Big picture

- All languages
- Decidable

  Turing machines
- NP
- P
- Context-free

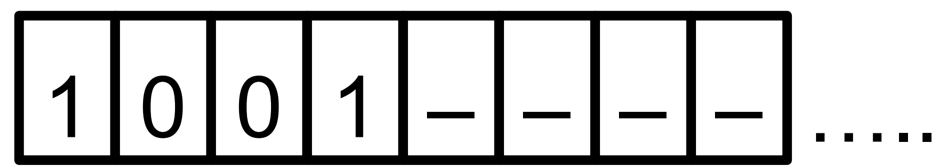
  Context-free grammars, push-down automata
- Regular

  Automata, non-deterministic automata, regular expressions

Turing Machines

Like DFA but

- Access to infinite tape, initially containing input and blank (–) everywhere else

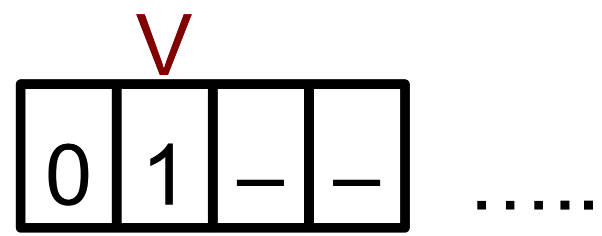


- Read and write on tape
- Move both ways on tape
- Accept, reject take action immediately

Turing Machines (TM)

Details:

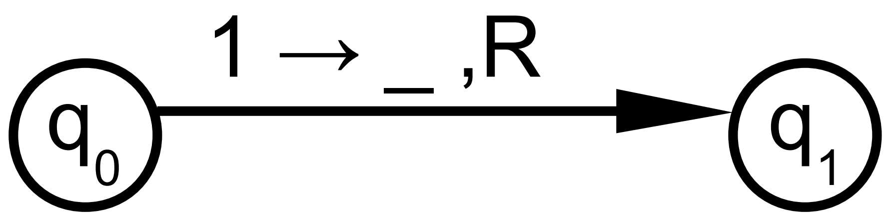
- Tape is infinite to the right only
- TM has head V on one tape cell



- In one step TM can:
- change state
- read/write cell under head,
- move to the left or right of 1 cell

(If TM attempts to go left of first cell, stay put)

May write TM like DFA with transitions:



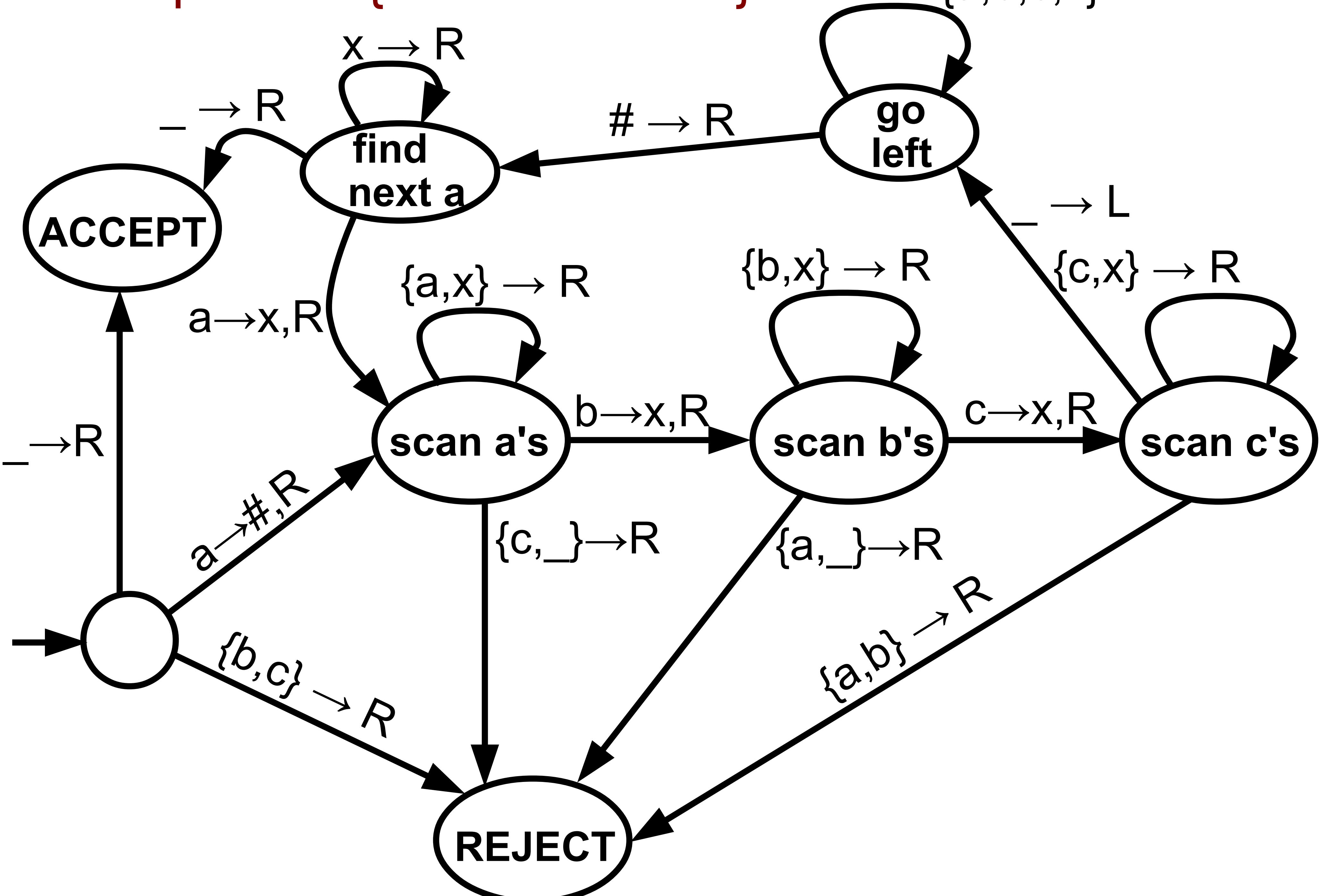
- If in state q_0 and tape cell under head contains 1:

  write blank (_),

  move head to the Right,

  go to state q_1

Example: $L = \{a^n b^n c^n : n \geq 0\}$



- Typically, we do not draw state diagrams of TM
- Two reasons:
- State diagrams are very complicated, hence useless
- There is equivalent, easier notation (we'll see later)
- Sufficient to give high-level description of TM

Example: A TM for the language $\{a^n b^n c^n : n \geq 0\}$

M := “On input w.

1) Scan tape and cross off one a, one b, and one c

2) If none of these symbols is found, ACCEPT

3) If not all of these symbols is found,

   or if found in the wrong order, REJECT

4) Go back to 1.”

- State diagram merely implements above

Example: A TM for the language $\{a^n b^n c^n : n \geq 0\}$

M := "On input w.

1) Scan tape and cross off one a, one b, and one c

2) If none of these symbols is found, ACCEPT

3) If not all of these symbols is found,

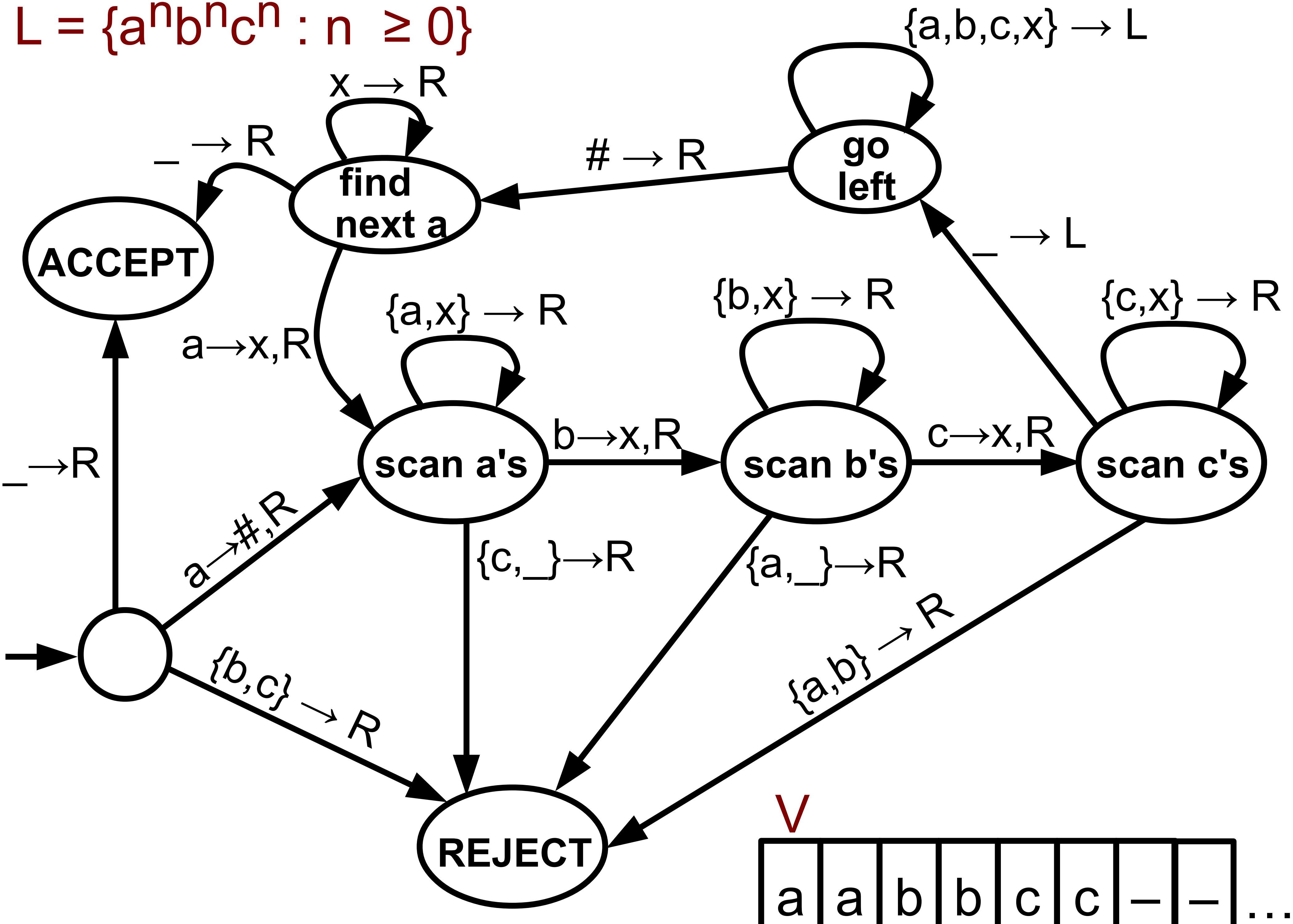
   or if found in the wrong order, REJECT

4) Go back to 1."

