Turing Machine Documentation

Ryan Peruski, Maria Hernandez

November 19, 2023

1 Roles

- **Ryan Peruski**: Created initial stack design and initial templates for the documentation. Revised the machine's design and suggested improvements. Added examples to the documentation.
- **Maria Hernandez**: Edited the stack design to adapt to the specific problem of "valid parentheses", added string and character constant parsing sections, and multi-line comments section in the TM and the documentation. Added examples in the documentation.

2 Definition

This Turing Machine inputs any C-code and places opening parentheses, square brackets, curly brackets, or opening multi-line comment symbols on the stack and pop their respective closing operators. If the machine encounters a closing operator and it does not see the matching operator of the right type on the top of the stack (represented by the first blank to the right), it rejects. If the Turing Machine finishes reading the input and there is something on the stack, it will reject. If the original input is empty, it will accept.

In addition, this Turing Machine pseudo-parses strings and constant characters, by applying certain syntax rules explained below.

3 States, Transitions, Image

The Turing machine operates by moving between states and performing transitions on the tape. The states and transitions are labeled as follows:

- *Red* : Reject states (q_{reject})
- *Green* : Accept state (q_{accept})
- *Blue* : Single quotes parsing
- *Pink* : Double quotes parsing

- *Cyan* : Stack's push states
- *LightCoral* : Stack's pop states
- *Purple* : Multi-line comments section
- *Black* : Initial states to set up the #
- *Beige* : Transition state (between stack push operations, pop operations, single quotes and double quotes parsing, and multi-line comments)
- *Gray* : State to determine the validity of the original input tape

Finally, the Turing Machine is shown in Figure 1.

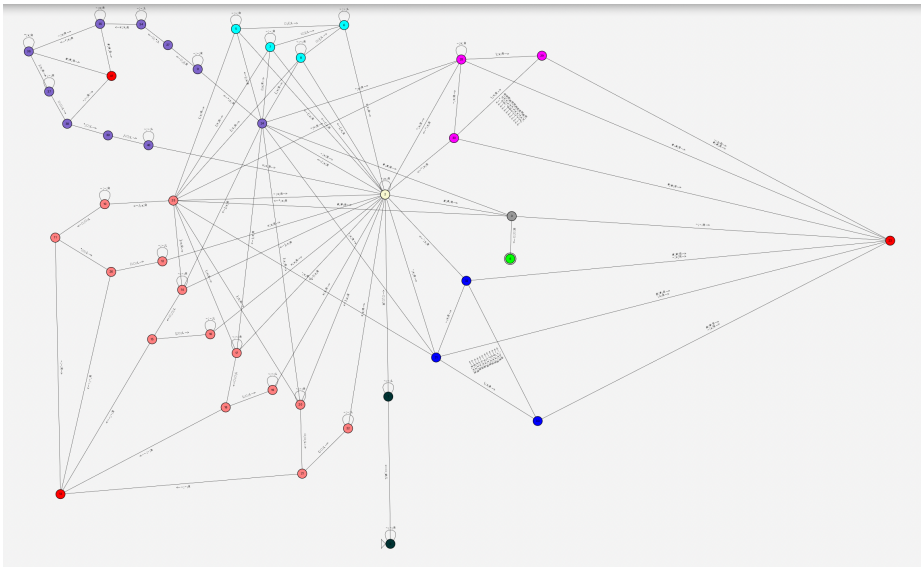

Figure 1: Turing Machine.

4 Machine Specifications

- Input alphabet: Any ASCII symbol, except #, x, and space
- Tape alphabet: Any ASCII symbol, including #, x, and space

4.1 Characters reserved for the tape alphabet

The following are reserved characters: #, x, and spaces. They are part of the tape alphabet, but cannot be part of the input alphabet. # is reserved to delimit the end of the original input data - as soon as the machine starts running it will

mark the end of the original input data. x is reserved to mark the characters that have already been read from the original input data. Space is reserved to know the following: 1. where to place the #, 2. where to start reading the original input data (after placing the # and going back), 3. whether the stack is empty or not after finishing reading the original input data.

The tape will be divided by a #. Anything before the # represents the original input data (code), and everything after the represents the stack.

4.2 How to start the machine

In the original tape, enter any character belonging to the input alphabet (do not enter , x, or space, or the machine will not work properly). The start state is the state 0. The first operation the machine will do is to place the # to delimit the end of the original input data (usually C-code) and the beginning of the "stack". The only elements that can be placed on the stack are the opening operators - namely, (,{,[,/*

5 Functional Description of the Turing Machine

5.1 Valid (,),[,],{,}

As previously mentioned, the TM evaluates whether the (,{,[characters in the original input data have their corresponding closing],},) using stack operations (push for the right ones, pop for the left ones). There is an exception when these characters are either inside double quotes or inside multi-line comments /**/. For instance, the machine will accept /*{*/ and ")" since in C anything inside /**/ is ignored and C allows us to write anything inside "".

