Universal Register Machine, U

High-level specification

Universal RM U carries out the following computation, starting with $R_0 = 0$, $R_1 = e$ (code of a program), $R_2 = a$ (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 $R_0 = 0, R_1 = a_1, \ldots, R_n = a_n$ (and any other registers occurring in P set to 0).

Mnemonics for the registers of U and the role they play in its program:

$R_1 \equiv P$ code of the RM to be simulated

$R_2 \equiv A$ code of current register contents of simulated RM

$R_3 \equiv PC$ program counter—number of the current instruction (counting from 0)

$R_4 \equiv N$ code of the current instruction body

$R_5 \equiv C$ type of the current instruction body

$R_6 \equiv R$ current value of the register to be incremented or decremented by current instruction (if not HALT)

$R_7 \equiv S$, $R_8 \equiv T$ and $R_9 \equiv Z$ are auxiliary registers.

R_0 result of the simulated RM computation (if any).

Overall structure of U's program

[1] copy PCth item of list in P to N (halting if PC > length of list); goto [2]

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

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

[3] copy ith item of list in A to R; goto [4]

[4] execute current instruction on R; update PC to next label; restore register values to A; goto [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 $z = \langle j, k \rangle$}

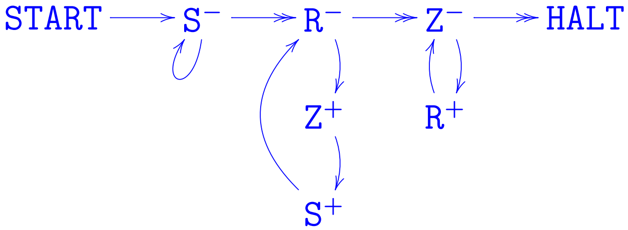
$\boxed{3}$ copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto $\boxed{4}$

$\boxed{4}$ 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. . .

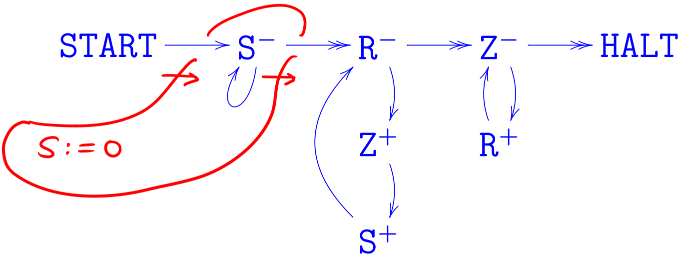
The program START⟶ $\boxed{S ::= R}$ ⟶HALT

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



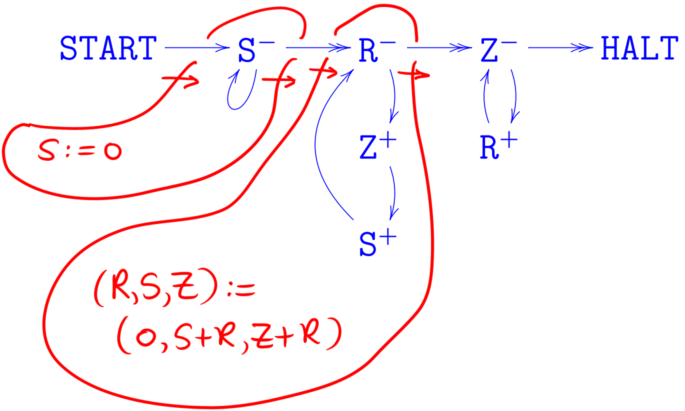
The program $\texttt{START} \longrightarrow \boxed{\texttt{S} ::= \texttt{R}} \longrightarrow \texttt{HALT}$

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



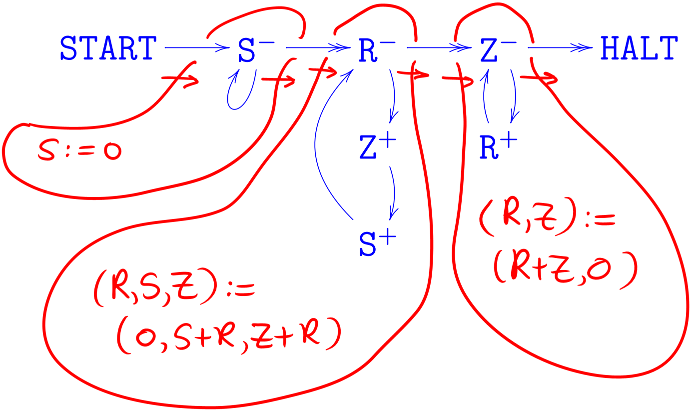
The program START$\longrightarrow$ S ::= R $\longrightarrow$HALT

