Universal register machine, U

High-level specification

Universal RM U carries out the following computation, starting with $\mathtt{R}_0 = 0$, $\mathtt{R}_1 = e$ (code of a program), $\mathtt{R}_2 = a$ (code of a list of arguments) and all other registers zeroed:

- decode e as a RM program P
- decode a as a list of register values $a_1, \ldots, a_n$
- carry out the computation of the RM program P starting with $\mathtt{R}_0 = 0, \mathtt{R}_1 = a_1, \ldots, \mathtt{R}_n = a_n$ (and any other registers occurring in P set to 0).

Mnemonics for the registers of $\boldsymbol{U}$ and the role they play in its program:

$\mathtt{R}_1 \equiv \mathtt{P}$ code of the RM to be simulated

$\mathtt{R}_2 \equiv \mathtt{A}$ code of current register contents of simulated RM

$\mathtt{R}_3 \equiv \mathtt{PC}$ program counter—number of the current instruction (counting from $\mathbf{0}$)

$\mathtt{R}_4 \equiv \mathtt{N}$ code of the current instruction body

$\mathtt{R}_5 \equiv \mathtt{C}$ type of the current instruction body

$\mathtt{R}_6 \equiv \mathtt{R}$ current value of the register to be incremented or decremented by current instruction (if not $\mathtt{HALT}$)

$\mathtt{R}_7 \equiv \mathtt{S}$, $\mathtt{R}_8 \equiv \mathtt{T}$ and $\mathtt{R}_9 \equiv \mathtt{Z}$ are auxiliary registers.

$\mathtt{R}_0$ result of the simulated RM computation (if any).

Overall structure of U's program

$\boxed{1}$ copy $\mathtt{PC}$th item of list in $\mathtt{P}$ to $\mathtt{N}$ (halting if $\mathtt{PC} >$ length of list); goto $\boxed{2}$

$\boxed{2}$ if $\mathtt{N} = 0$ then halt, else decode $\mathtt{N}$ as $\langle\!\langle y, z \rangle\!\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto $\boxed{3}$

{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$,
or $\mathtt{C} = 2i + 1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k \rangle$}

$\boxed{3}$ copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto $\boxed{4}$

$\boxed{4}$ execute current instruction on $\mathtt{R}$; update $\mathtt{PC}$ to next label; restore register values to $\mathtt{A}$; goto $\boxed{1}$

Overall structure of U's program

$\boxed{1}$ copy $\mathtt{PC}$th item of list in $\mathtt{P}$ to $\mathtt{N}$ (halting if $\mathtt{PC}$ > length of list); goto $\boxed{2}$

$\boxed{2}$ if $\mathtt{N} = 0$ then halt, else decode $\mathtt{N}$ as $\langle\!\langle y, z \rangle\!\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto $\boxed{3}$

{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$, or $\mathtt{C} = 2i + 1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k \rangle$}

$\boxed{3}$ copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto $\boxed{4}$

$\boxed{4}$ execute current instruction on $\mathtt{R}$; update $\mathtt{PC}$ to next label; restore register values to $\mathtt{A}$; goto $\boxed{1}$

To implement this, we need RMs for manipulating (codes of) lists of numbers. . .

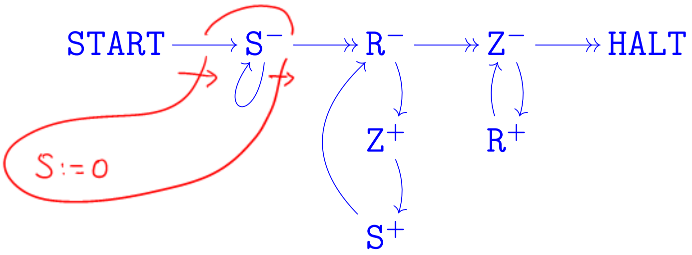
The program $\mathtt{START} \longrightarrow \boxed{\mathtt{S} ::= \mathtt{R}} \longrightarrow \mathtt{HALT}$