5.2 Double quote section ""

This TM is not only evaluating the parity of "", but also other syntax properties of C. If the original input data contains double quotes, the only way the machine will accept it is if it has an even number of double quotes. Namely, it accepts if after opening a double quote and writing any character in the string, the TM finds the respective closing double quote – the only way to ensure this is if the machine goes back to state 2. There is an exception to this rule, the TM may accept an uneven number of " only if some of them are preceded by a \ (escape character). After the escape character, only certain characters are allowed ("',n,0,', a,b,f,r,t,v, ?), we did not account for escape sequences for octal numbers `\000` and hexadecimal numbers `\xhh` since that would have made the TM needlessly complicated.

If the TM is currently at state 28 and it reads a # the original input data has an uneven number of ", hence the machine rejects the string (ex: "Hello""). If the TM reads an escape character in state 28, it goes to state 29. If the TM is at state 29 and the following character in the tape is not any of the special sequence characters or if there are no more characters from the initial data

(next character is #), then the machine rejects the string (ex: "\k, "\). State 30 allows the TM to either read a new escape character \, go back to state 2 - if reading a closing ", read any other character on the string, or reject the string if reading a #.

5.3 Single quote section

This TM is not only evaluating the parity of ', but also other syntax properties of C, including, not accepting an empty character '', or only accepting certain escape sequence characters (the same ones that double quotes accept). The only way to accept an input containing single quotes is if it contains an opening and closing '. An uneven number of ' may be accepted if ' is preceded by an escape character. Even though the usage of multi-character constants is not strictly prohibited in C, functions like "putc" are designed to output only a single character. Hence, when including multi-characters in '', "putc", for instance, will likely just output the lowest byte of the integer – if the machine is little endian. Due to this behavior, we are restricting our machine to only accept one character (maximum two if the input data includes the escape sequence) between ''. If while in state 31, the machine reads a # (end of original input data), or a ', the machine rejects it because either there is an opening ' without its corresponding closing ', or because it is an empty character. If while in state 32, the machine reads another character other than the closing ', the machine rejects the input – this machine does not allow multiple characters inside ''.

5.4 Comments

The operators that gave us the most work were /* and */. Since the machine reads one character at a time, we needed to take into account several possible scenarios so that the machine always halts, and correctly behaves in cases when the original tape has something like */, or /(*), or, *(*), or *[/} (reject in this case). Also, we needed a special branch (purple branch) for the comments so that we could ignore everything inside /* */.

If the TM reads a / followed by a *, it X them out and then places them on the "stack" section. Then the machine goes back and X out all characters until reading both a * followed by a /. The machine rejects if while reading characters, it finds the end of the original input data # before finding */ (ex: #: /*Hello#, /*Hello*#, /*Hello***#). If the TM reads a */ in the original input data, and it does not find the corresponding opening /* in the stack section, it rejects (ex: */#, Hello*/#). If the TM finds nested comments, it rejects (ex: /*/**/*/#, /*/*****H**i*/*/), in this case, the /* inside /**/ is considered a comment, thus is ignored, consequently the last */ will not have its corresponding opening /*. Everything inside the comments is ignored, hence in this case the TM allows single {,[,(,),],}.

To be able to correctly implement the comments this machine needs to find two consecutive characters in the input tape. Thus, this TM accounts for cases when it reads from the input tape a single * or a single / immediately followed

by one of the other special characters {,[,(,),],},",',#, this is why states 24 and 25 have several transitions - to account for cases like *(*), or *[/}.

6 Examples

Here are some examples of input and output for the Turing machine:

