Steps to compiling:

Give every line (including those inside higher level commands) a unique line number

Determine which lines branches link to

Determine the order of all other lines

Replace lines that correspond to steps with those states (for each of the 4 acc values)

Identify consecutive lines that act on the same variable and optimize

Apply *LOADI acc*, *BRANCH,* and *MAP* to change the state transitions of previous lines

Apply *LOADI temp and MAP* to change the symbols written by subsequent lines

Iteratively look for identical state transition tables and combine them

Complete

Optimizations:

Skip the search step when consecutive instructions use the same of adjacent bits

Skip the search step by always going to the un-read that is the same as the current variable first

Skip the search step by arranging variables so at the end of one you end up at the next

Skip the un-read because the next operation is an “other primitive”\*

\*The “other primitive” should only unread the variable if it was directly or indirectly preceded by an unread, otherwise it should preserve the read state

In addition to un-read, nots and zeros can also be done in reverse

Types of errors:

Using the TEMP in a map when it doesn’t exist

If it starts of the first variable it searches for

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | FunctionCall | Instruction | End | Step | State |
| FunctionCall | quasi.next\_quasis[k] | | | |  |
| Instruction |  |
| End |  |
| Step |  |
| State | quasi.transitions[symbol][2] | | | | |