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



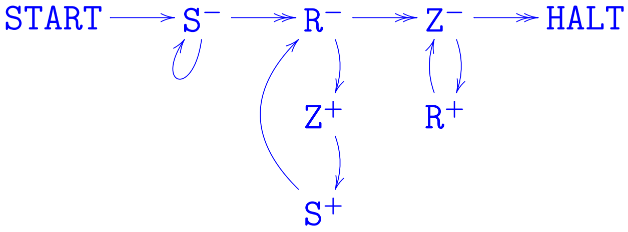
The program START$\longrightarrow$ $\boxed{S ::= R}$ $\longrightarrow$HALT

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



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

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



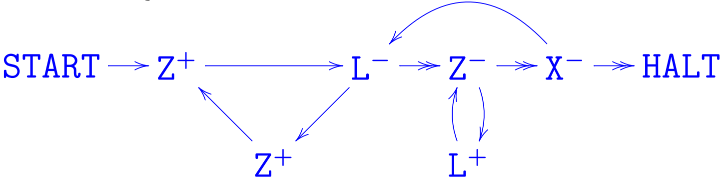
precondition:
$\texttt{R} = x$
$\texttt{S} = y$
$\texttt{Z} = 0$

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

The program START $\longrightarrow$ *push* X *to* L $\longrightarrow$ HALT

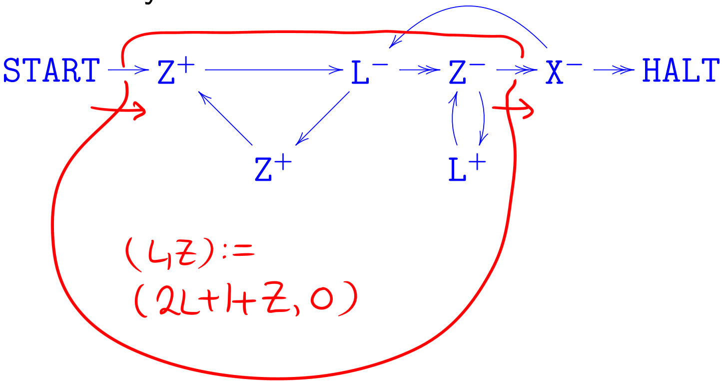
$2^{\mathtt{X}}(2\mathtt{L}+1)$

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



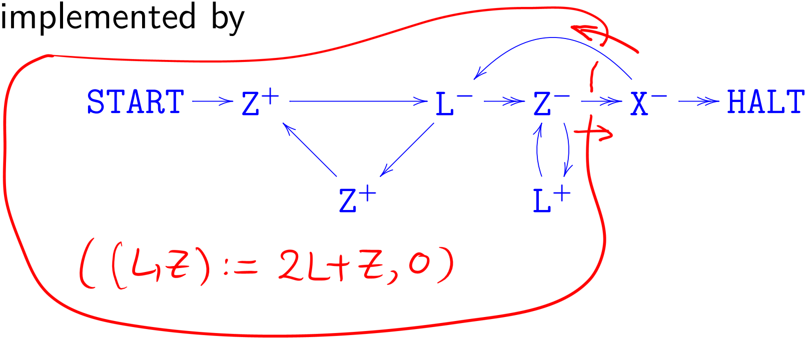
The program $\texttt{START} \longrightarrow$ [*push* X *to* L] $\longrightarrow \texttt{HALT}$

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



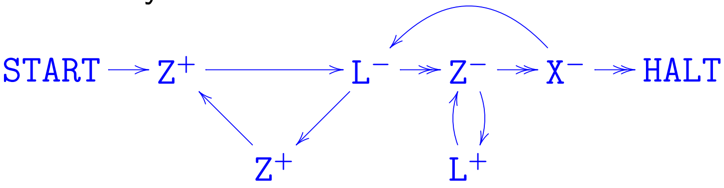
The program START ⟶ [*push* X *to* L] ⟶ HALT

