetbrain MPS, Meta Programming System. The purpose of this language is to express the concepts behind Turing machines. This is done through two primary methods, the table machine and the combination machine. The table machine is the more standard version of a Turing machine and is designed to be able to manipulate a machine tape. The combination machine in contrast has the primary purpose of running other Turing machines, both table machines and other combination machines. Compared to the table machine the combination machine is not able to do base level tape manipulation. This report will discuss the state of the language from the work of previous student groups and the changes made by our group. At the start of this project most functionality existed in the combination machine whereas the table machine was mostly unfinished. The project therefore focused mostly on finishing the table machine. At the end of the project we can now say that the table machine is in a fully functional state to the same degree the combination machine was.

Contents

1 Introduction **3**
- 1.1 Background . . . 3
  - 1.1.1 Planned Improvements . . . 3
- 1.2 Previous Work . . . 3
  - 1.2.1 Version 2017 . . . 3
  - 1.2.2 Version 2018 . . . 3
  - 1.2.3 Version 2019 . . . 4
  - 1.2.4 Version 2021 . . . 4
- 1.3 Report Structure . . . 4

2 Language **5**
- 2.1 Turing Machine . . . 5
- 2.2 Combination Machine . . . 6

3 Solution **7**
- 3.1 Structure . . . 7
- 3.2 Syntax . . . 7
- 3.3 Semantics . . . 9
- 3.4 Run Configurations . . . 9
- 3.5 Examples . . . 9
- 3.6 Documentation . . . 9

4 Discussion **10**
- 4.1 Run button . . . 10
- 4.2 Runtime . . . 10
- 4.3 Constraints . . . 10
- 4.4 Combination editor . . . 10
- 4.5 Final state . . . 10
- 4.6 Redundancy . . . 10

5 Plan and reflection **11**
- 5.1 GIT Management . . . 11
- 5.2 Management . . . 11

1 Introduction

The concepts and the relations to the concepts in the Turing Machine can be represented through the Turing language. The Turing programming language was build using the language workbench: Meta Programming System, also called MPS, which is distributed by Jetbrains. The Turing language is a domain specific language designed to be a simplistic language for developers working on a Turing machine. This domain specific language allows developers to create Turing machines without requiring the same level of programming expertise as you would need to develop in general purpose languages.

1.1 Background

This project started in 2017 by a group of students at UiA, but was not finished this year. The following years this project was passed along to new groups, and was improved every year except in 2020, due to the project not being worked on. Our group intend to continue the development of this project, the Turing language, where the last group in 2021 left off. When we got the project the combination machine and runtime was working, but the table machine was left unfinished. We intend to fix up on minor problems, but our primary focus is gonna be to make the table machine working and give it a tabular editor.

1.1.1 Planned Improvements

1. **Table Machine** - Give the table machine a tabular editor view, and make it work as wanted.
2. **Fix other to-do list points** - Fix up in minor problems listed in the to-do list.
3. **Documentation** - Add additional information to the documentation to make the development easier for future group who are gonna pick up this project after us.

1.2 Previous Work

1.2.1 Version 2017

This project started up in 2017 by a couple of students at UiA. They built the Turing language in a Jetbrains software called Meta Programming System, MPS for short. These students made a text-based tabular machine with all the necessary concepts and relations to make a working Turing machine. However they did not manage to implement a runtime environment, which makes the machine unable to run already built machines.

1.2.2 Version 2018

The group working on this project in 2018 started off by scrapping what the previous group did, and started over again. They made a table machine and a combination machine, as well as a runtime solution. This was entered by pressing alt+enter, and when the runtime display was up the user would then input the symbols into the tape. After the machine had done it's operations, the resulting output tape would be shown in a pop up. Finally they made a textual node based editor for the combination machine.

1.2.3 Version 2019

The 2019 group started by implementing a tape input field and a run button in the editor. This provided the user with a new way to run a machine, and by typing in the tape content in the top of the editor one can set the input. This group also changed the presentation of the combination machine from textual node based into only textual based. They also removed the 2018 runtime solution and replaced it with a class under behavior. This group started to implement changes to the table machine, but were unable to finish it.

1.2.4 Version 2021

This group started with adding a new run button that allows the users to run the Turing machines without entering each machines' editor. They also improved the combination machines editor for easier reading. They tried to improve and fix on the table machine, but were unable to finish this part similar to the 2019 group. This group also added some comments and improved the user interface in the editor for less experienced programmers.

1.3 Report Structure

The different sections of this report will contain the following:

1. **Language** - In this section we will describe the key components of the Turing Machine and how the machine in general functions. This description will then be used to explain the Turing language.

2. **Solution** - This section describes the different aspects of the Turing language, in the order of; Structure, Syntax and Semantics. This also includes constraints.