- State diagram merely implements above

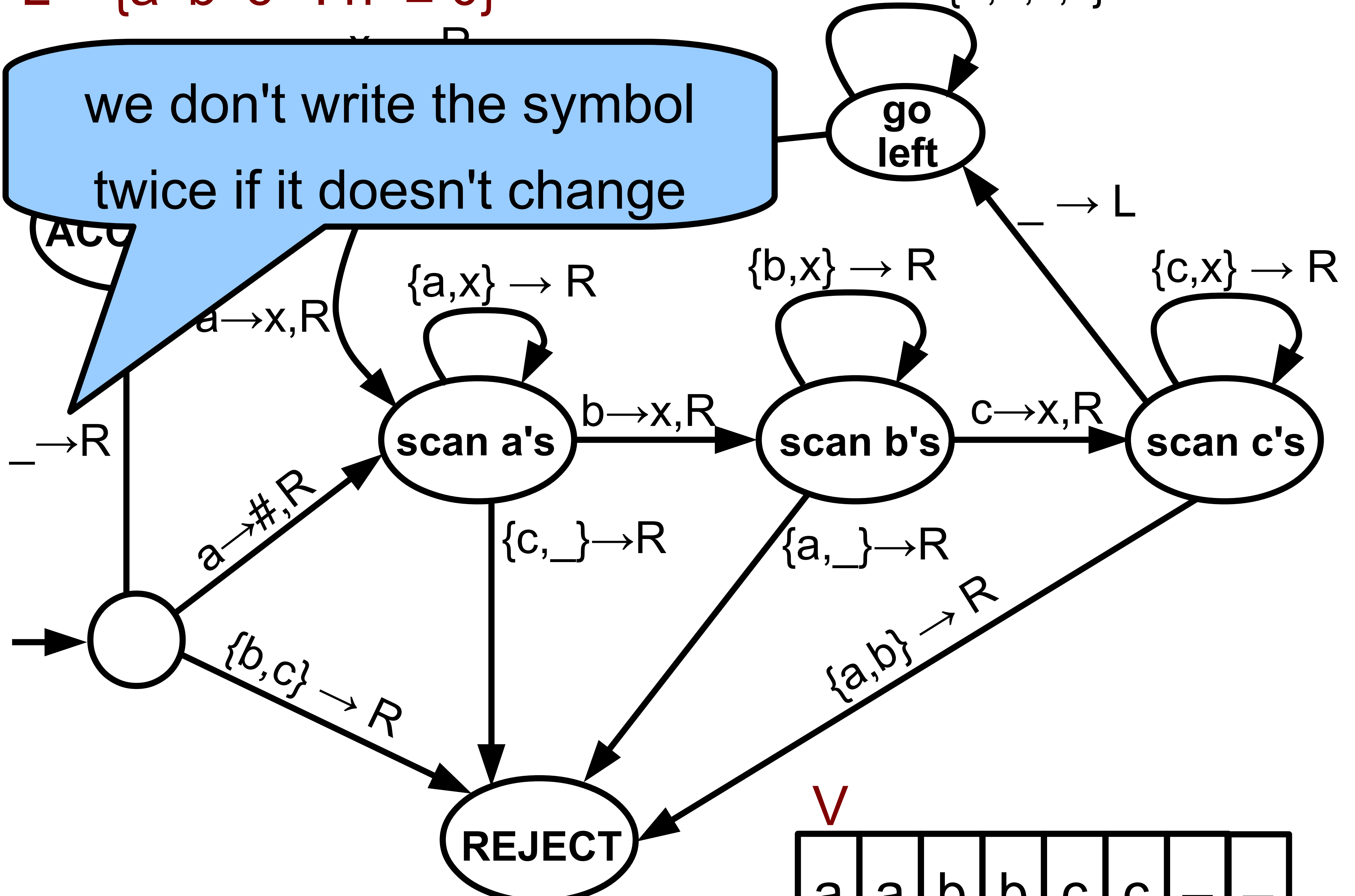
Have extra tape symbols #, X

L = {$a^n b^n c^n$: n ≥ 0}

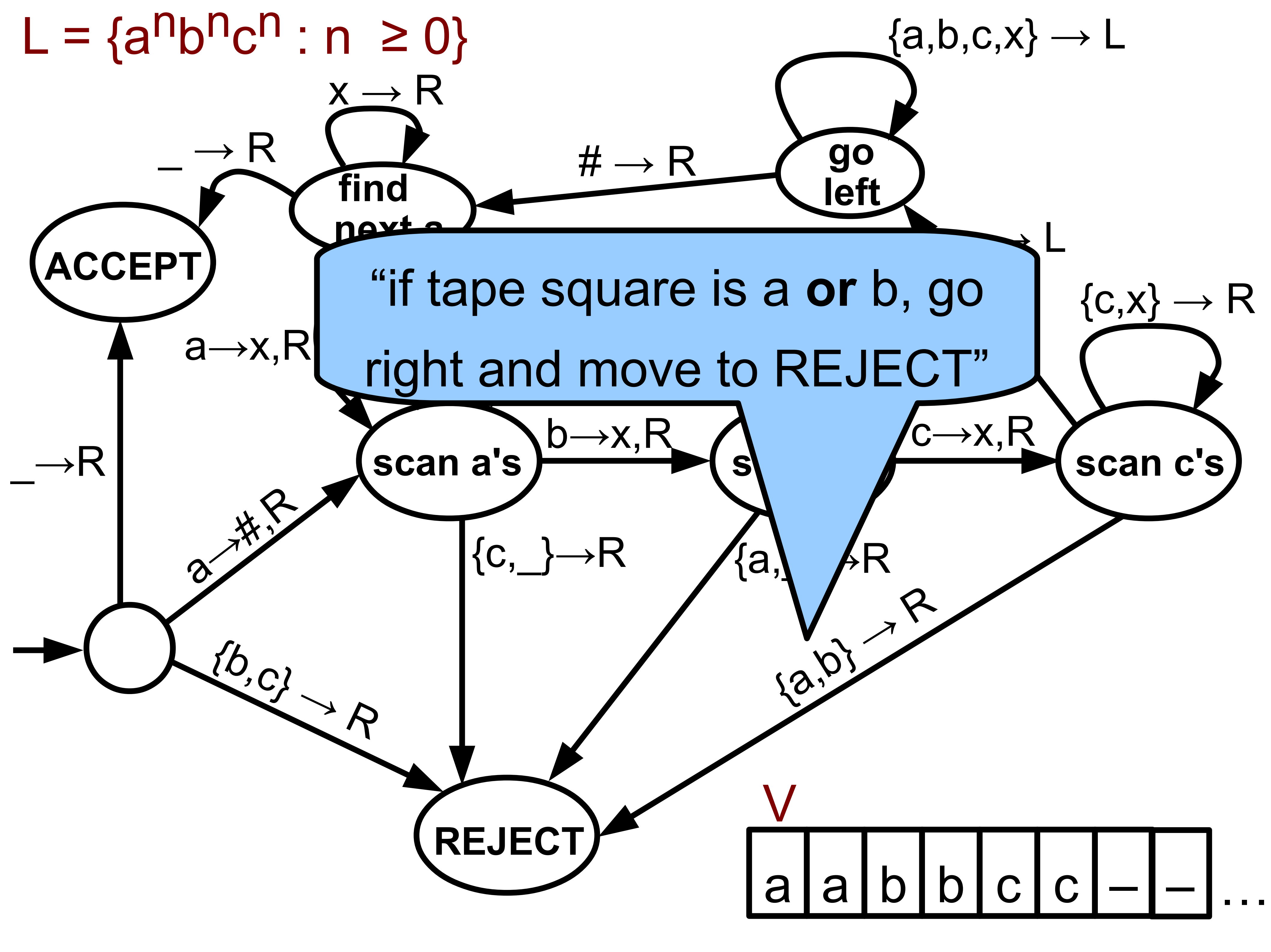


{a,b,c,x} → L
x → R
_ → R
go left
find next a
→ R
ACCEPT
a→x,R
_ → L
{a,x} → R
{b,x} → R
{c,x} → R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a
a
b
b
c
c
_
_
...

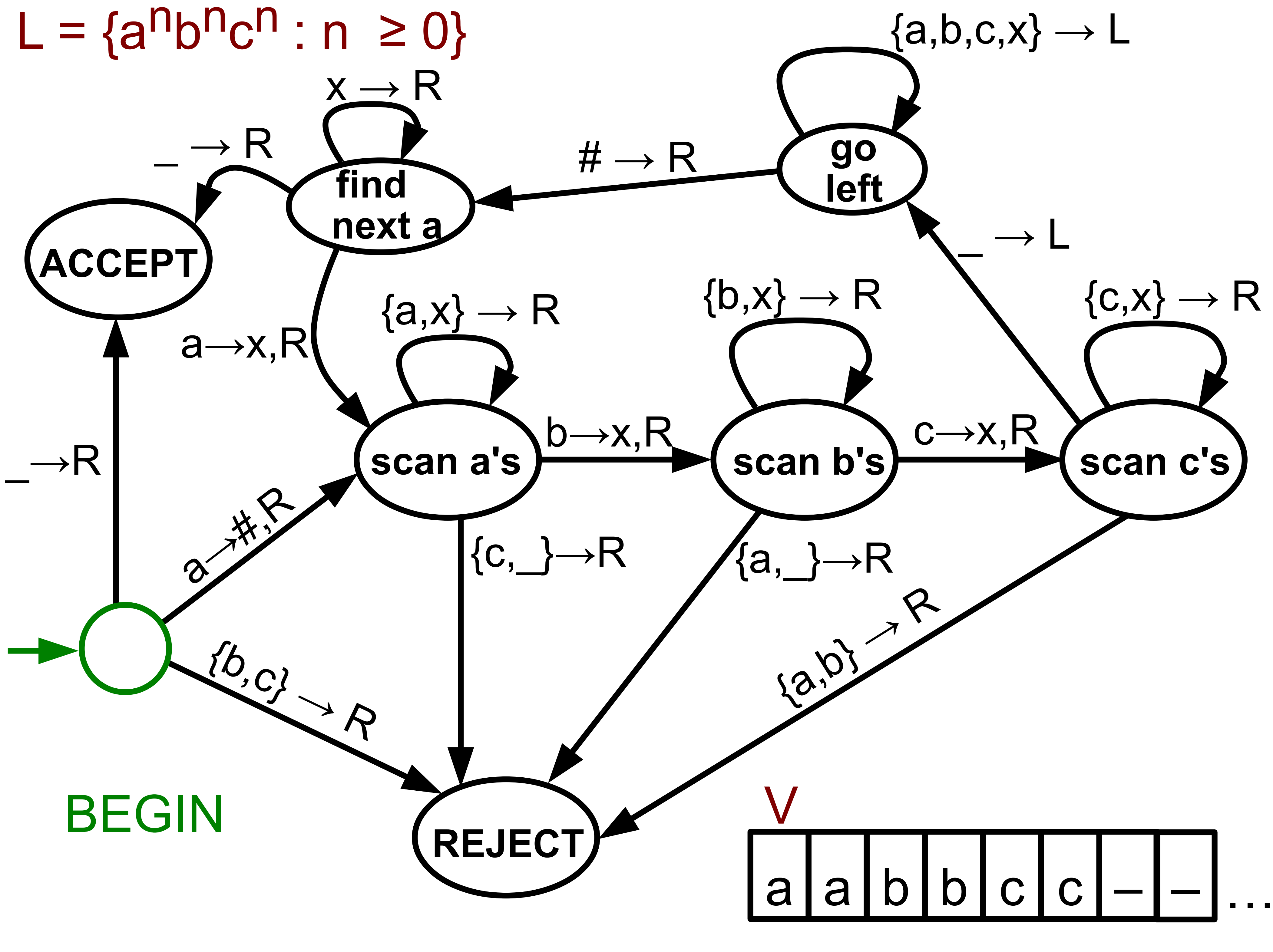
L = {$a^n b^n c^n$: n ≥ 0}



we don't write the symbol twice if it doesn't change
{a,b,c,x} → L
go left
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
a a b b c c _ _ …

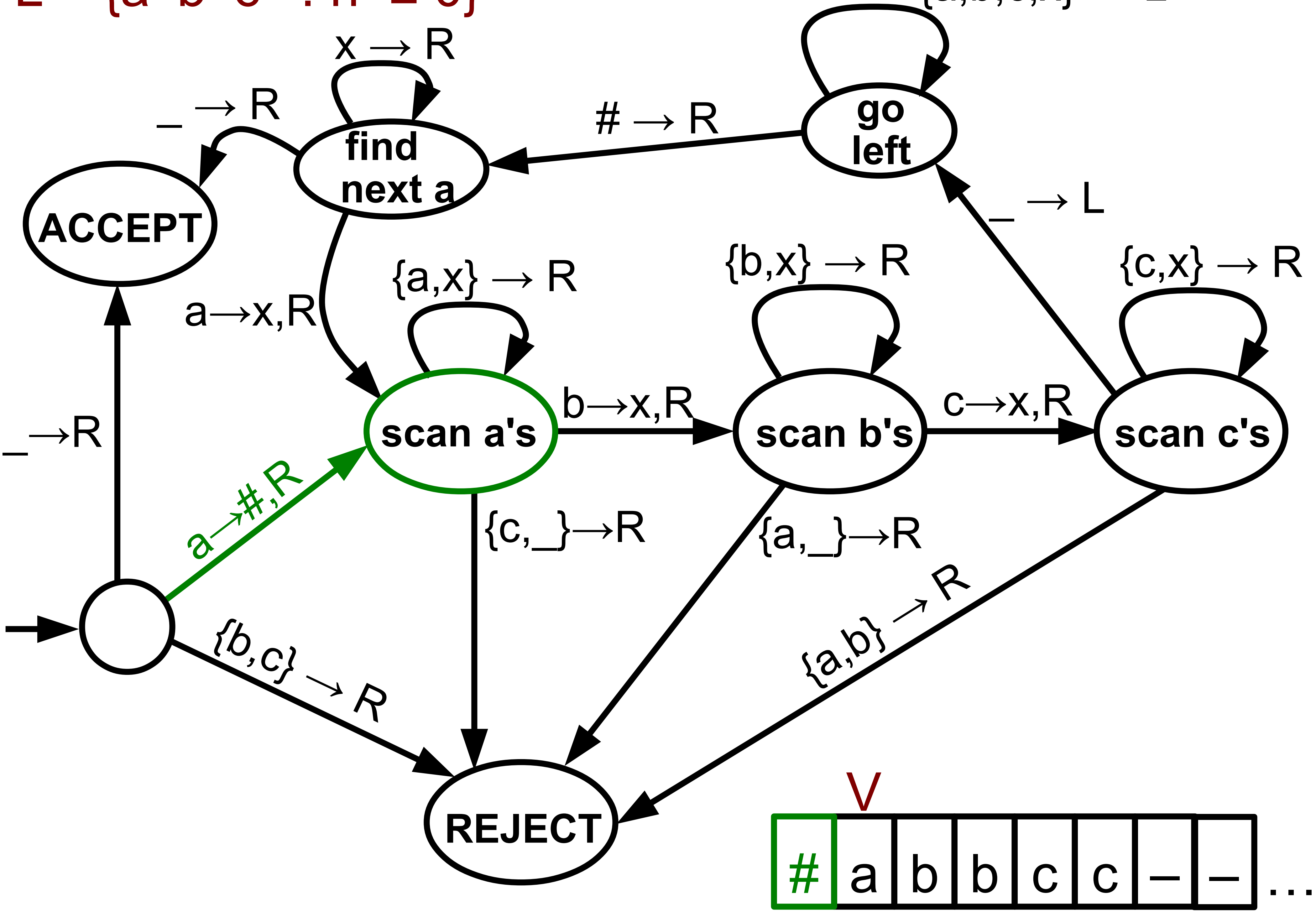


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
→ L
"if tape square is a **or** b, go right and move to REJECT"
{c,x} → R
a→x,R
scan a's
b→x,R
c→x,R
scan c's
_→R
a→#,R
{c,_}→R
→R
{a,b} → R
{b,c} → R
REJECT
V
a a b b c c _ _ …

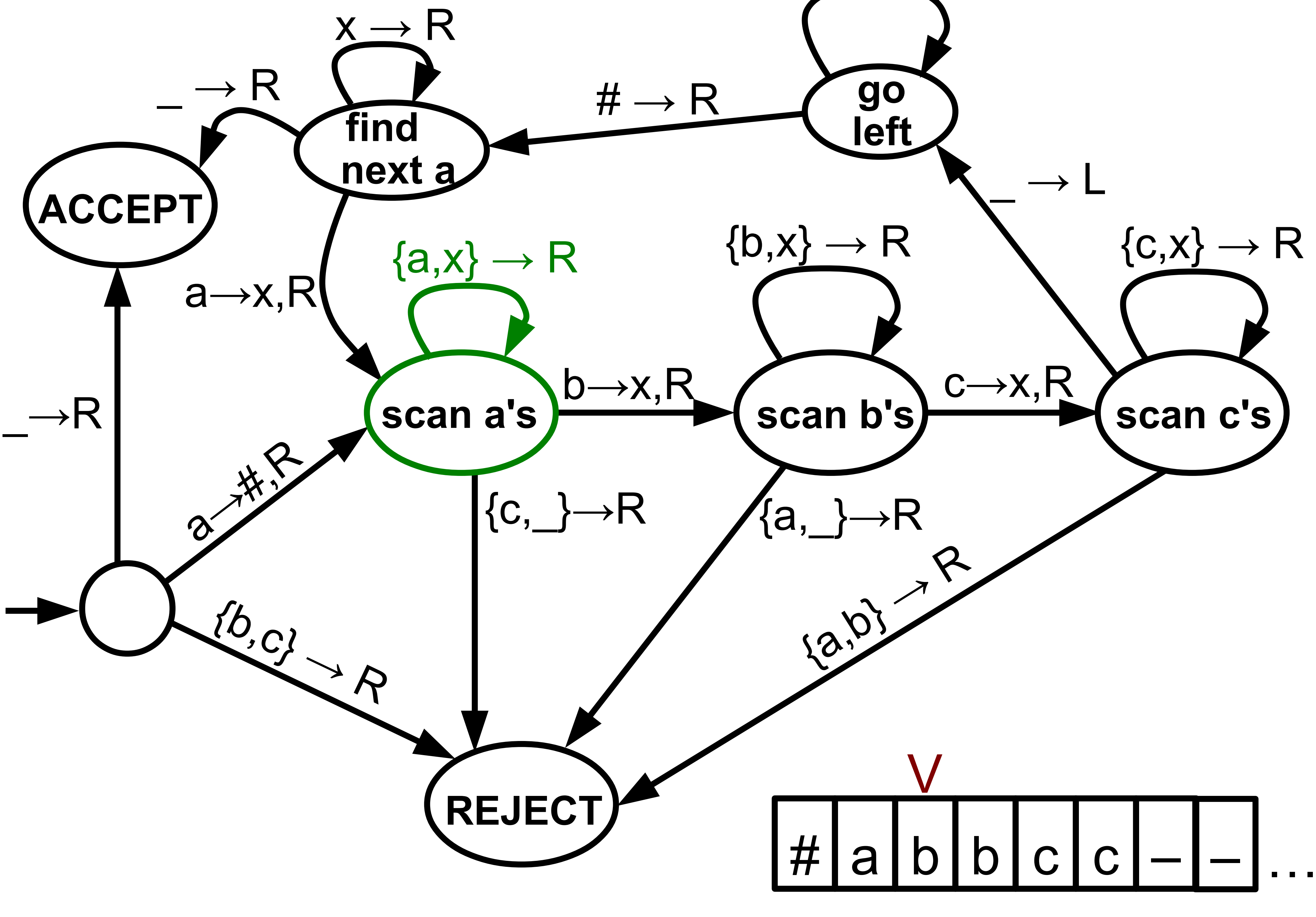


L = {aⁿbⁿcⁿ : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
BEGIN
REJECT
V
a a b b c c _ _ ...