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



The program START $\longrightarrow$ [*push* X *to* L] $\longrightarrow$ HALT

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



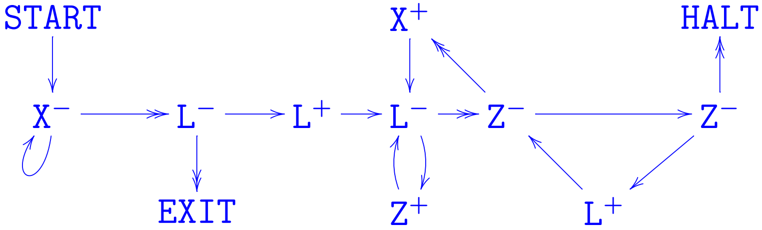
precondition:
$\mathtt{X} = x$
$\mathtt{L} = \ell$
$\mathtt{Z} = 0$

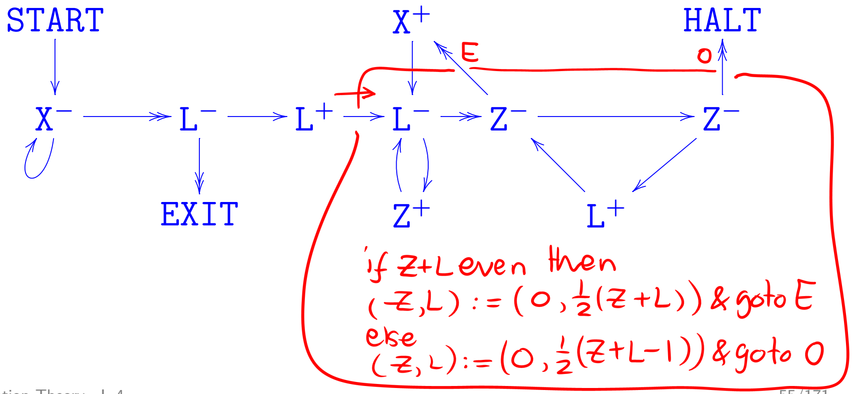
postcondition:
$\mathtt{X} = 0$
$\mathtt{L} = \langle\langle x, \ell \rangle\rangle = 2^x(2\ell + 1)$
$\mathtt{Z} = 0$

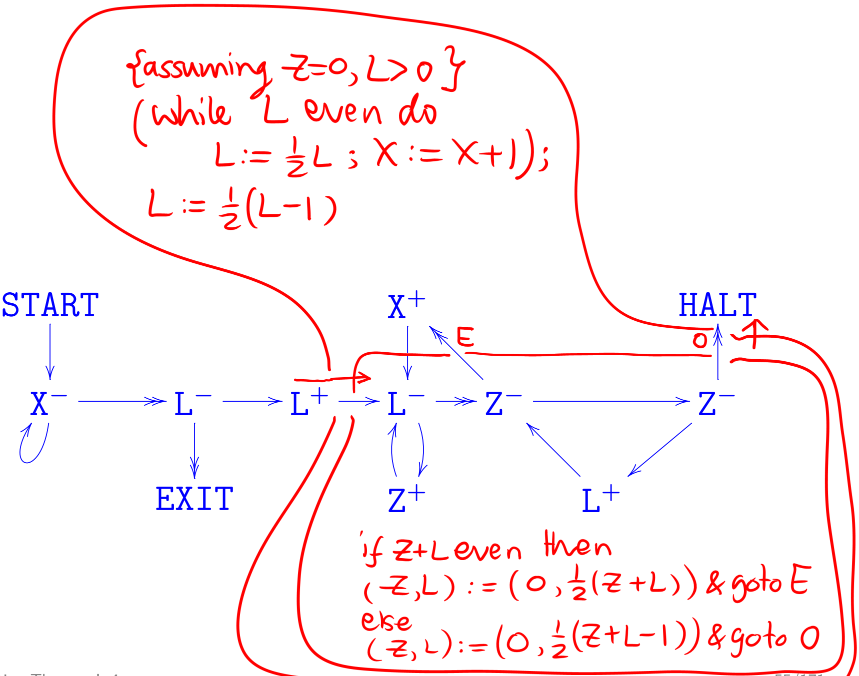
The program $\mathtt{START} \longrightarrow \boxed{\begin{array}{c}\textit{pop}\ \mathtt{L}\\ \textit{to}\ \mathtt{X}\end{array}} \begin{array}{l}\longrightarrow \mathtt{HALT}\\ \twoheadrightarrow \mathtt{EXIT}\end{array}$ specified by