- `Input: v[i]=a[i][j];, Output:  xxxxxxxxxxxxx#, ACCEPTS`
- `Input: for(;;)break;, Output:   xxxxxxxxxxxxx#, ACCEPTS`
- `Input: {x=7}, Output:  xxxxx#, ACCEPTS`
- `Input: for(i=0;i<v.size();i++)v[i]=2;,`
  `Output: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx#, ACCEPTS`
- `Input: v[i]=a[i, Output:  xxxxxxxx#[, REJECTS`
- `Input: for(;;, Output: xxxxxx#(, REJECTS`
- `Input: printf("HelloWorld!\n");, Output: xxxxxxxxxxxxxxxxxxxxxx#, ACCEPTS`
- `Input: printf("Hereisa{(");, Output:  xxxxxxxxxxxxxxxxxx#, ACCEPTS`
- `Input: """, Output: xxx#, REJECTS`
- `Input: 'a''\n', Output:  xxxxxxx#, ACCEPTS`
- `Input: /*WeSupportComments!*/, Output: xxxxxxxxxxxxxxxxxxxx#, ACCEPTS`
- `Input: /*WeSupportComments!{{{{*/, Output: xxxxxxxxxxxxxxxxxxxxxxxx#, ACCEPTS`
- `Input: /*/*/, Output: xxxxx# , ACCEPTS`
- `Input: /*/*, Output: xxxx#/* , REJECTS`
- `Input: */*/, Output:  xx*/#  , REJECTS`
- `Input: /*a*b*/, Output: xxxxxxx# , ACCEPTS`
- `Input: /*HelloWorld*/*/, Output: xxxxxxxxxxxxxxx#  , REJECTS`
- `Input: /*/**/, Output:  xxxxxx# , ACCEPTS,`
  `NOTE: It accepts because it treats the inner /* as a comment, but normal compilers throw a warning in these cases.`
- `Input: /*}*/, Output: xxxxx# , ACCEPTS,`
  `NOTE: } doesn't have the opening {, but this accepts because it treats anything inside /**/ as comments.`
- `Input: /*/**/*/, Output: xxxxxxxx# , REJECTS,`
  `NOTE: last */ does not have an opening /*. Compilers do not accept nested comments.`

- Input: /*a*b* , Output: xxxxxx#/* , REJECTS
- Input: (/, Output: xx#(, REJECTS
- Input: (/*****)*/* , Output: xxxxxxxxxxx#(, REJECTS, NOTE: needs closing).
- Input: (/*****)*/*), Output: xxxxxxxxxxxx# , ACCEPTS
- Input: (/*****)*/*)*, Output: xxxxxxxxxxxxx# , ACCEPTS
- Input: (/*)Hi***/}"", Output: xxxxxxxxxxx""#(, REJECTS,
  NOTE: rejects as soon as } doesn't match (
- Input: *{/}, Output: xxxx# , ACCEPTS
- Input: /(*), Output: xxxx# , ACCEPTS
- Input: *(*), Output: xxxx# , ACCEPTS
- Input: *[/}, Output: xxxx# , ACCEPTS
- Input: '\k', Output: xxx'# , REJECTS
- Input: '\n, Output: xxx# , REJECTS
- Input: 'a', Output: xxx# , ACCEPTS
- Input: '\'', Output: xxxx# , ACCEPTS
- Input: "\"\0"\'\r", Output: xxxxxxxxxxx# , REJECTS
- Input: "\"\0"\'\r', Output: xxxxxxxxxxx# , ACCEPTS,
  NOTE: The TM is evaluating the parity of "" and '' (it is not evaluating
  whether it makes sense for C to divide "\"\0" by '\r'
- Input: "\tHello\\\", Output: xxxxxxxxxxxx# , REJECTS
- Input: "\tHello\\\"", Output: xxxxxxxxxxxxx# , ACCEPTS
- Input: /"\tHi", Output: xxxxxxx# , ACCEPTS
- Input: "{"{[/*}*/]}, Output: xxxxxxxxxxxx# , ACCEPTS
- Input: '{'{[/*/*(}[*/]}, Output: xxxxxxxxxxxxxxx# , ACCEPTS
- Input: "{"{[/*/**/]}, Output: xxxxxxxxxxxx# , ACCEPTS
- Input: "{"{(/*/*(}[*/]}, Output: xxxxxxxxxxxxxx}#{(, REJECTS

7 Conclusion

This Turing Machine represents a system designed specifically for processing and analyzing C code. Its primary function is to systematically evaluate the syntactical correctness of parentheses, brackets, and comment structures within C code, ensuring that each opening symbol has a corresponding and appropriately placed closing symbol. The machine achieves this by utilizing a state transition mechanism that accounts for various scenarios and exceptions inherent in C syntax.

Key Features:

- Handling of Code Structures: The TM adeptly manages typical C code constructs, including parentheses, square brackets, and curly brackets, ensuring that they are correctly paired and nested. This includes special handling for code within double quotes, single quotes, and comments, where the usual syntax rules differ.
- Complex State Transitions: The machine is equipped with a range of states to handle different aspects of C code syntax. These states enable pseudo-parse string literals and constant characters and deal with comment blocks.
- Reserved Characters: The use of reserved characters (#, x, and spaces) is a critical aspect of the TM's design. These characters facilitate the machine's operations, such as marking the end of input data and tracking read characters, without interfering with the input code analysis.
- Syntax Rule Application: The machine pseudo-parses strings and characters by applying specific syntax rules of C. This includes handling escape sequences within strings and ensuring the correct use of single and double quotes.
- Error Detection and Rejection Criteria: The TM is designed to reject incorrect code structures, such as unmatched parentheses or nested comments.
- Special Handling of Comments: This TM ignores content within comment blocks while ensuring the structural integrity of these blocks.

In summary, this Turing Machine is a specialized tool for verifying some syntax rules of C code.