to copy the contents of $\mathtt{R}$ to $\mathtt{S}$ can be implemented by

$$
\begin{tikzcd}
\mathtt{START} \arrow[r] & \mathtt{S}^- \arrow[r, two heads] \arrow[loop below] & \mathtt{R}^- \arrow[r, two heads] \arrow[d, bend left] & \mathtt{Z}^- \arrow[r, two heads] \arrow[d, bend left] & \mathtt{HALT} \\
 & & \mathtt{Z}^+ \arrow[d, bend left] & \mathtt{R}^+ \arrow[u, bend left] & \\
 & & \mathtt{S}^+ \arrow[uu, bend left=60] & &
\end{tikzcd}
$$

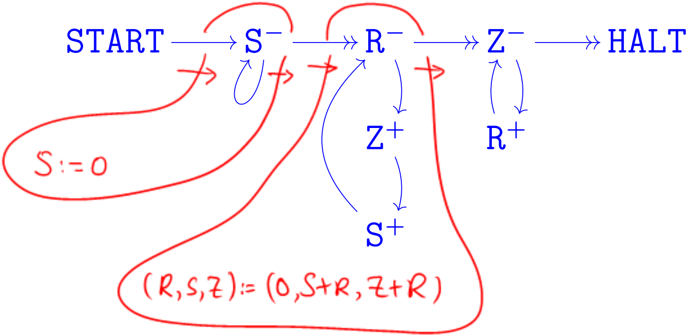
The program $\mathtt{START} \longrightarrow \boxed{\mathtt{S} ::= \mathtt{R}} \longrightarrow \mathtt{HALT}$

to copy the contents of $\mathtt{R}$ to $\mathtt{S}$ can be implemented by



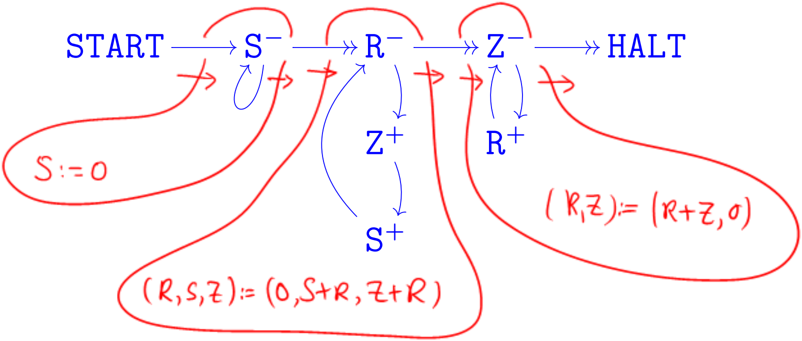
The program $\texttt{START} \longrightarrow \boxed{\texttt{S} ::= \texttt{R}} \longrightarrow \texttt{HALT}$

to copy the contents of $\texttt{R}$ to $\texttt{S}$ can be implemented by



The program START $\longrightarrow$ $\boxed{\texttt{S ::= R}}$ $\longrightarrow$ HALT

to copy the contents of R to S can be implemented by



The program $\mathtt{START} \longrightarrow \boxed{\mathtt{S} ::= \mathtt{R}} \longrightarrow \mathtt{HALT}$

to copy the contents of $\mathtt{R}$ to $\mathtt{S}$ can be implemented by

$$
\begin{tikzcd}
\mathtt{START} \arrow[r] & \mathtt{S}^- \arrow[r, two heads] \arrow[loop below] & \mathtt{R}^- \arrow[r, two heads] \arrow[d] & \mathtt{Z}^- \arrow[r, two heads] \arrow[d, bend left] & \mathtt{HALT} \\
& & \mathtt{Z}^+ \arrow[d] & \mathtt{R}^+ \arrow[u, bend left] & \\
& & \mathtt{S}^+ \arrow[uu, bend left=60] & &
\end{tikzcd}
$$

