ToC: Final Project [250 points]

Several deadlines: November 20th, December 4th, December 15th

How to submit

For all parts of the project that you submit: please put your solution in a folder whose name is your Hawkid. If you worked with a partner please include a file called `partner.txt` that lists the Hawkids for you and your partner. Only one partner should submit. You will both receive the same grade. Once you begin working on the project with a partner, you should continue with them through the end of the project. If you run into difficulties, email Prof. Stump.

How to get help

Please post questions to ICON Discussions, if possible (as opposed to emailing us directly), so that our answers can benefit others who likely have similar questions. You may also come to our Zoom office hours, listed on ICON under Pages - Office Hours.

Overview

The project is to implement a universal Turing machine in Haskell in `UniversalTM.hs`, starting from the encoding of Turing machines that is already there. I presented this encoding in class a couple weeks ago. **You are free to change the encoding if you find that would help you.** The function `inputU` in `UniversalTM.hs` turns a TM and an input string for that TM into a single input to the universal TM. Your universal TM should have the same behavior on such strings as the encoded machine has on the encoded input (accepting, rejecting, or looping). To keep this problem from getting too difficult, you may assume that the input TM is deterministic. But you may wish to utilize nondeterminism in writing your universal TM, as this is supported by `TM.hs`. (We will not attempt to run the universal TM inside the universal TM!)

Because the universal TM will be more complicated than others we have tried to write in class, it will be necessary to have some debugging tools for execution of TMs. Therefore, one of your tasks is to come up with some debugging function that you think will help you keep track of what is happening as a TM executes, and implement it. You will put the code for this in `TM.hs`. Example things you could implement are:

- a function which is like `accepts` (in `TM.hs`) except that it takes in an input and a number `n`, and returns the list of configurations reached after `n` calls to `newConfigs`.
- a function which is like `accepts` but returns the final configuration, rather than just accepting.
- a function which lets you start the TM from a configuration that you specify by a current `state` and finite list of `tape` symbols (which your code should then pad out with an infinite list of blanks, as done already in `initialConfig`.

- anything else you can think of.

I have not tried to solve this problem myself, and am not sure how hard it will turn out to be. I expect the hardest part will be debugging (hence the debugging tools). Just do the best you can, and document in your final report what you were able to get working. Submissions that do not completely work but demonstrate good effort will still get high marks.

Deadlines

The work of the project is divided up, with the following deadlines. The different tasks are explained in the following sections.

- **November 20th (Saturday):** project proposal
- **December 4th (Saturday):** status report
- **December 7th/9th (in class):** presentation in class on the work done so far (does not need to be complete)
- **December 15th (Wednesday of finals week):** final submission

Your submission should be cumulative: please include all the earlier files you submitted, as part of each later submission. I will just refer to your "submission folder" below, where you will accumulate all the things you are supposed to submit. You will submit the whole folder for each deadline. Please keep your source code in there, too.

1 Project proposal [40 points]

Deadline: November 20th. Add to your submission folder a plain text file with the following information (and then submit your folder on ICON):

- Name of your planned universal TM (catchy names wanted!)
- Who is working on the project (just you or you and your partner)
- What debugging tool will you implement for TMs? Bear in mind that a tool which might be harder to implement could make your life much more pleasant when you actually have to debug your universal TM.
- Rough idea of the algorithm you will try to implement for your universal TM. You should sketch how you will organize the tape to keep track of the current state of the encoded machine, and current location of its read head. Also, do you expect you will change the encoding (this is allowed)?
- Preferred class date for presenting your project over Zoom (either Tuesday, Dec. 7th, or Thurday, Dec. 9th). You will just be presenting what you have done to that point; you are not expected to be finished with the project by then.

2 Status report [25 points]

Deadline: December 4th. Add to your submission folder a text file called `status.txt` briefly stating what you have done so far. You will not receive points for this part unless you have actually written some code for the project, and describe what you have done. A paragraph is enough. You should include your updated `.hs` files in your submission folder so we can see what you have been doing (which you are describing also in the `status.txt` file).

3 Presentation [35 points]

In class, either Dec. 7th or Dec. 9th. The schedule will be posted after the project proposals are reviewed. Prepare a brief presentation with slides, to highlight parts of the code you have written. Probably you will show transition diagrams or pieces of them from your universal TM. You can also present your tape layout, and your high-level idea for the algorithm. The exact time limit will be determined after the project proposals are reviewed. Probably it will be 6-8 minutes. So probably 5-7 slides will be enough. If you work with a partner, you both need to talk at least a little, during the presentation.

4 Final submission [150 points]

Write a PDF document called `final.pdf` that explains:

- what you were able to get working (including your debug function)
- what is not working, if anything
- testing you have done
- high-level explanation of the different parts of your universal TM; this can also be a guide to your code. Probably your transition diagram will not fit but you can include it in smallish size if you want.

Additionally, your code for the universal TM (in `UniversalTM.hs`), or as much of it as you have been able to get working, should be commented so we can try to understand how it works. You could put a comment to explain what each group of transitions is supposed to do. Even better would be to break up the definition of the transitions into several helper definitions, and comment those. That would make it easier to display parts of your TM and might also make debugging easier (just a thought).