3. **Discussion** - Here we will reflect on our work on the project. The section will also go over some pros and cons on the current solution, as well as go over any current problems. We will also include some ideas and plans for the project going forward.

4. **Plan and Reflection** - This section will go over our plan for the project, and how we utilized our time according to the plan. We will also reflect on our project management and on our execution of our plan.

2 Language

2.1 Turing Machine

The Turing machine is a hypothetical computing machine conceived by Alan Turing in 1936. The way the machine works is that it reads an infinitely long tape which is divided into cells[2]. Each cell can contain symbols from a predetermined alphabet, in our case they are zero(0), one(1), and a blank character(#) which represents the absence of 1 and 0.

The part of the machine that reads the current value from the tape is known as the head, and it is always placed directly over one cell at the time. It is also responsible for other operations besides read, and those are to write to the tape and to move to the left or to the right.

Which action the head performs is dependent on the current state, it's set of rules, and the current value the machine has read.

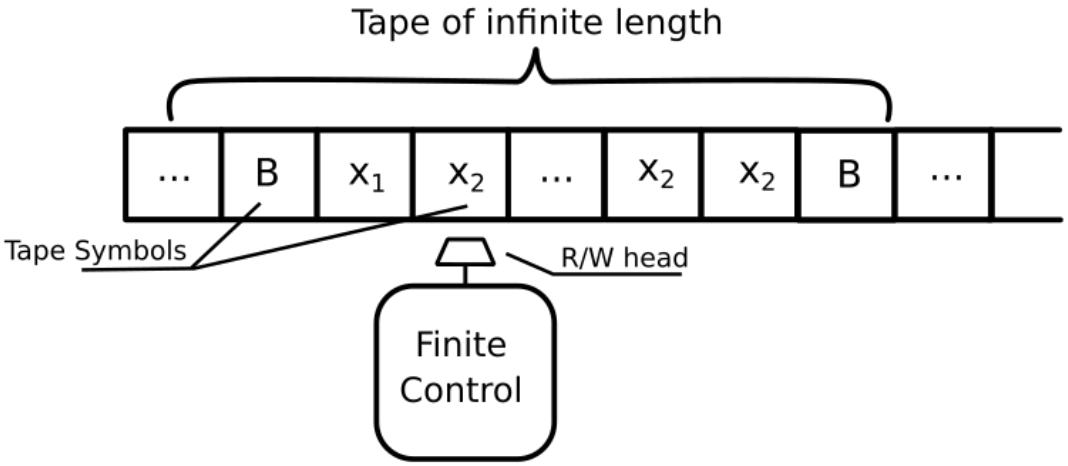

Figure 1: An example of the head placed over the tape of the Turing machine[1]

When the machine is initialized it will enter the initial state, which is the first of a number of states. The last state is called the final state. When all of the operations are completed the machine will enter the final state and the results of the operations are printed to the tape.

Between the initial and final state there can be an infinite amount of other states which contains rules as to which operations are to be performed, as well as instructions on when to move onto the next state.

Each operation performed contains several transitions, always performed in the same order: read value, write to tape, move either left or right(or stay), and then move to a

new state or stay at the current one.

Despite it's simplicity the Turing machine is able to implement any computer algorithm.

2.2 Combination Machine

The Combination Machine is a Turing Machine that can run other Turing machines. The way it works is that once one machine is finished, another one is called and picks up where the first left off. It keeps doing it until the entire stack/list of machines has been run.

3 Solution

When we started this project the combination machine was in a functional state whereas the table machine was not. The main focus of this project has therefore been to bring the table machine up to a functional state, as well as finalizing the connection between the two machines. This included removing functionality, like write and move, which were only supposed to exist in the table machine.

3.1 Structure

The structure of the table machine at the beginning of the project consisted of a TableMachine concept, a TableState concept and a TableOperation concept. A TableMachine has one or more TableStates as children. A TableState has three TableOperations as children, one for each possible input; blank, zero and one. Originally the TableOperations had the properties write and move, as well as the reference nextState. These parts were all moved to separate concepts so they could be assigned as children to the TableOperation. These new concepts were called TableWrite, TableMove and TableGoto. This was to make the structure more compatible with the table editor format. Additionally we also created a new HeaderString concept which is simply used to create static labels for the table itself.

For the combination machine there were not many major changes. The structure of the combination machine is a little different from the table machine. The difference is that the the CombinationOperation does not contain any children. Instead there are three concepts that extend the CombinationOperation, in other words they are types of CombinationOperation. This means that these concepts can be inserted directly as children of the CombinationState. These concepts are Conditional, which functions as a read and contains three lists of operations based on the input symbol, RunMachine which references another Turing machine, and Goto which references another state in the combination machine, similarly to the table machine. The changes made to the combination machine structure in this project was to remove write and move as possible CombinationOperations. This was because the combination machine should not have access to such low tier operations, it should have to run a table machine in order to manipulate the machine tape in any way.