precondition:
$\mathtt{R} = x$
$\mathtt{S} = y$
$\mathtt{Z} = 0$

postcondition:
$\mathtt{R} = x$
$\mathtt{S} = x$
$\mathtt{Z} = 0$

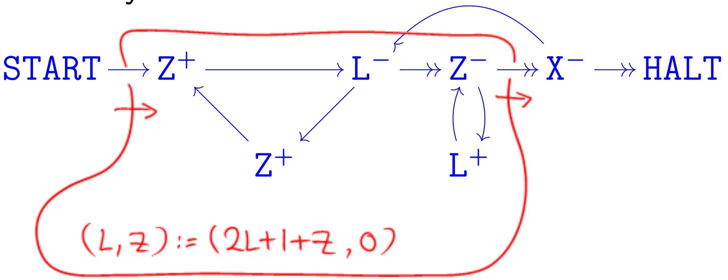
The program $\mathtt{START} \longrightarrow \boxed{\begin{matrix} \textit{push } \mathtt{X} \\ \textit{to } \mathtt{L} \end{matrix}} \longrightarrow \mathtt{HALT}$ $\boxed{2^{\mathtt{X}}(2\mathtt{L}+1)}$

to carry out the assignment $(\mathtt{X}, \mathtt{L}) ::= (0, \mathtt{X} :: \mathtt{L})$ can be implemented by

$$
\begin{tikzcd}
\mathtt{START} \arrow[r] & \mathtt{Z}^+ \arrow[rr] & & \mathtt{L}^- \arrow[r, two heads] \arrow[dl] & \mathtt{Z}^- \arrow[r, two heads] \arrow[d, bend left] & \mathtt{X}^- \arrow[r, two heads] \arrow[ll, bend right] & \mathtt{HALT} \\
 & & \mathtt{Z}^+ \arrow[ul] & & \mathtt{L}^+ \arrow[u, bend left] & &
\end{tikzcd}
$$

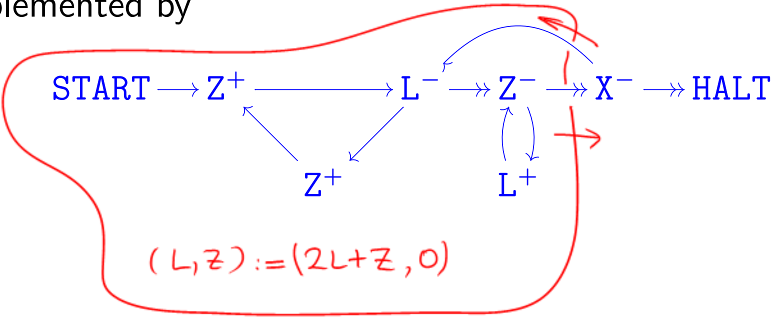
The program $\mathtt{START} \longrightarrow \boxed{\begin{matrix}\textit{push } \mathtt{X} \\ \textit{to } \mathtt{L}\end{matrix}} \longrightarrow \mathtt{HALT}$

to carry out the assignment $(\mathtt{X}, \mathtt{L}) ::= (0, \mathtt{X} :: \mathtt{L})$ can be implemented by



The program $\mathtt{START}\longrightarrow\boxed{\begin{array}{c}\textit{push}\ \mathtt{X}\\ \textit{to}\ \mathtt{L}\end{array}}\longrightarrow\mathtt{HALT}$

to carry out the assignment $(\mathtt{X},\mathtt{L}) ::= (0, \mathtt{X} :: \mathtt{L})$ can be implemented by



The program $\mathtt{START} \longrightarrow \boxed{\begin{matrix} \textit{push } \mathtt{X} \\ \textit{to } \mathtt{L} \end{matrix}} \longrightarrow \mathtt{HALT}$

to carry out the assignment $(\mathtt{X}, \mathtt{L}) ::= (0, \mathtt{X} :: \mathtt{L})$ can be implemented by

$$\begin{tikzcd}
\mathtt{START} \arrow[r] & \mathtt{Z}^+ \arrow[rr] & & \mathtt{L}^- \arrow[r, two heads] \arrow[dl] & \mathtt{Z}^- \arrow[r, two heads] \arrow[d, bend left] & \mathtt{X}^- \arrow[r, two heads] \arrow[ll, bend right] & \mathtt{HALT} \\
& & \mathtt{Z}^+ \arrow[ul] & & \mathtt{L}^+ \arrow[u, bend left] & &
\end{tikzcd}$$