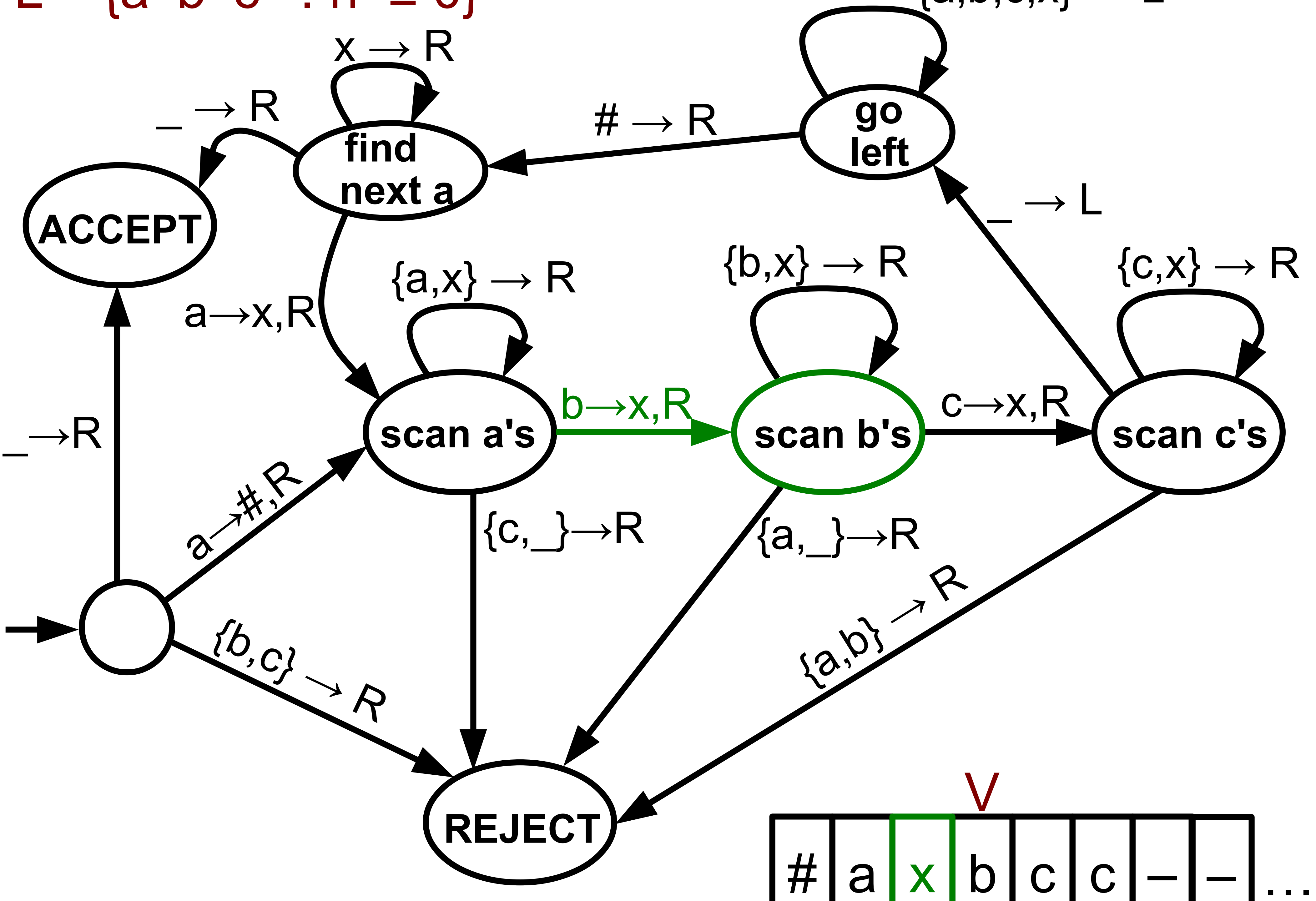
$L = \{a^n b^n c^n : n \geq 0\}$

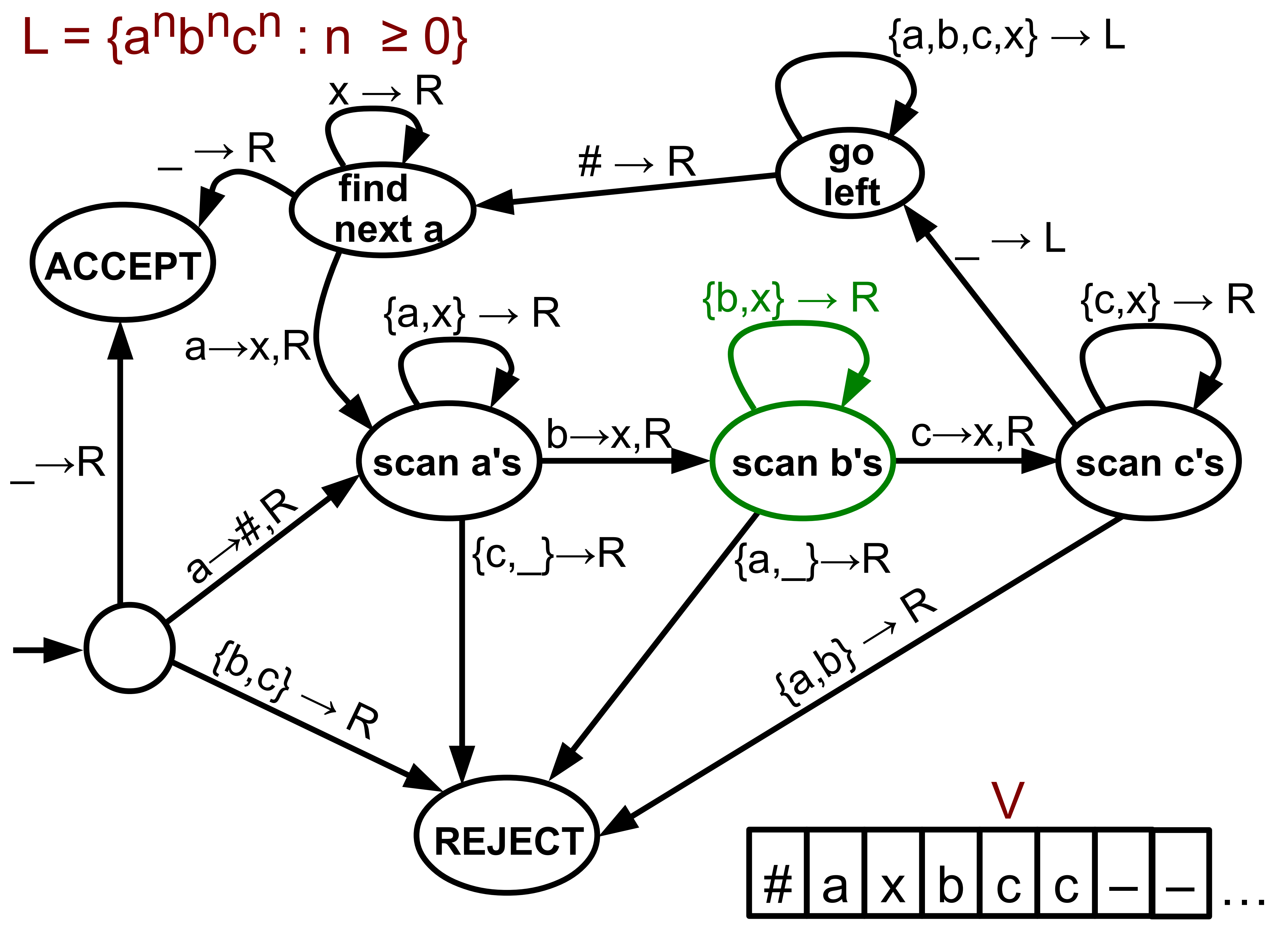


$L = \{a^n b^n c^n : n \geq 0\}$



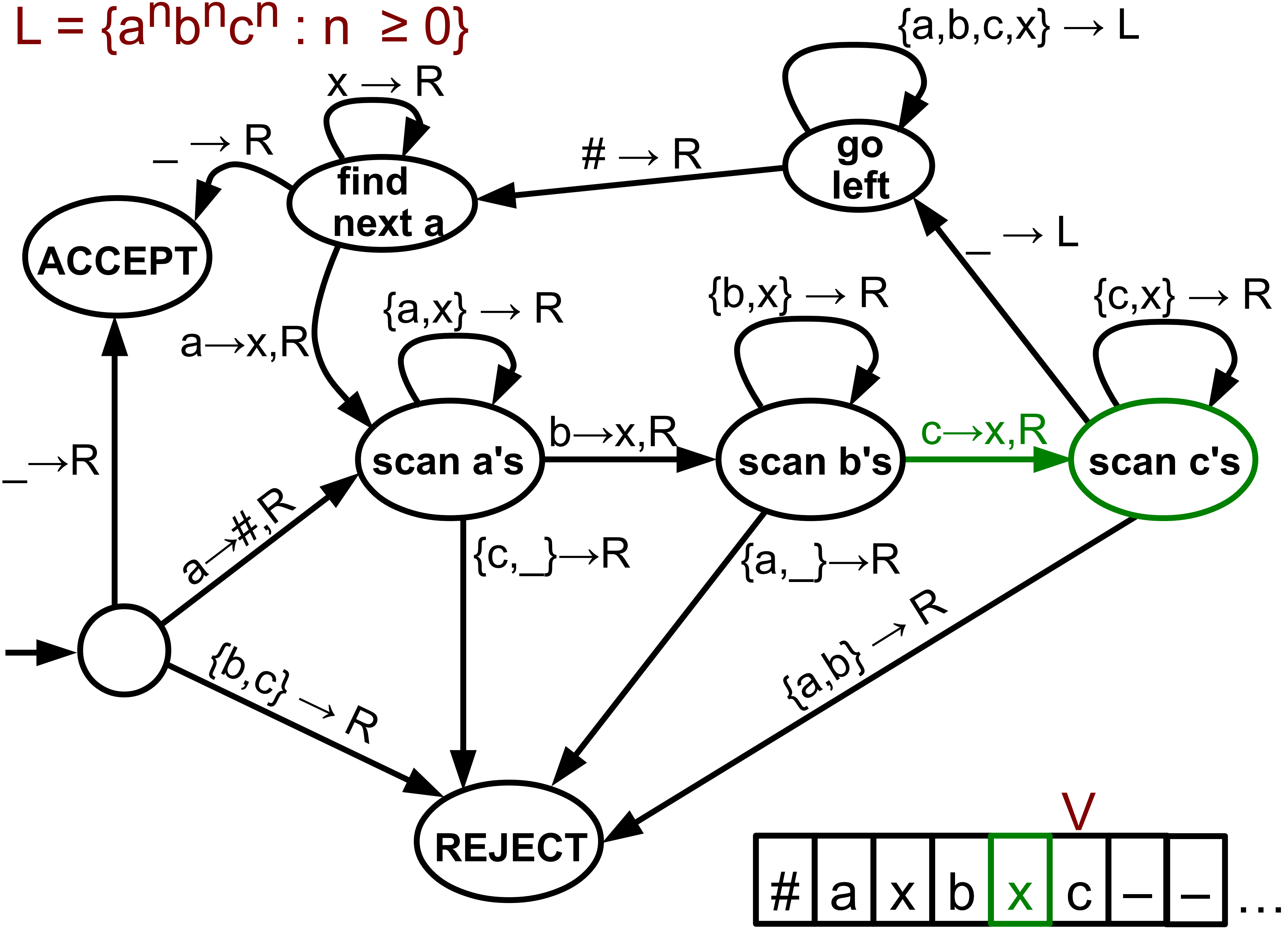
$L = \{a^n b^n c^n : n \geq 0\}$



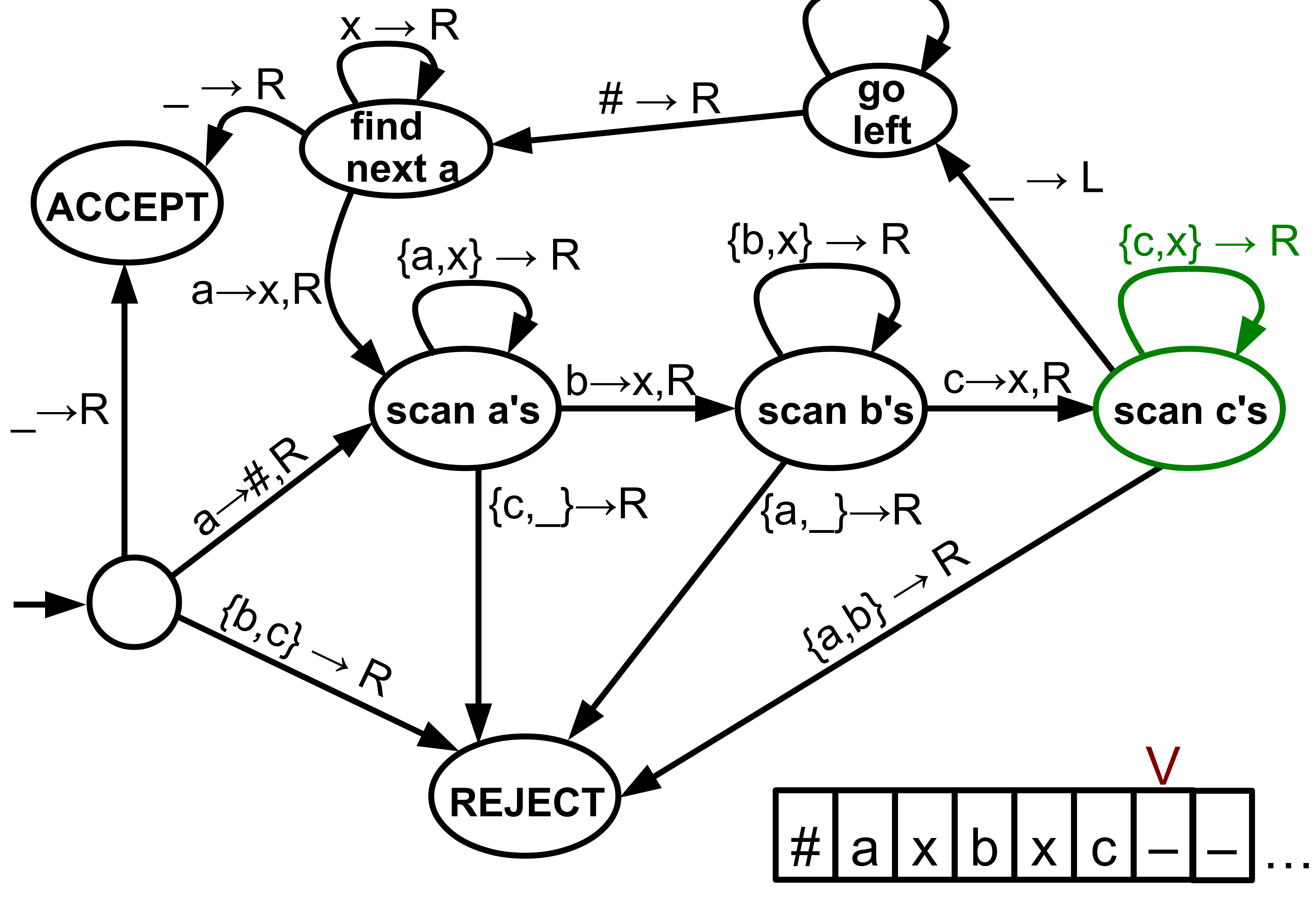


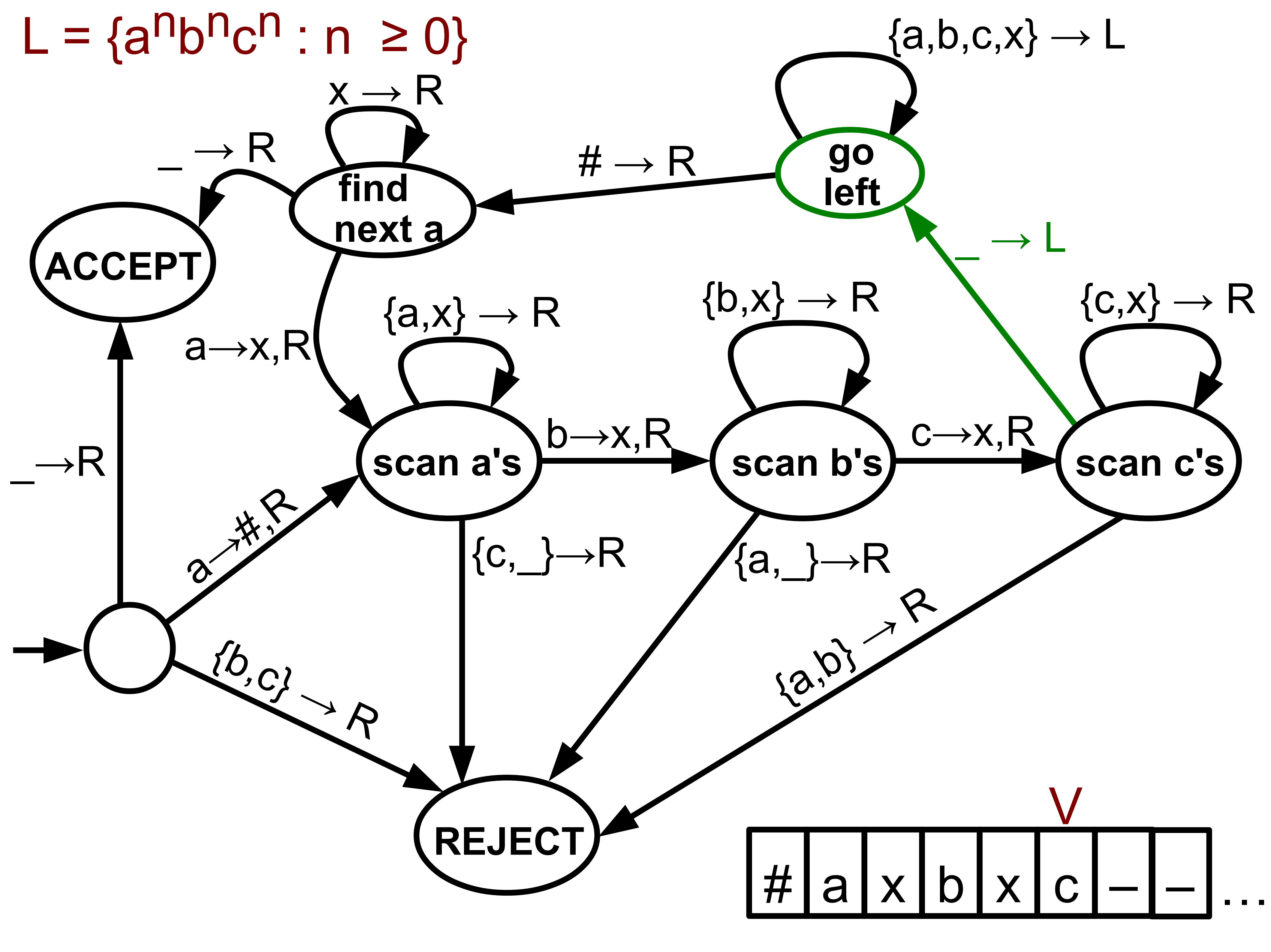
