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] | | | | |

Unread all bits UNREAD X

Unread all bits in an element UNREAD A[memory] 0

~~Unread all bits in every element~~

Unread all elements UNREAD A 1

Unread the whole thing UNREAD A 0

Unread all bits in an element UNREAD M[memory][memory] 0

~~Unread all bits in every element in a row~~

~~Unread all bits in every element~~

Unread all elements in a row UNREAD M[memory] 1

Unread a whole row UNREAD M[memory] 0

Unread all elements and rows UNREAD M 1

Unread all rows UNREAD M 2

Unread the whole thing UNREAD M 0

01 X[]

01 23 X[][]

01 23 -> 0123 X[FLAT][]

01 23 45 67 X[][][]

01 23 45 67 -> 01 23 45 67 X[FLAT][][]

01 23 45 67 -> 0123 4567 X[][FLAT][]

01 23 45 67 -> 01234567 X[FLAT][FLAT][]

|  |  |  |  |
| --- | --- | --- | --- |
| X[NEXT] | X[NEXTBIG] | X[NEXTRED] | X[NEXTBIGRED] |
| LOADNEXT X | LOADNEXTBIG X | LOADNEXTRED X | LOADNEXTBIGRED X |
| X[RED] | X{BIGRED] | X[] | X[BIG] |

|  |  |  |  |
| --- | --- | --- | --- |
| X[NEXT][NEXT] | X[][NEXT] | X[NEXT][] |  |
| LOADNEXT X[NEXT] | LOADNEXT X[] | LOADNEXT X |  |
| X[RED][RED] | X[][RED] | X[RED][] |  |

next, big, red = False, big, next^red

**X / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 Y**

**X / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 Y**

**X / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 Y**

**X / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 / 0 0 0 0 Y**

LOAD X[] X, right to first unread comma, right to first unread bit

LOAD X[BIG] Y, left to first unread comma, right to first unread bit

LOAD X[RED] Y, left to first red comma, right to first unread bit

LOAD X[BIGRED] X, right to first red comma, right to first unread bit

*LOADBIG X[]* *X, right to first unread comma, right to comma, left to first unread bit*

LOADBIG X[BIG] X, right to first red comma, left to first unread bit

LOADBIG X[RED] X, right to first unread comma, left to first unread bit

*LOADBIG X[BIGRED]* *X, right to first red comma, right to comma, left to first unread bit*

*LOADRED X[] X, right to first unread comma, right to comma, left to first red bit*

LOADRED X[BIG] X, right to first red comma, left to first red bit

LOADRED X[RED] X, right to first unread comma, left to first red bit

*LOADRED X[BIGRED]* *X, right to first red comma, right to comma, left to first red bit*

LOADBIGRED X[] X, right to first unread comma, right to first red bit

LOADBIGRED X[BIG] Y, left to first unread comma, right to first red bit

LOADBIGRED X[RED] Y, left to first red comma, right to first red bit

LOADBIGRED X[BIGRED] X, right to first red comma, right to first red bit

0.big==0.red & 1.big^1.red Y, left to first

Else X, right to first

0.big==0.red ? 1.red : ~1.big red comma,

Else unread comma,

0.big==0.red || 1.big^1.red -

Else right to comma,

0.big==0.red right to

Else left to

0.red red bit

Else unread bit

Reverse the value of big if red is true

LOAD X[] X, right to first unread comma, right to first unread bit

LOAD X[BIG] Y, left to first unread comma, right to first unread bit

LOAD X[RED] X, right to first red comma, right to first unread bit

LOAD X[BIGRED] Y, left to first red comma, right to first unread bit

*LOADBIG X[]* *X, right to first unread comma, right to comma, left to first unread bit*

LOADBIG X[BIG] X, right to first red comma, left to first unread bit

*LOADBIG X[RED]* *X, right to first red comma, right to comma, left to first unread bit*

LOADBIG X[BIGRED] X, right to first unread comma, left to first unread bit

LOADRED X[] X, right to first unread comma, right to first red bit

LOADRED X[BIG] Y, left to first unread comma, right to first red bit

LOADRED X[RED] X, right to first red comma, right to first red bit

LOADRED X[BIGRED] Y, left to first red comma, right to first red bit

*LOADBIGRED X[] X, right to first unread comma, right to comma, left to first red bit*

LOADBIGRED X[BIG] X, right to first red comma, left to first red bit

*LOADBIGRED X[RED]* *X, right to first red comma, right to comma, left to first red bit*

LOADBIGRED X[BIGRED] X, right to first unread comma, left to first red bit

~0.big & 1.big Y, left to first

Else X, right to first

1.red ^ (0.big & 1.big) red comma,

Else unread comma,

0.big & ~1.big right to comma,

Else ,

0.big left to

Else right to

0.red red bit

Else unread bit

**Search from the [Cn+1?“right”:“left”] until you find the first [(Bn ^ An&Cn)?“red”:“unread”]**

**[(!An & Cn)?“then go to the right again”:“”]**

**An = bign ^ redn**

**Bn = redn**

**Cn+1 = An & !Cn**

**C0 = false**