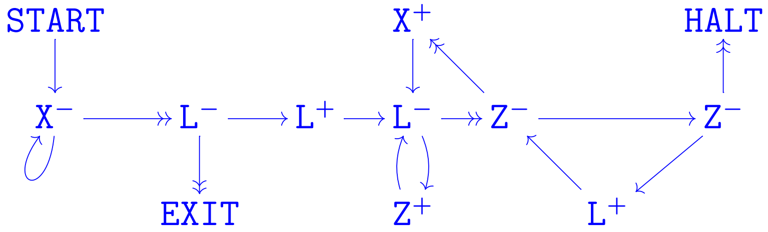
precondition:
$\mathtt{X} = x$
$\mathtt{L} = \ell$
$\mathtt{Z} = 0$

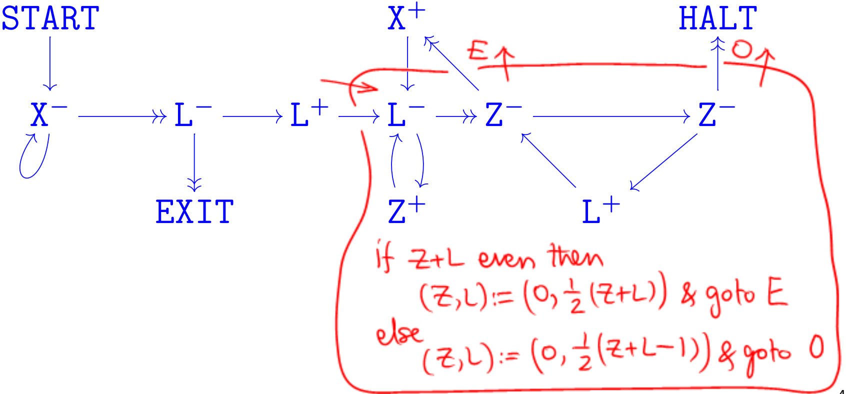
postcondition:
$\mathtt{X} = 0$
$\mathtt{L} = \langle\!\langle x, \ell \rangle\!\rangle = 2^x(2\ell + 1)$
$\mathtt{Z} = 0$

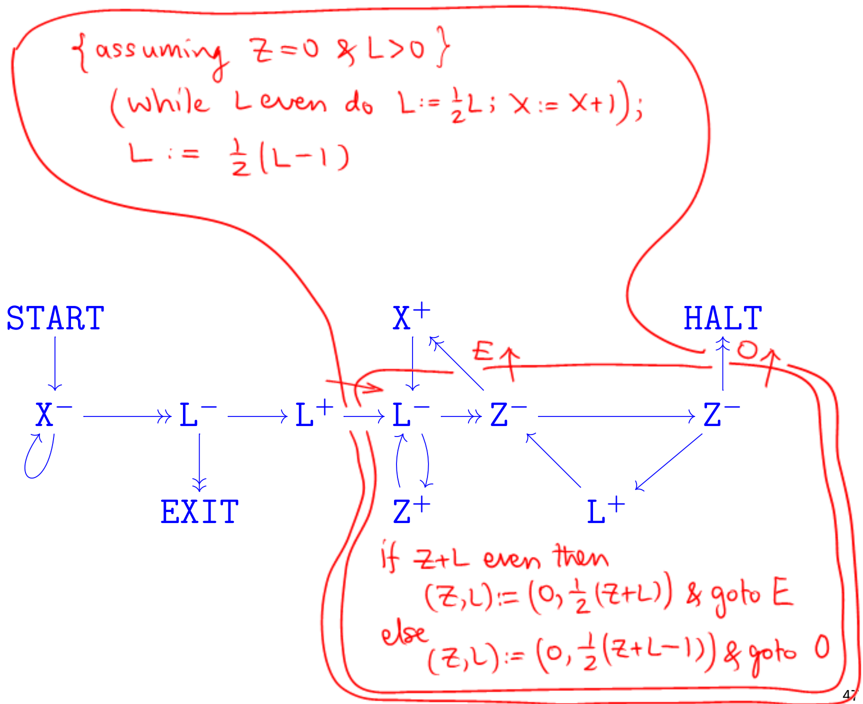
The program $\mathtt{START} \longrightarrow \boxed{\begin{array}{c}\textit{pop } \mathtt{L}\\ \textit{to } \mathtt{X}\end{array}} \begin{array}{l}\longrightarrow \mathtt{HALT}\\ \twoheadrightarrow \mathtt{EXIT}\end{array}$ specified by