L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b c c _ _ …

$L = \{a^n b^n c^n : n \geq 0\}$



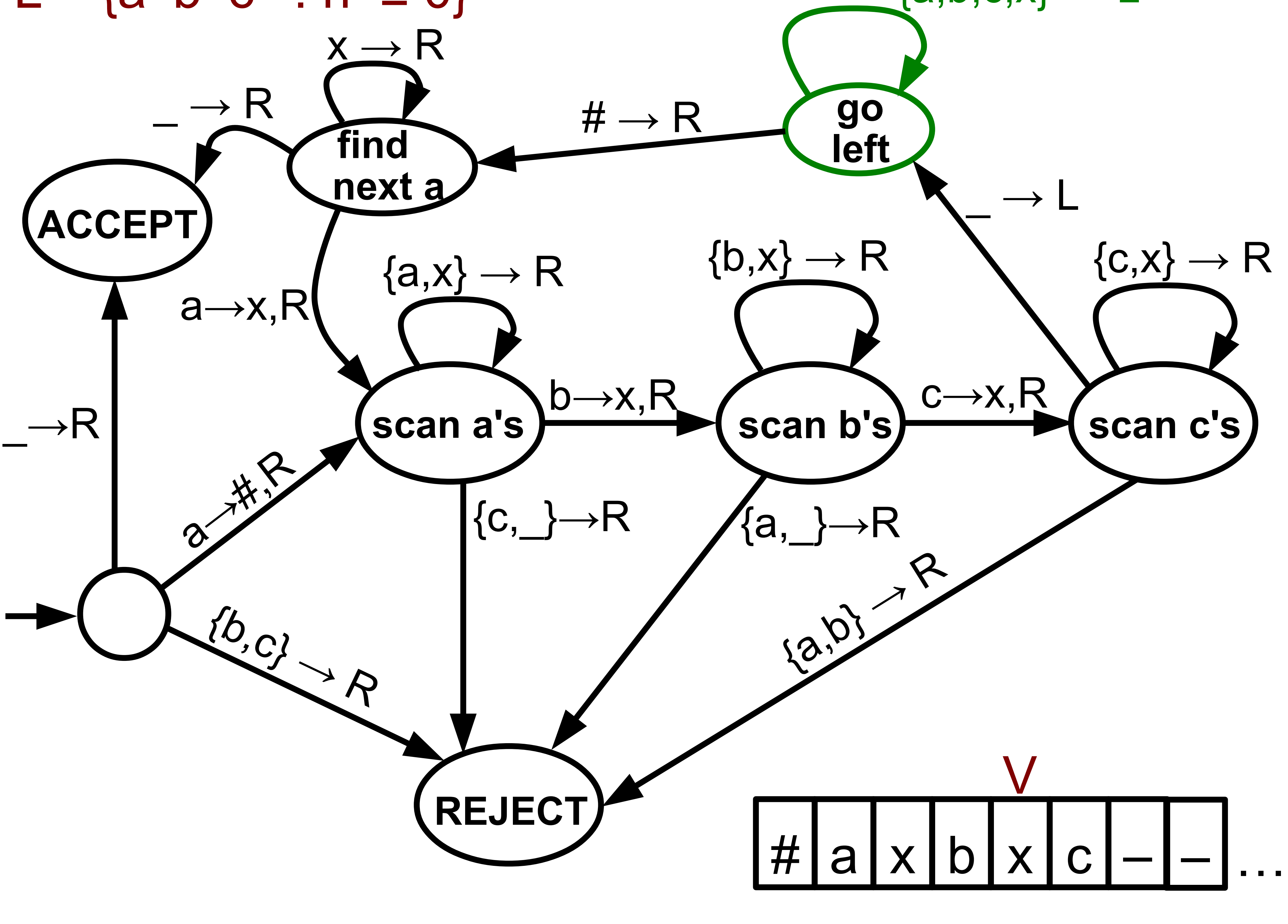
$L = \{a^n b^n c^n : n \geq 0\}$



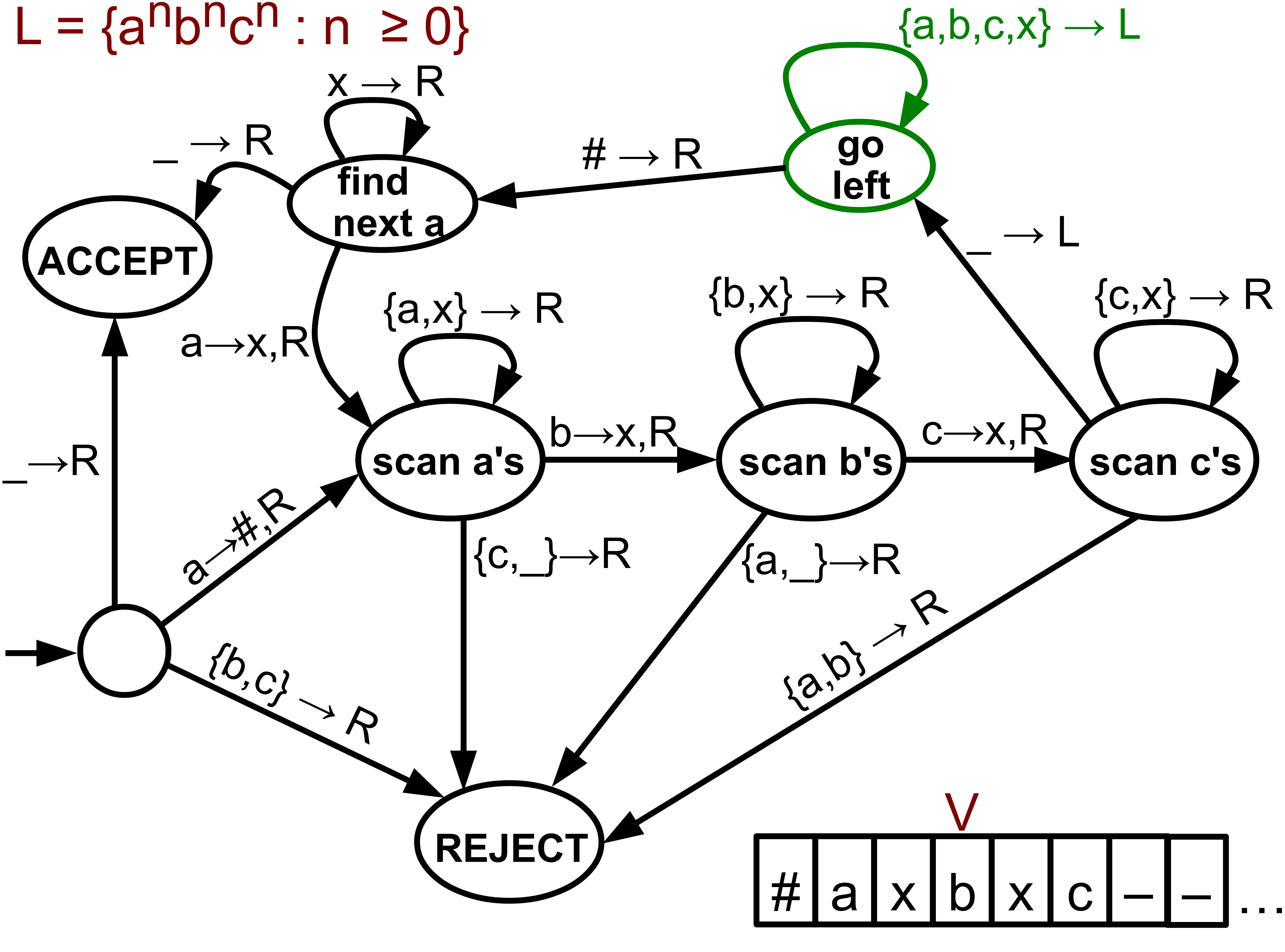


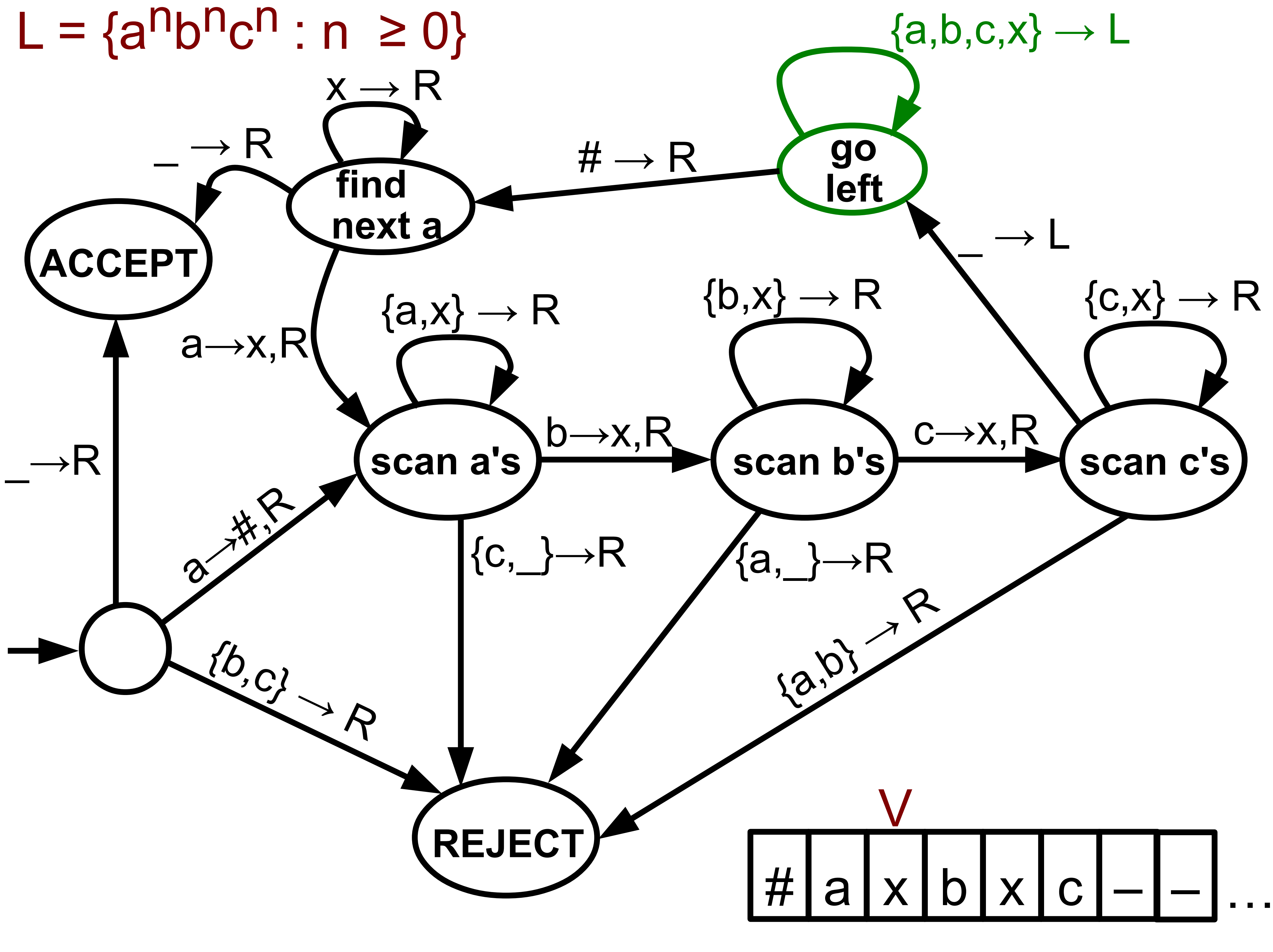
L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b x c _ _ ...