When it comes to language constraints no new ones were added to the combination machine, but the existing constraints were adapted for the table machine. This includes making it so the string input in the initial property can only contain legal values, restricting state names so they have to be unique, and restricting the nextState value so it can only point to states of the machine it is currently in.

3.2 Syntax

The editorial aspects of the table machine were pretty much non-existent, only utilizing the default editor. This meant the table structure had to be implemented from scratch. The way this was done was by taking inspiration from the MultipleProjections project that exists in MPS. The editor aspect for the TableMachine creates an instance of a java

class. This class contains functions that fetch the number of rows and columns which returns five for the number of columns, and the number of rows is the number of states times three plus one. This is because each state has three operations and the table also has a header row. The class also returns the value of each table element. For the top row the class returns a HeaderString with an appropriate name. This is purely for the sake of the user. The first column returns a TableState, and lets the user assign a name. The second column also returns a HeaderString, which simply gives the values blank, 0 and 1, to account for the three possible read values. The third column returns the TableWrite concept, the forth column returns the TableMove and the final column returns the TableGoto concept. The class also allows for creating and deleting rows, using enter and backspace respectively. It also allows for creating and deleting table elements, specifically transitions. This is also done with enter and backspace.

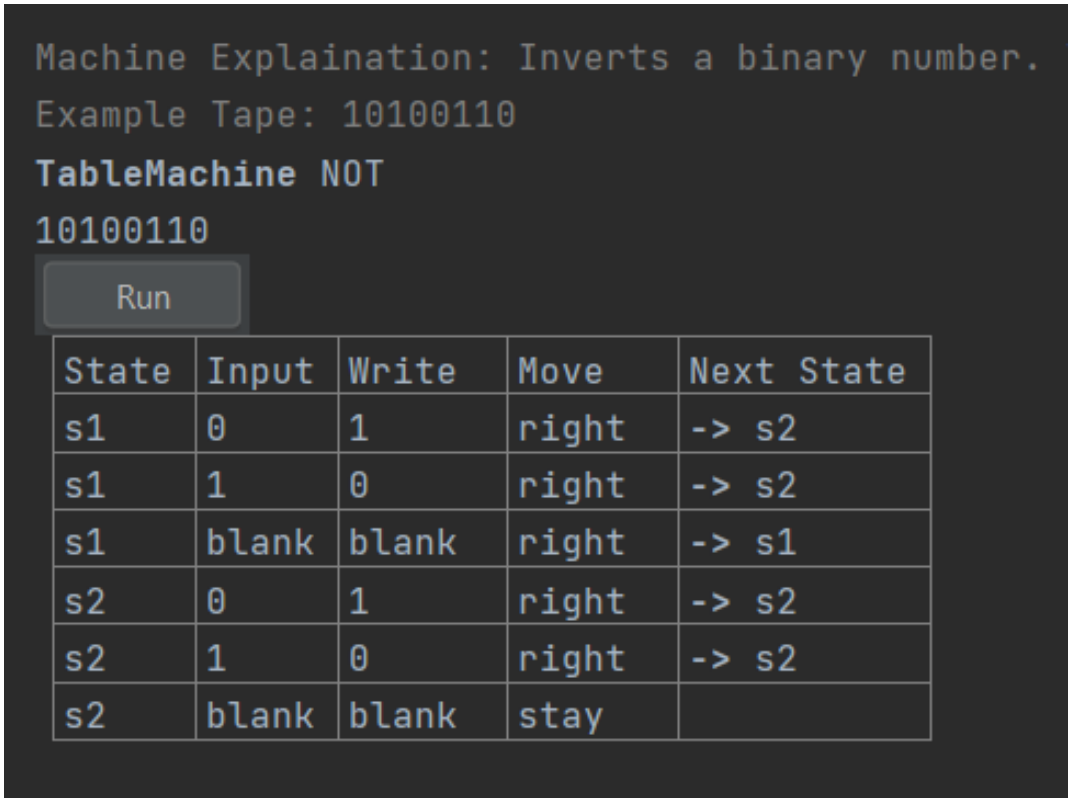

Figure 2: An example of a table machine

The editorial aspects of the combination machine are comparatively simple. The machine contains a list of states and each state contains a list of operations. RunMachine and Goto are both represented as a single line, however Conditional is represented as three labels who each have contain a list of operations. This aspect was not changed in this project

3.3 Semantics

When implementing the behavioral aspects of the table machine we took inspiration from the existing behavior of the combination machine, however it did require some rearrangement because of the variations in structure. Luckily most of the actual tape manipulation is handled by a java class called MachineState, which we did not have to change. The MachineState is what handles all actual tape manipulation, including read, write and move. When a machine is started the class is initialized with the input tape string. The tape is saved in the MachineState as an arraylist. This MachineState is then sent to the run function which starts the first state of the machine. The TableState then checks the current value of the tape to determine which TableOperation to run. Then finally the TableOperation runs TableWrite, TableMove and TableGoto. TableWrite checks its own value and writes said value to the machine tape. TableMove determines if the machine head should move to either side or stay where it is. Finally TableGoto checks its nextState value which references another TableState. This TableState is then run in the same way the TableMachine does. This process is repeated until the machine reaches a state that does not have a next state.