"$\textbf{\textit{if}}\ \mathtt{L} = \mathbf{0}\ \textbf{\textit{then}}\ (\mathtt{X} ::= \mathbf{0};\ \textbf{\textit{goto}}\ \mathtt{EXIT})\ \textbf{\textit{else}}$
$\textbf{\textit{let}}\ \mathtt{L} = \langle\!\langle x, \ell \rangle\!\rangle\ \textbf{\textit{in}}\ (\mathtt{X} ::= x;\ \mathtt{L} ::= \ell;\ \textbf{\textit{goto}}\ \mathtt{HALT})$"

can be implemented by



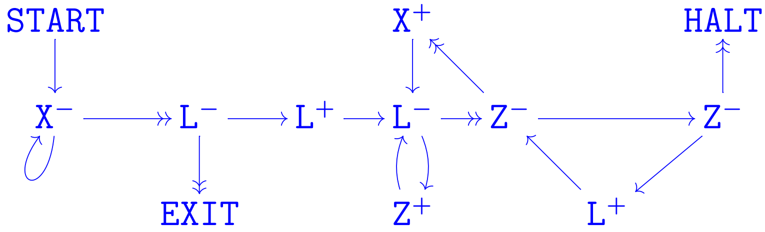

START
HALT
X+
X−
L−
L+
L−
Z−
Z−
EXIT
Z+
L+
E
O
if Z+L even then
(Z,L) := (0, ½(Z+L)) & goto E
else
(Z,L) := (0, ½(Z+L−1)) & goto O


{assuming Z=0 & L>0}
(while L even do L:= ½L; X:= X+1);
L := ½(L−1)
START
X⁻
L⁻
EXIT
L⁺
X⁺
L⁻
Z⁺
Z⁻
E
HALT
O
Z⁻
L⁺
if Z+L even then
(Z,L):= (0, ½(Z+L)) & goto E
else
(Z,L):= (0, ½(Z+L−1)) & goto O

The program $\mathtt{START}\longrightarrow\boxed{\begin{array}{c}\textbf{\textit{pop}}\ \mathtt{L}\\ \textbf{\textit{to}}\ \mathtt{X}\end{array}}\begin{array}{l}\longrightarrow\mathtt{HALT}\\ \twoheadrightarrow\mathtt{EXIT}\end{array}$ specified by

"$\textbf{\textit{if}}\ \mathtt{L} = \mathbf{0}\ \textbf{\textit{then}}\ (\mathtt{X} ::= \mathbf{0};\ \textbf{\textit{goto}}\ \mathtt{EXIT})\ \textbf{\textit{else}}$
$\textbf{\textit{let}}\ \mathtt{L} = \langle\!\langle x, \ell \rangle\!\rangle\ \textbf{\textit{in}}\ (\mathtt{X} ::= x;\ \mathtt{L} ::= \ell;\ \textbf{\textit{goto}}\ \mathtt{HALT})$"

can be implemented by



Overall structure of U's program

$\boxed{1}$ copy $\mathtt{PC}$th item of list in $\mathtt{P}$ to $\mathtt{N}$ (halting if $\mathtt{PC}$ > length of list); goto $\boxed{2}$