$L = \{a^n b^n c^n : n \geq 0\}$

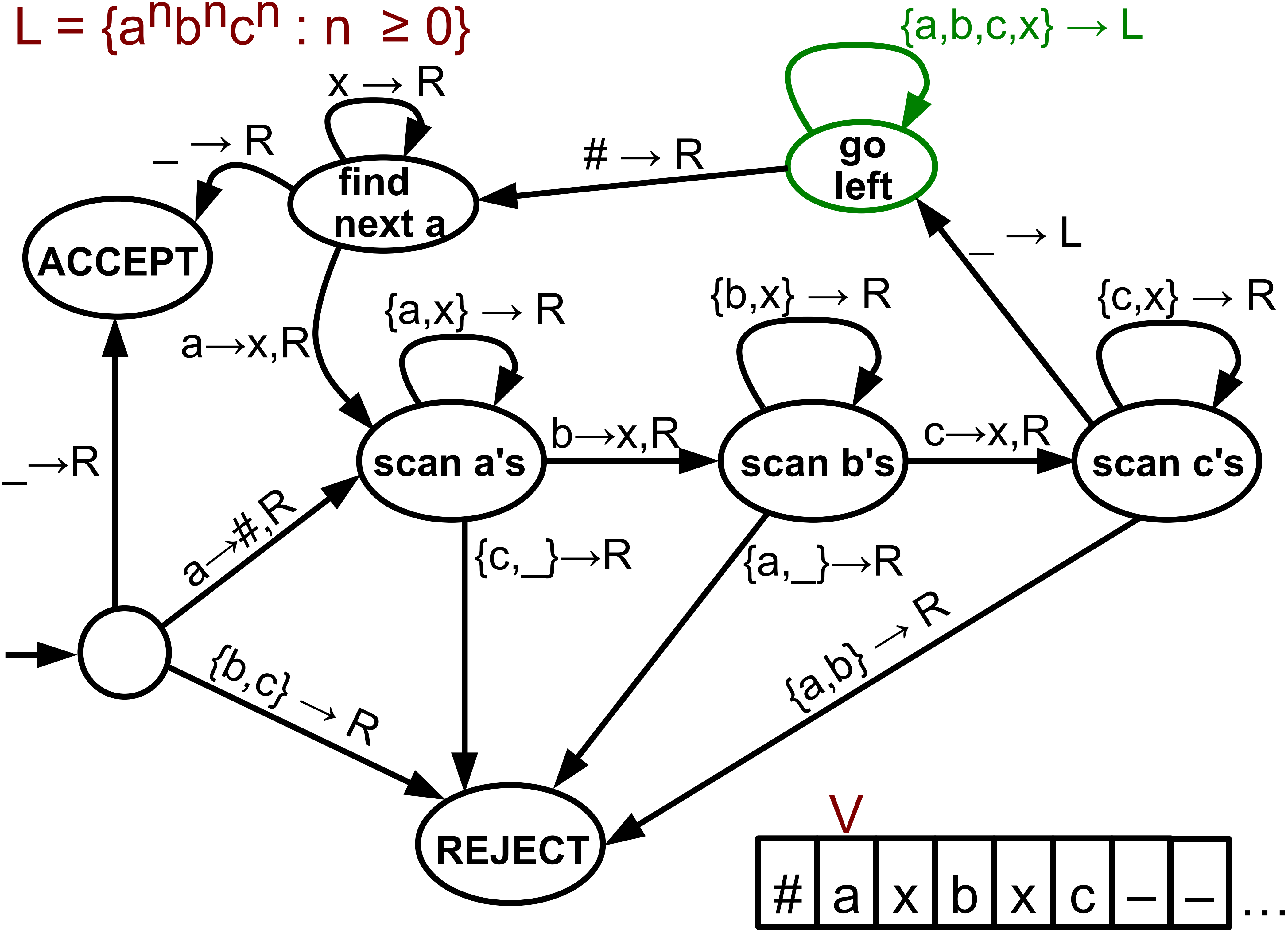


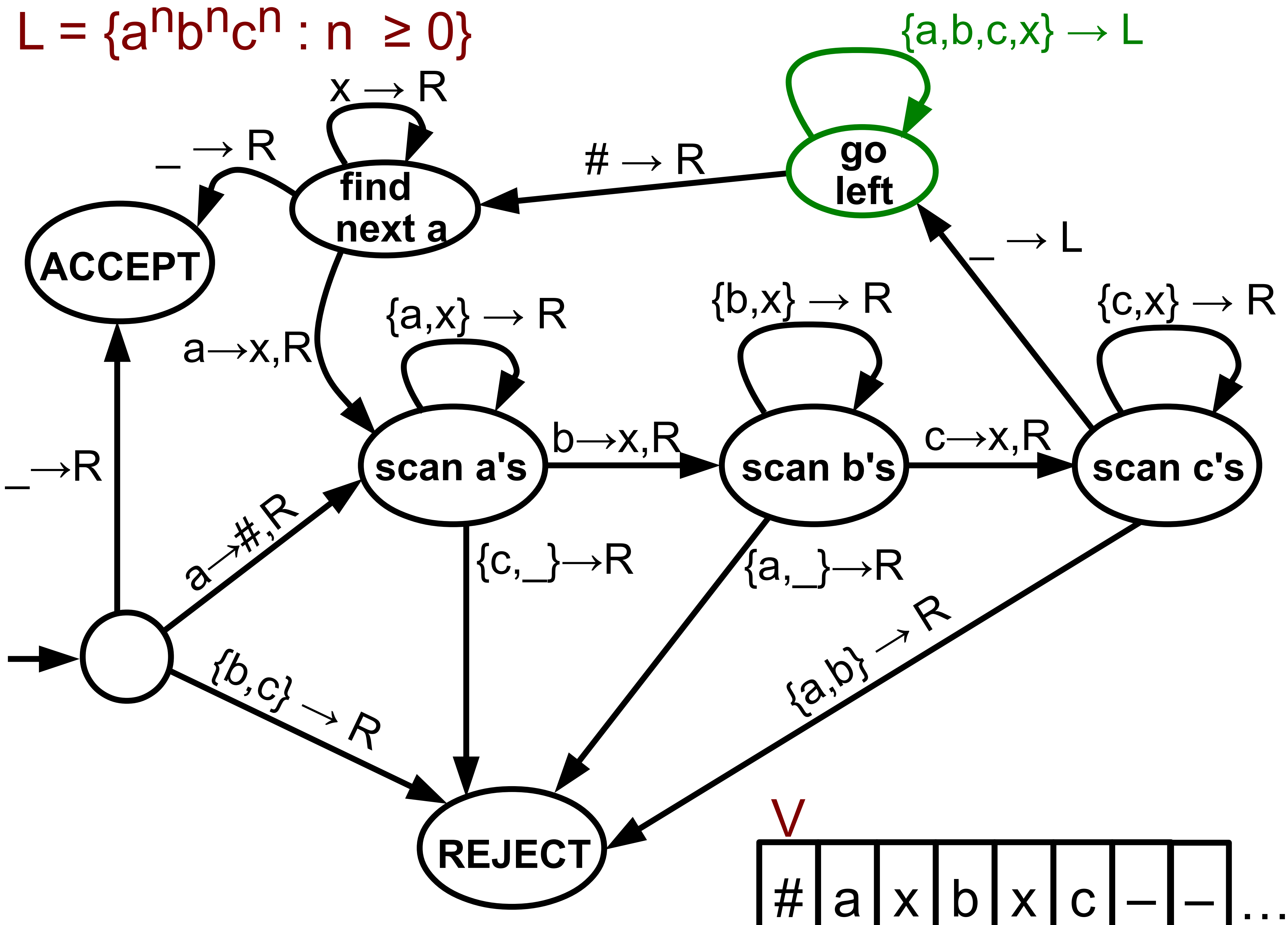
$L = \{a^n b^n c^n : n \geq 0\}$




L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b x c _ _ ...