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

can be implemented by



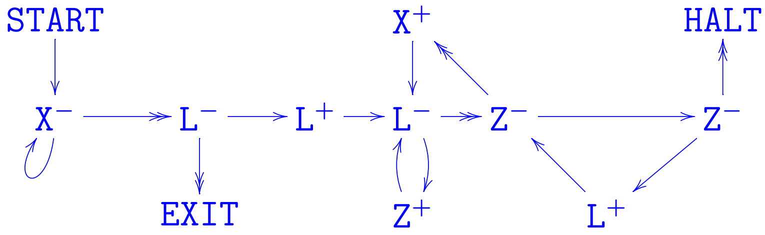

START
X⁻
L⁻
EXIT
L⁺
X⁺
L⁻
Z⁺
Z⁻
HALT
Z⁻
L⁺
E
O
if Z+L even then
(Z,L) := (0, ½(Z+L)) & goto E
else
(Z,L) := (0, ½(Z+L−1)) & goto O


{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
0
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 $\texttt{START}\longrightarrow\boxed{\begin{array}{c}\textit{pop}\ \texttt{L}\\ \textit{to}\ \texttt{X}\end{array}}\begin{array}{l}\longrightarrow\texttt{HALT}\\ \twoheadrightarrow\texttt{EXIT}\end{array}$ specified by

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

can be implemented by



Overall structure of U's program

[1] copy PCth item of list in P to N (halting if PC > length of list); goto [2]

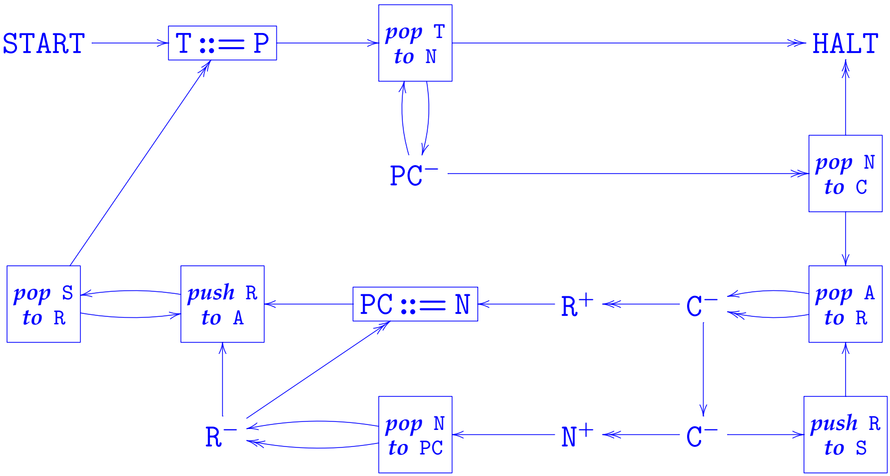
[2] if $N = 0$ then halt, else decode N as $\langle\langle y, z\rangle\rangle$; $C ::= y$; $N ::= z$; goto [3]

{at this point either $C = 2i$ is even and current instruction is $R_i^+ \to L_z$,
or $C = 2i + 1$ is odd and current instruction is $R_i^- \to L_j, L_k$ where $z = \langle j, k\rangle$}

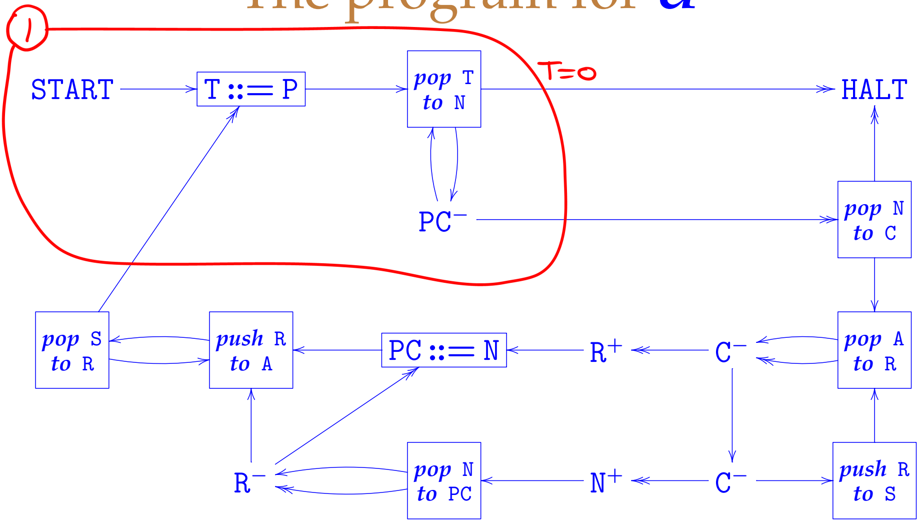
[3] copy ith item of list in A to R; goto [4]