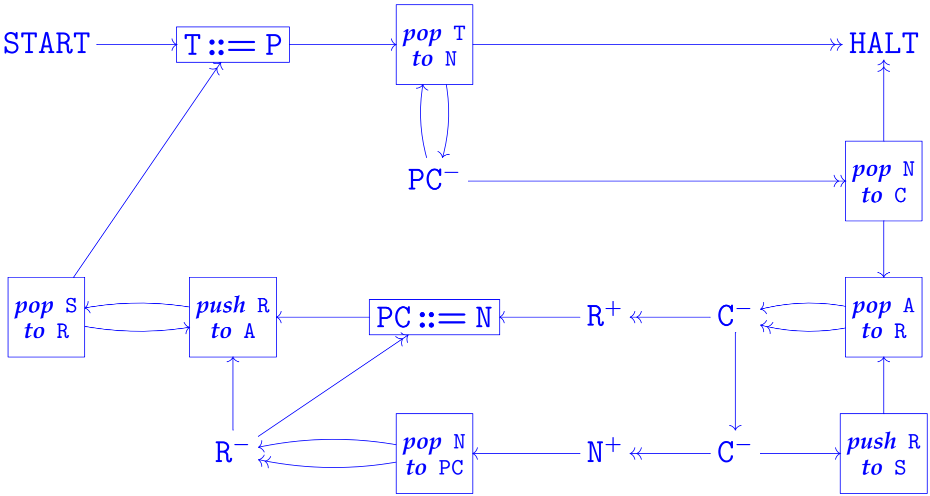
$\boxed{2}$ if $\mathtt{N} = 0$ then halt, else decode $\mathtt{N}$ as $\langle\!\langle y, z \rangle\!\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto $\boxed{3}$

{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$,
or $\mathtt{C} = 2i + 1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k \rangle$}

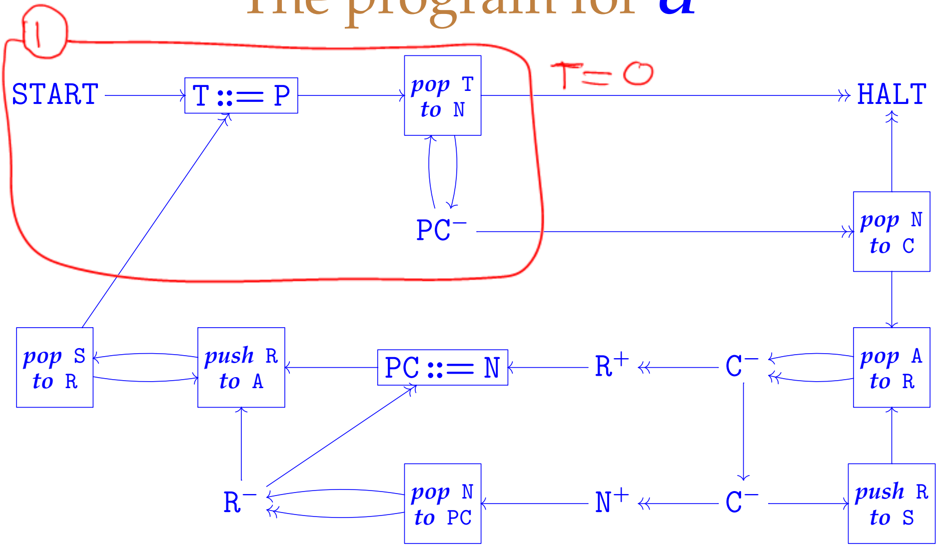
$\boxed{3}$ copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto $\boxed{4}$

$\boxed{4}$ execute current instruction on $\mathtt{R}$; update $\mathtt{PC}$ to next label; restore register values to $\mathtt{A}$; goto $\boxed{1}$