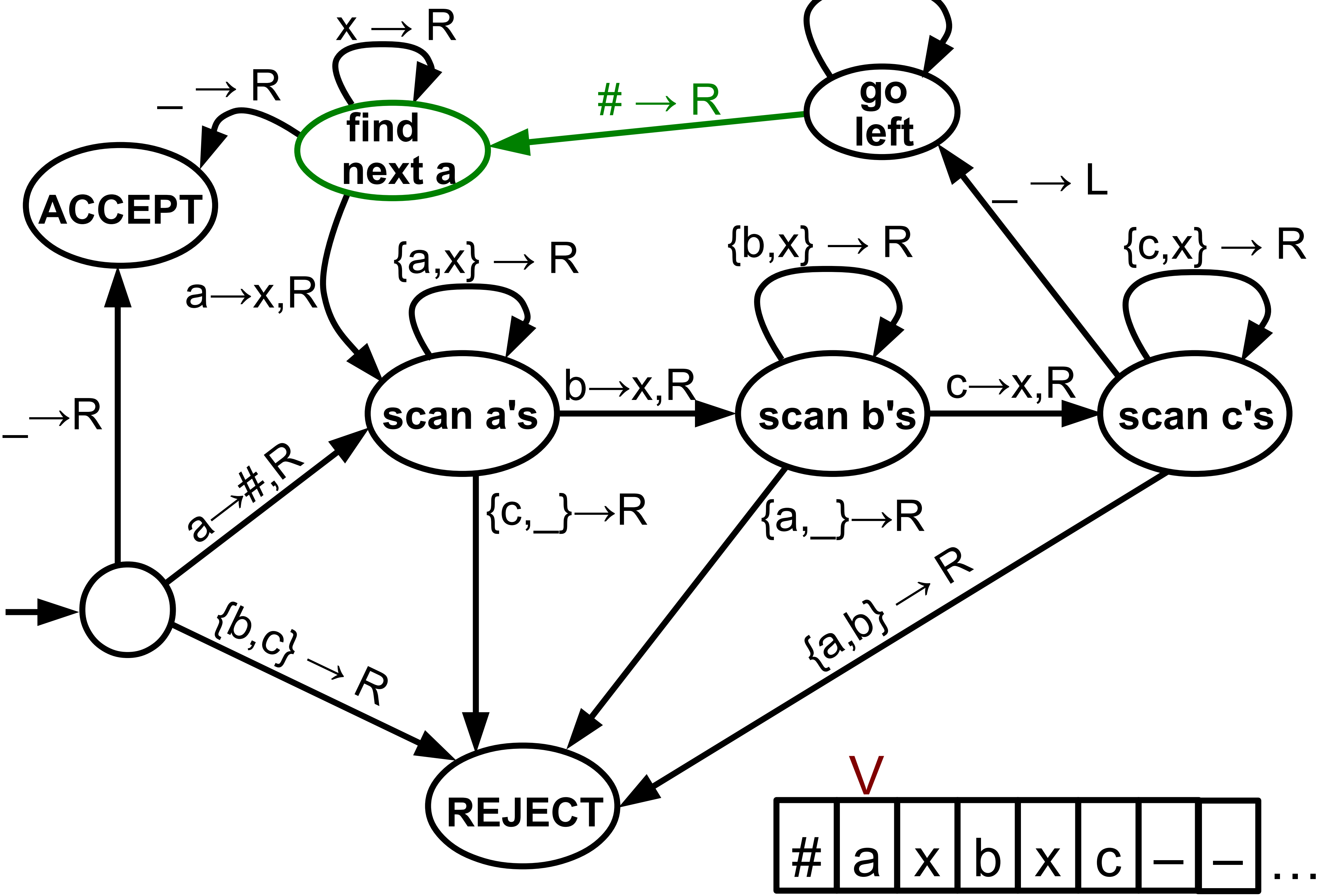
$L = \{a^n b^n c^n : n \geq 0\}$



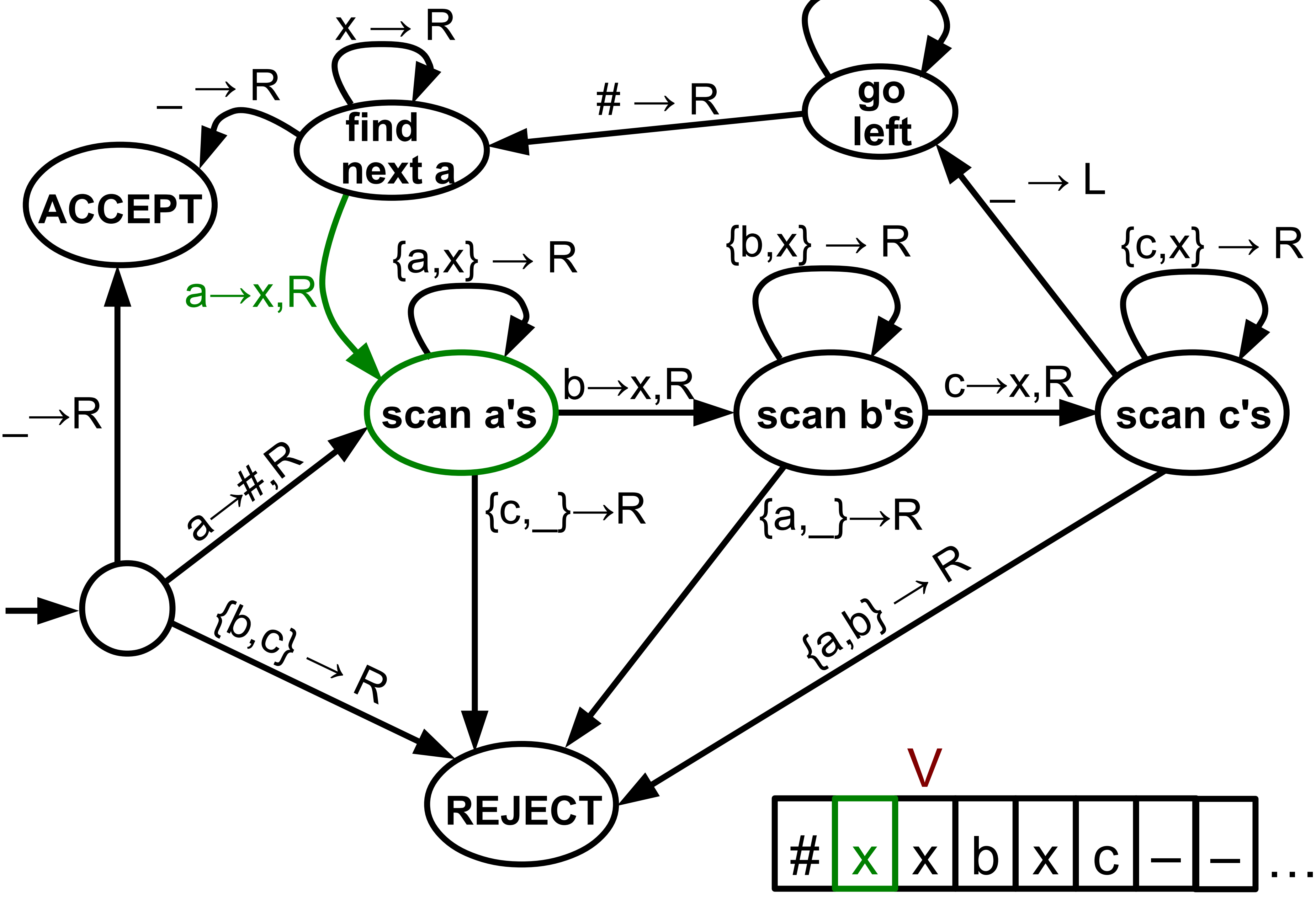


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
a→x,R
{a,x} → R
{b,x} → R
_ → L
{c,x} → R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b x c _ _ ...