The combination machine is pretty similar to the table machine in how it is run, as it is what we based ourselves on when making the table machine behavior, but it does contain some differences. The CombinationState does not perform any read functionality, instead it simply runs all operations. It is instead the behavior for the Conditional concept that runs the read function and runs all operations in one of the three lists based on the read value. The Goto is identical to its equivalent in the table machine and simply runs the next CombinationState. RunMachine also functions in a similar way, only that it runs a machine instead of a state.

3.4 Run Configurations

The Run Machine plugin that is used to run a machine from the file overview originally only allowed for running combination machines. We changed this to check if the selected concept is a table machine or a combination machine, and then run the appropriate one.

3.5 Examples

The machine examples have been updated to account for the changes made to the language. There are also a few new examples to showcase the table machines. A lot of the basic machines that previously existed as combination machines have been converted to table machines and moved to an accessory model for the language itself. The icons for the two machines have also been changed to make it easier to distinguish them from each other.

3.6 Documentation

The documentation has been updated to account for the changes made, especially the table machine. We also tried to make the explanation of both the language itself and the user manual more clear and detailed, since we found them to be a bit too surface level.

4 Discussion

We have made a good number of changes to the Turing language and we are overall happy with the improvements we have made. The language is now in such a state that both the table machine and combination machine are fully operational. There are however still some changes that could be implemented for a better language.

4.1 Run button

The run button and initial tape value should not exist in the machine itself as the machine should only contain the machine specification itself. These should therefore be moved to a separate concept.

4.2 Runtime

The runtime solution can be improved in a number of ways. One change could be to change the runtime elements so that the next state/machine is returned to the surface element rather than added to a stack. This stack approach could cause poor performance with very large machines. Another element that can be improved in runtime is to add some method of debugging, for instance breakpoints or an operation log.

4.3 Constraints

There are a number of ways the machines could be constrained to be both easier to use and more clean. For instance it should not be possible in the combination machine to use a read operation inside another read operation. There should also be constraints in place to try to avoid infinite loops.

4.4 Combination editor

The editor for the combination machine is somewhat lackluster and could use improvement.

4.5 Final state

Currently the machine ends if it reaches an operation with no next state. This could be changed to refer to a final state instead. If this idea is implemented it would also not be necessary to allow empty values in the next state column of the table, and the create and delete element functions can therefore be removed.

4.6 Redundancy

There are currently some concepts, such as operation and state, that are labeled as common. These are somewhat redundant and could be removed. There might also be some concepts that could be renamed for the sake of clarity.

5 Plan and reflection

5.1 GIT Management

GIT is a wonderful tool for both version control and for providing a good backup solution. Unfortunately we were not as good at utilizing GIT as we should have been during the project. This caused us some problems. Due to the nature of MPS and the directory including multiple versions of the language we suddenly realized that trying to access a previous version of the language caused all our changes to be overwritten. This could have been avoided with better use of GIT. Luckily a lot of the time spent on the project at that time had gone towards understanding the existing structure, so redoing the work took significantly less time. We also got much better at using GIT in the later parts of the project. On a similar note we have also removed the previous versions of the language from the GIT repository so this error is now hopefully less likely to occur for future groups. The language has gone through many rewrites by various student groups, and there might still be some aspects of the previous language versions worth adapting, therefore those versions are still available from GIT commit 7057918.

5.2 Management

Since the three of us live together it was very easy to have meetings to discuss the project and how to solve the various problems we encountered. The reflection tasks throughout the semester helped give us a better understanding of the structure of the language. We also believe the biweekly project meetings at the university helped us better understand how to solve the problems in the language. However, it might have been better for those meetings to have a heavier focus on the projects themselves rather than a large part being dedicated to the general syllabus. Those sessions could have been more of a shared workshop, being able to discuss project issues between groups.

References

[1] Gabriel Lechenco. *A general introduction to turing machine*. Sept. 2019. URL: https://iq.opengenus.org/general-introduction-to-turing-machine/.

[2] Alan Turing. 'Computing Machinery and Intelligence (1950)'. In: *The Essential Turing*. Oxford University Press, Sept. 2004.