[4] execute current instruction on R; update PC to next label; restore register values to A; goto [1]

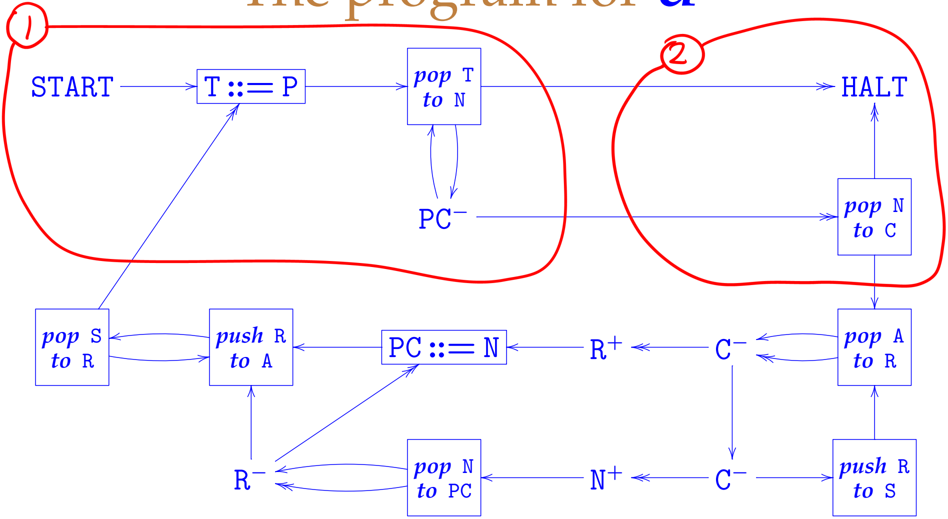
The program for *U*



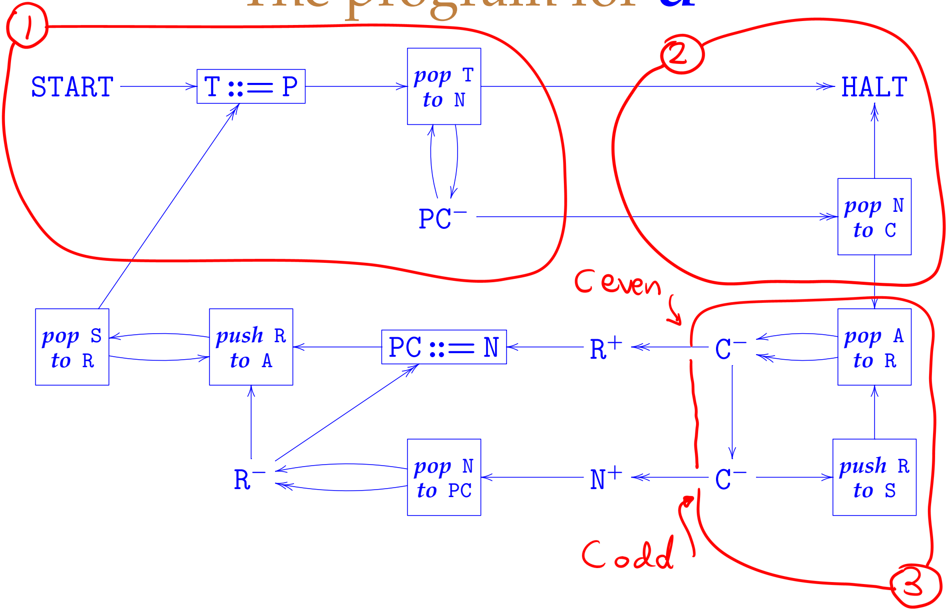
The program for *U*



The program for U



The program for U



The program for U