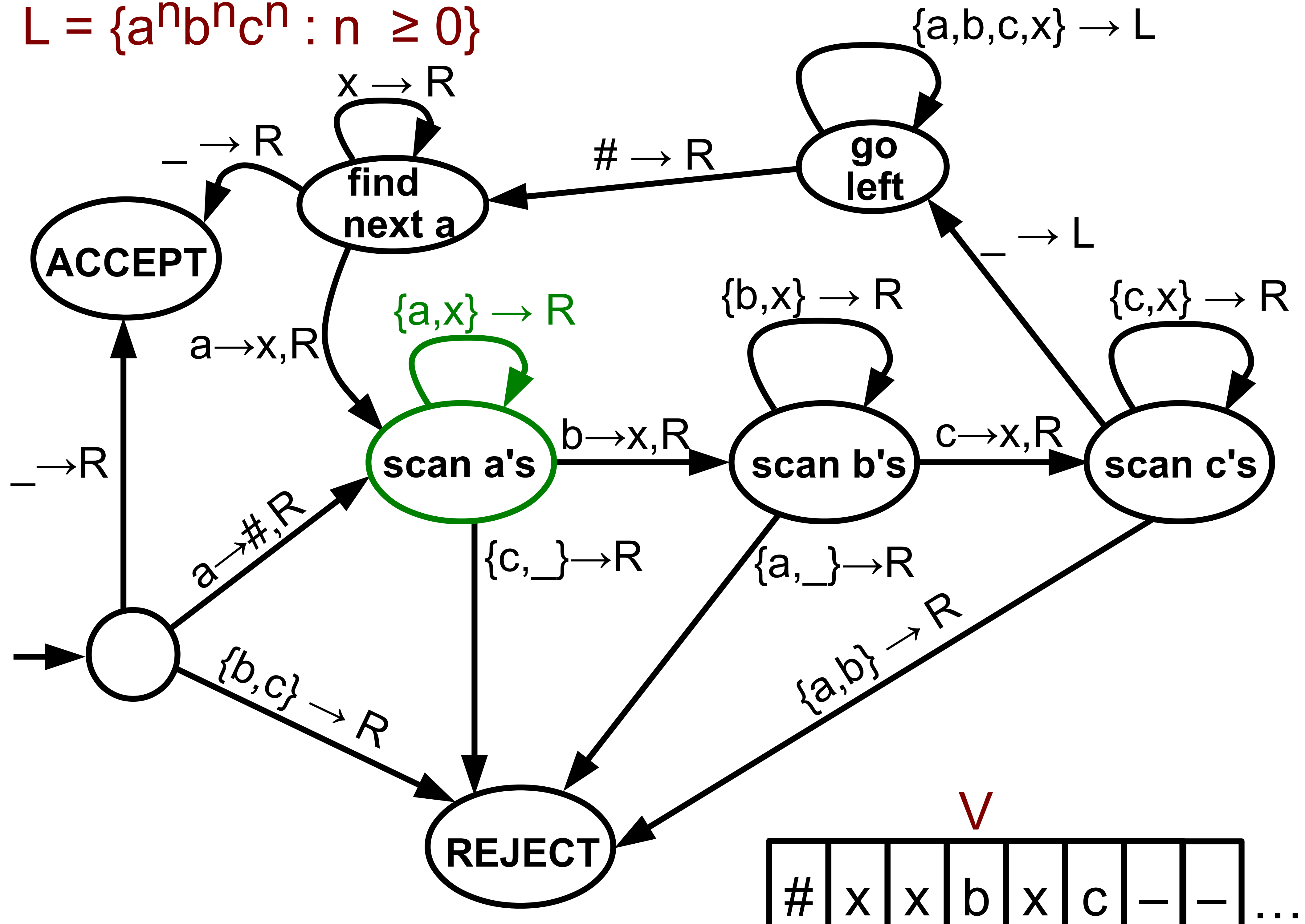
$L = \{a^n b^n c^n : n \geq 0\}$



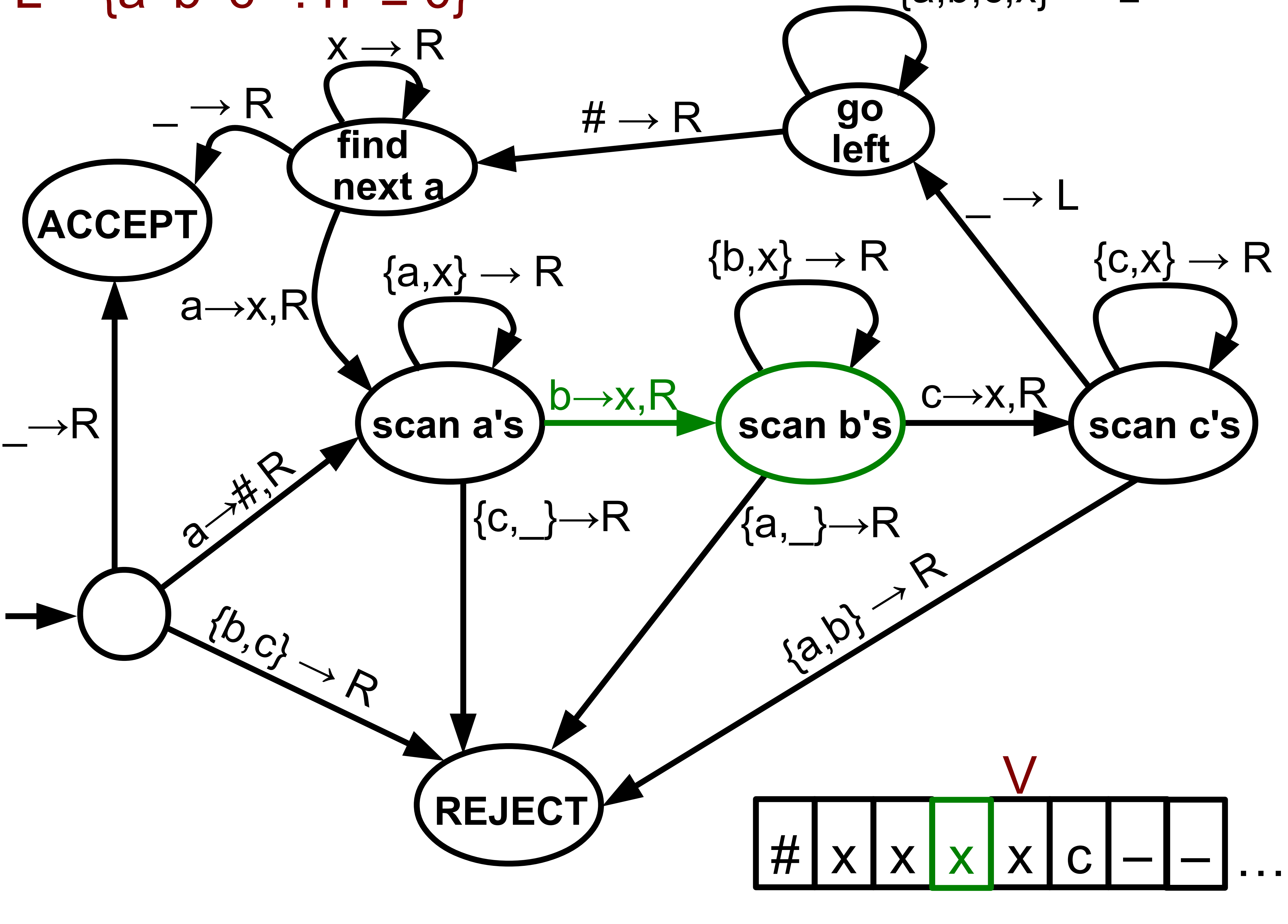
$L = \{a^n b^n c^n : n \geq 0\}$

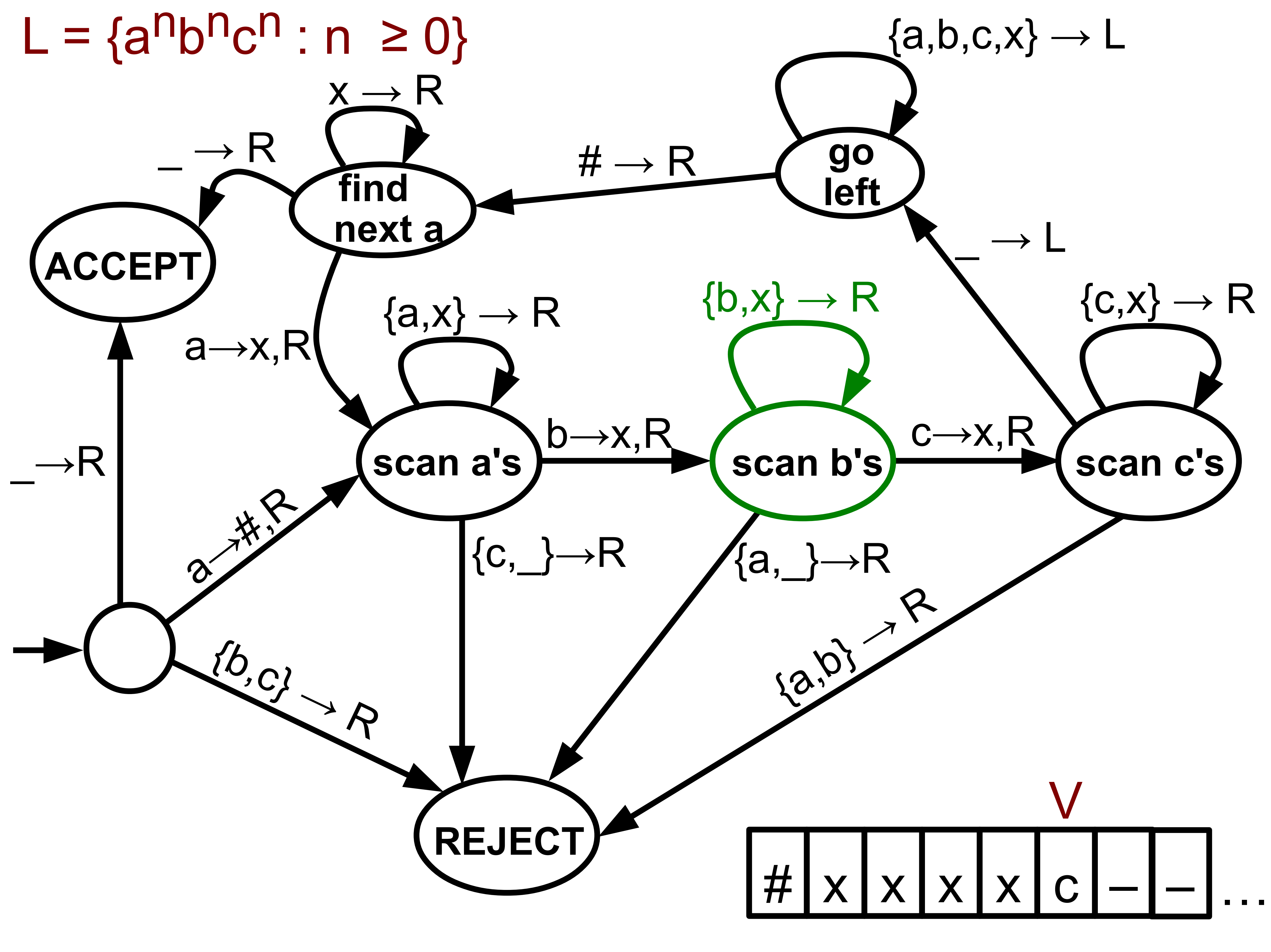


$L = \{a^n b^n c^n : n \geq 0\}$



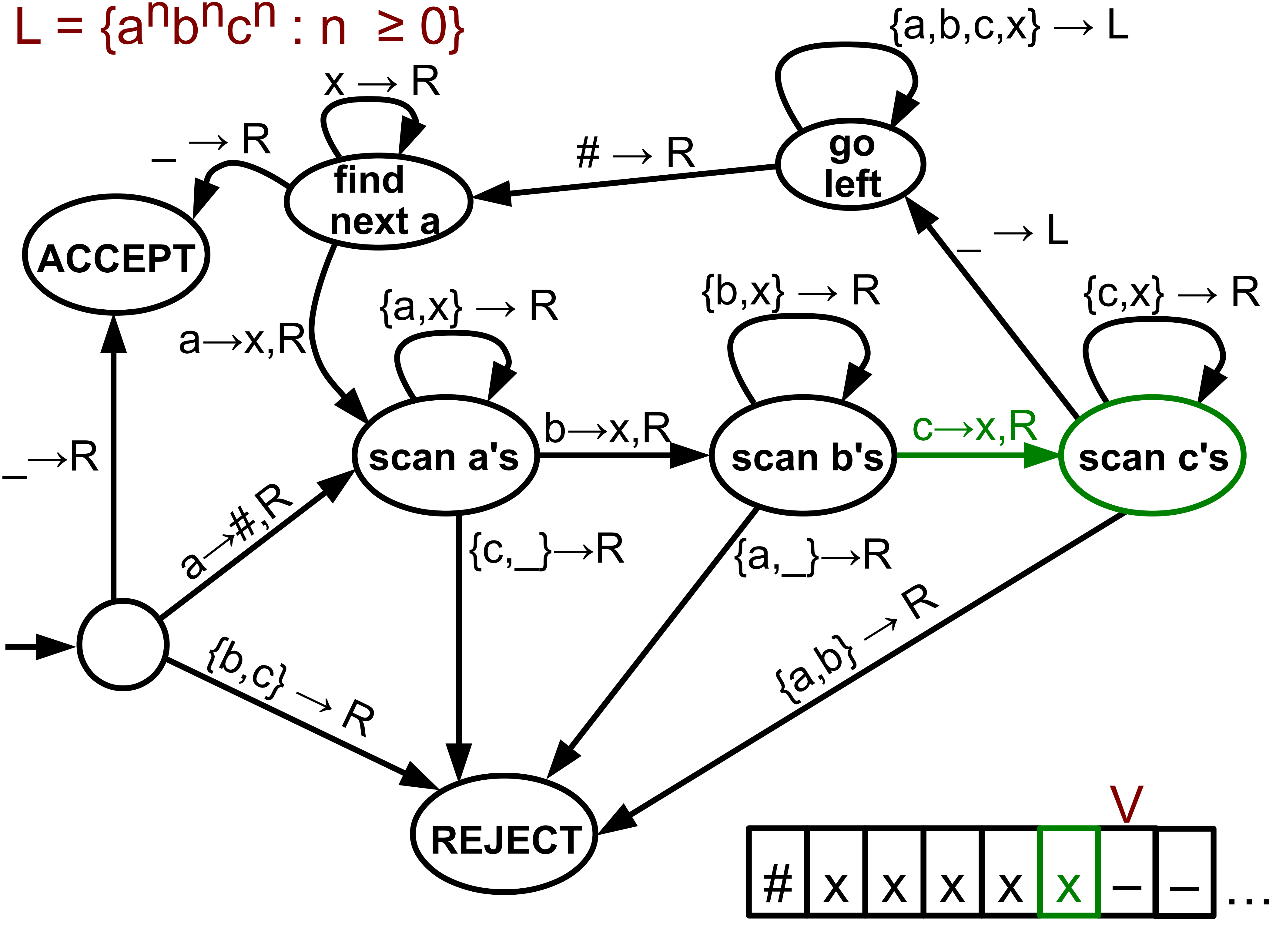
$L = \{a^n b^n c^n : n \geq 0\}$



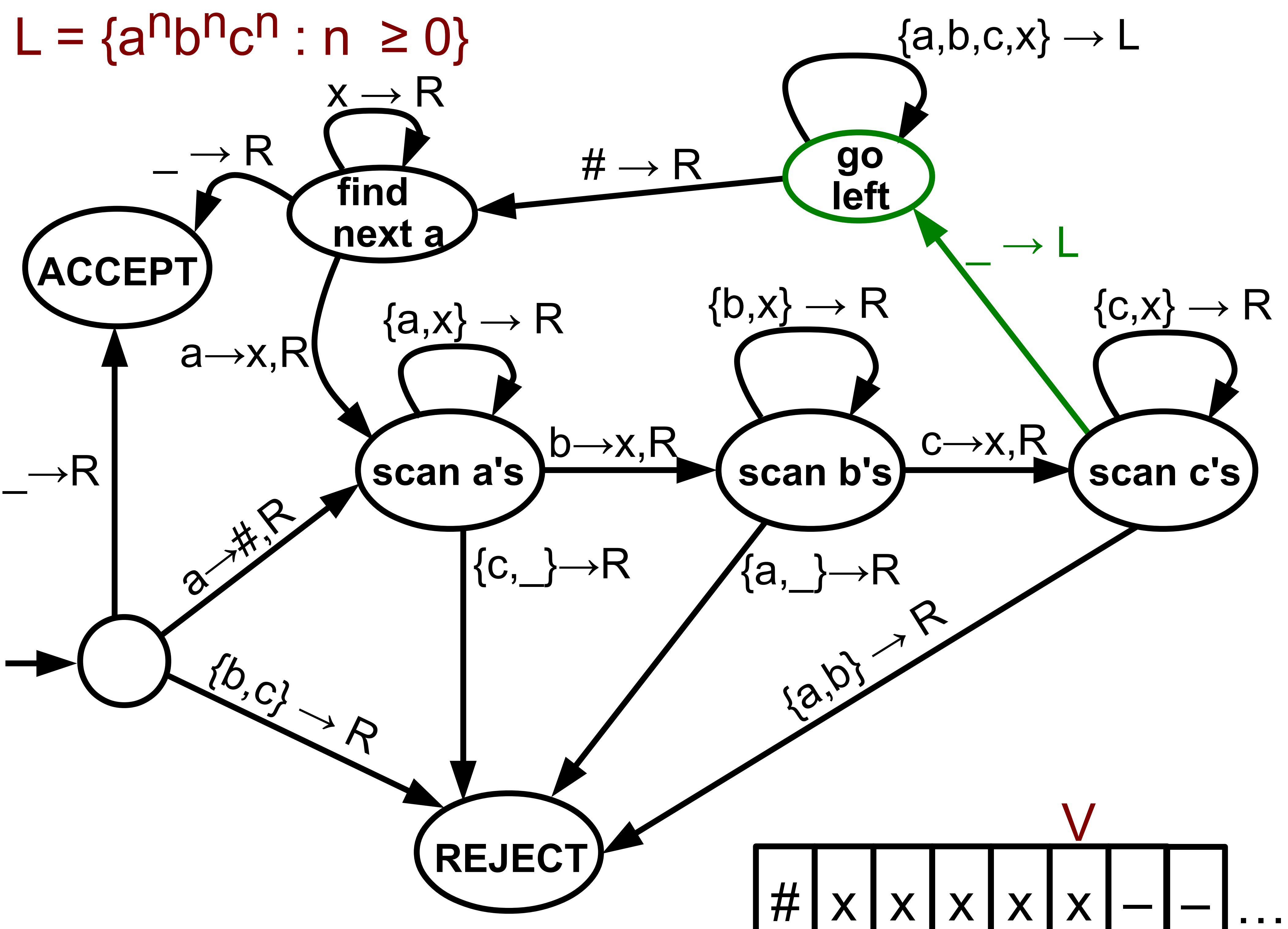


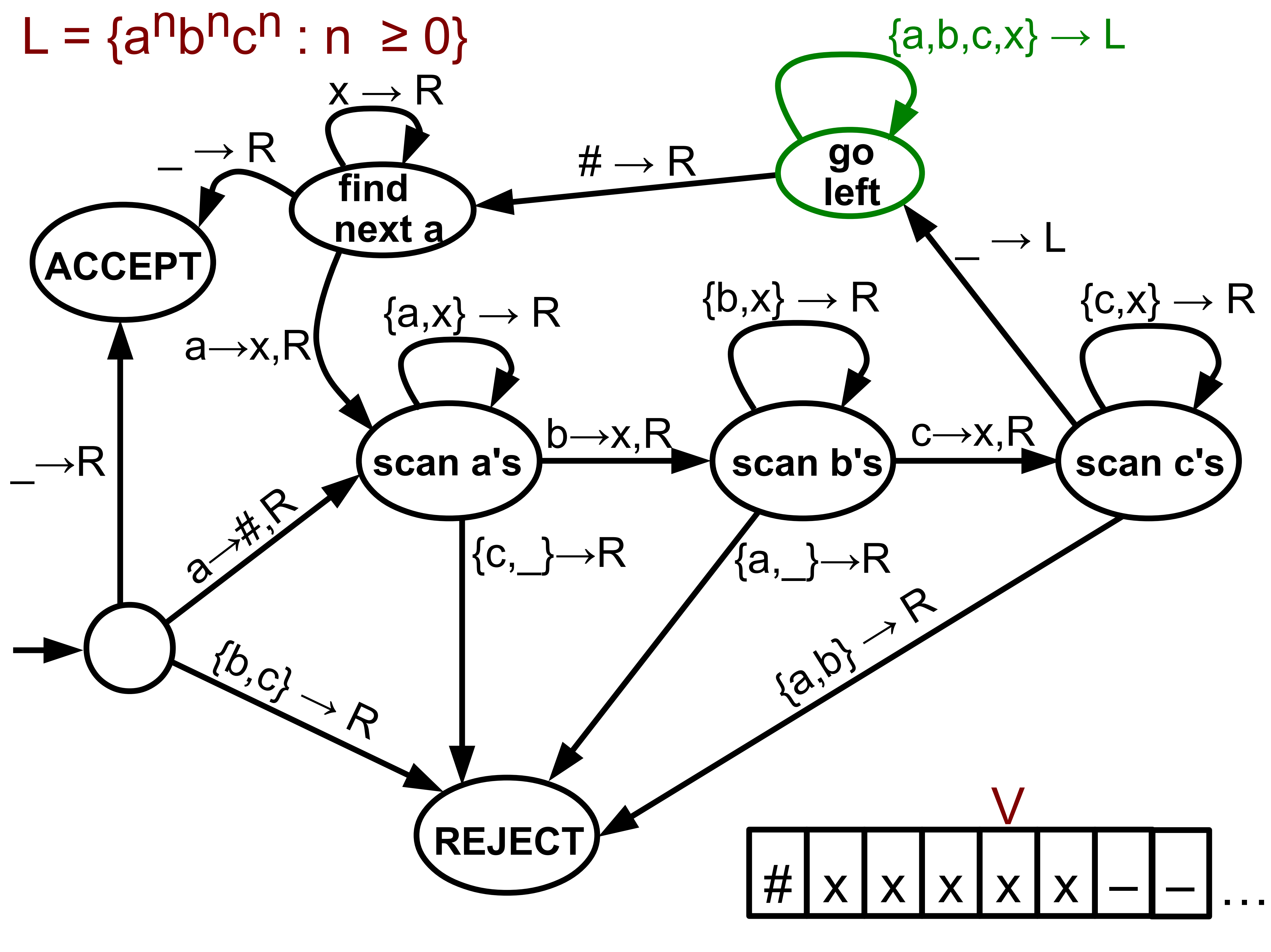
L = {aⁿbⁿcⁿ : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x c _ _ ...