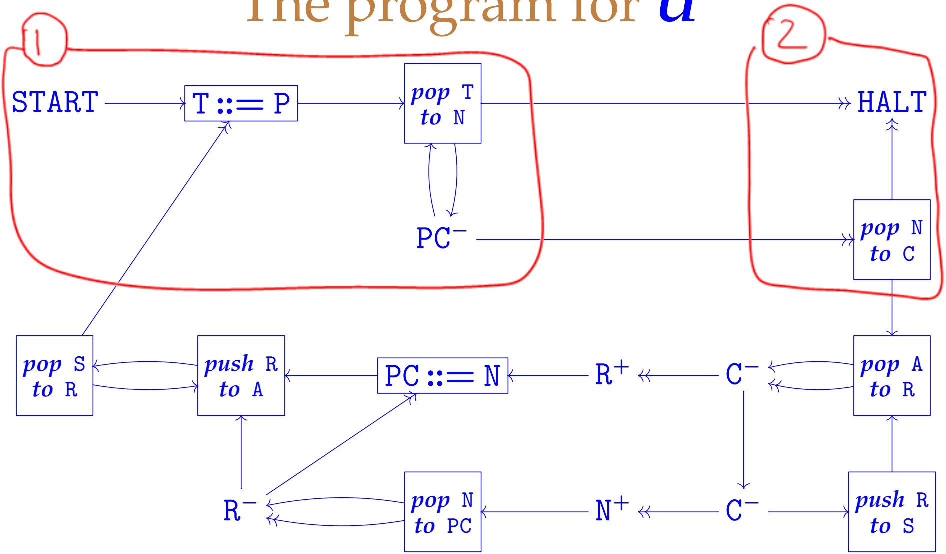
The program for U



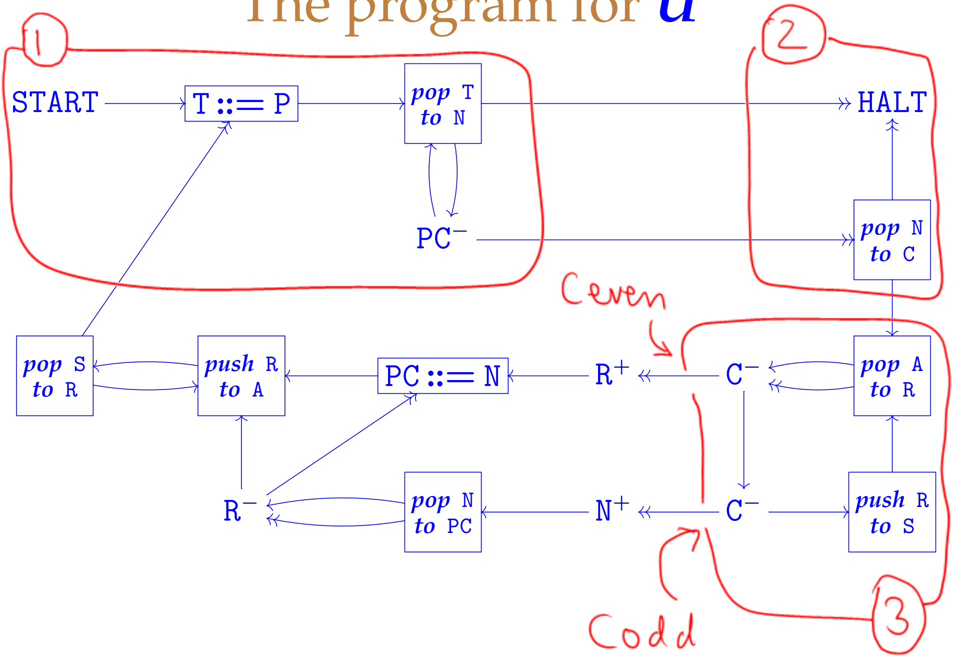
The program for U



The program for U



The program for U



The program for U