$L = \{a^n b^n c^n : n \geq 0\}$

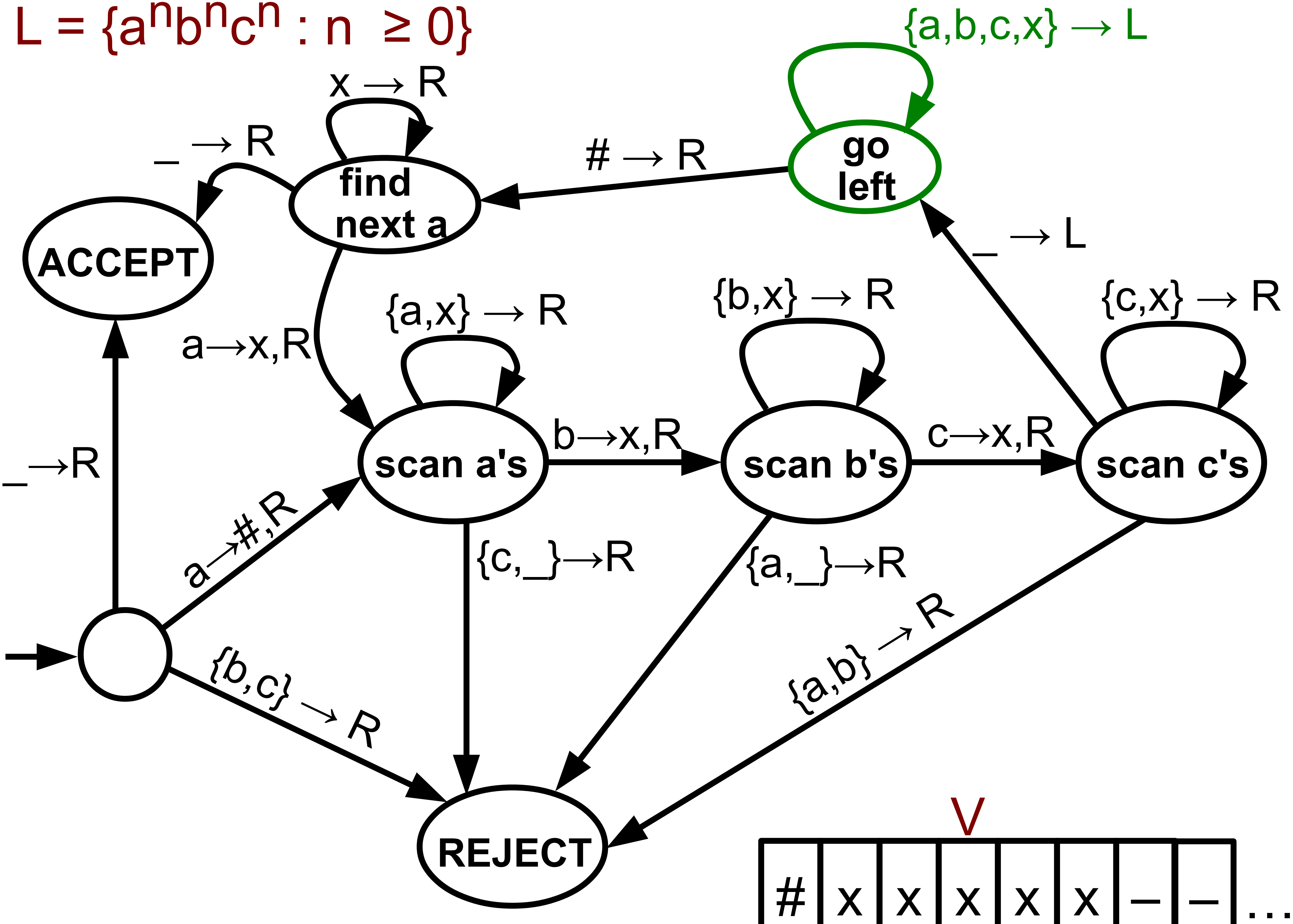


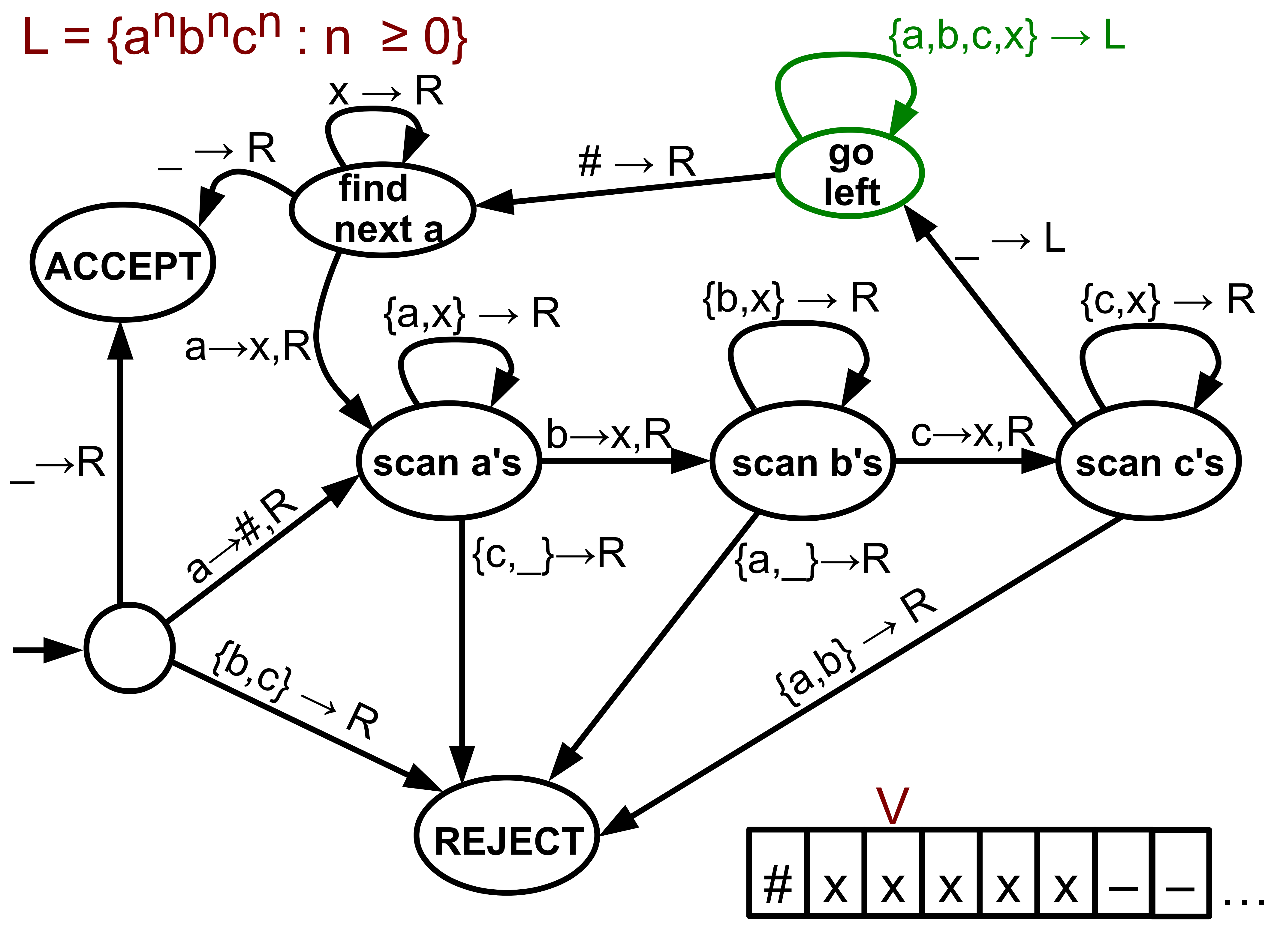
$L = \{a^n b^n c^n : n \geq 0\}$

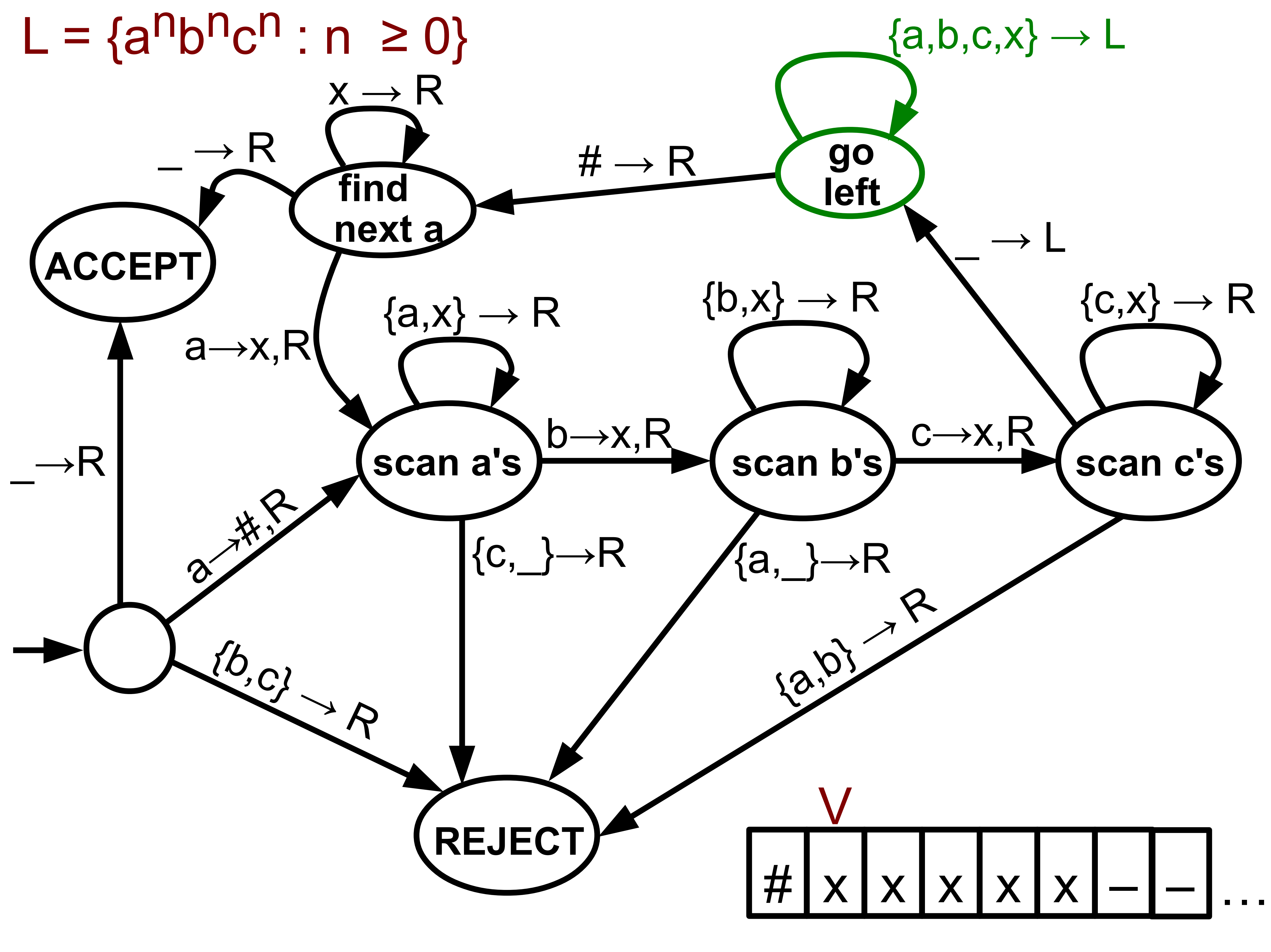



L = {aⁿbⁿcⁿ : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
scan a's
b→x,R
scan b's
c→x,R
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x x _ _ ...

$L = \{a^n b^n c^n : n \geq 0\}$

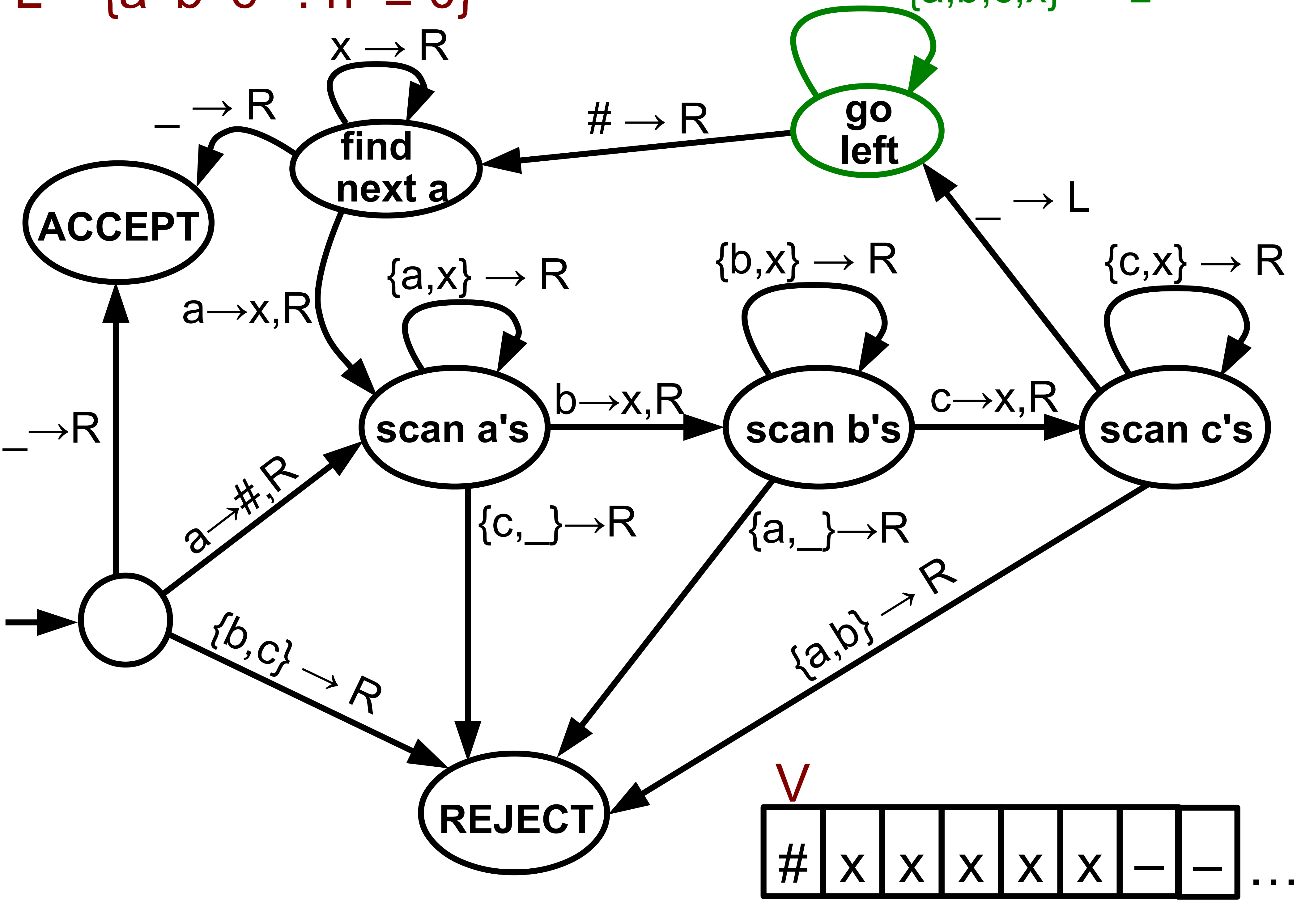



L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x x _ _ …

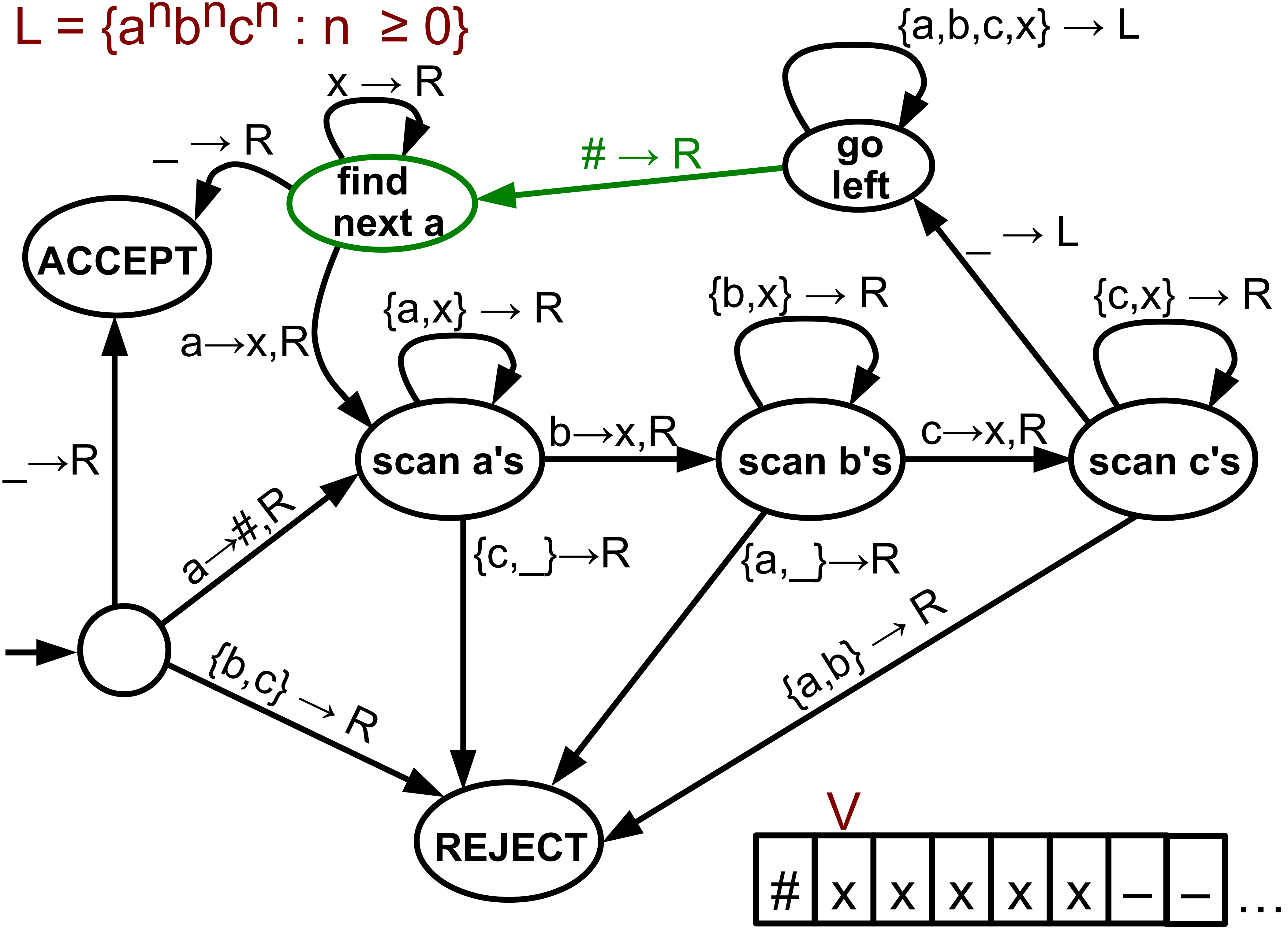


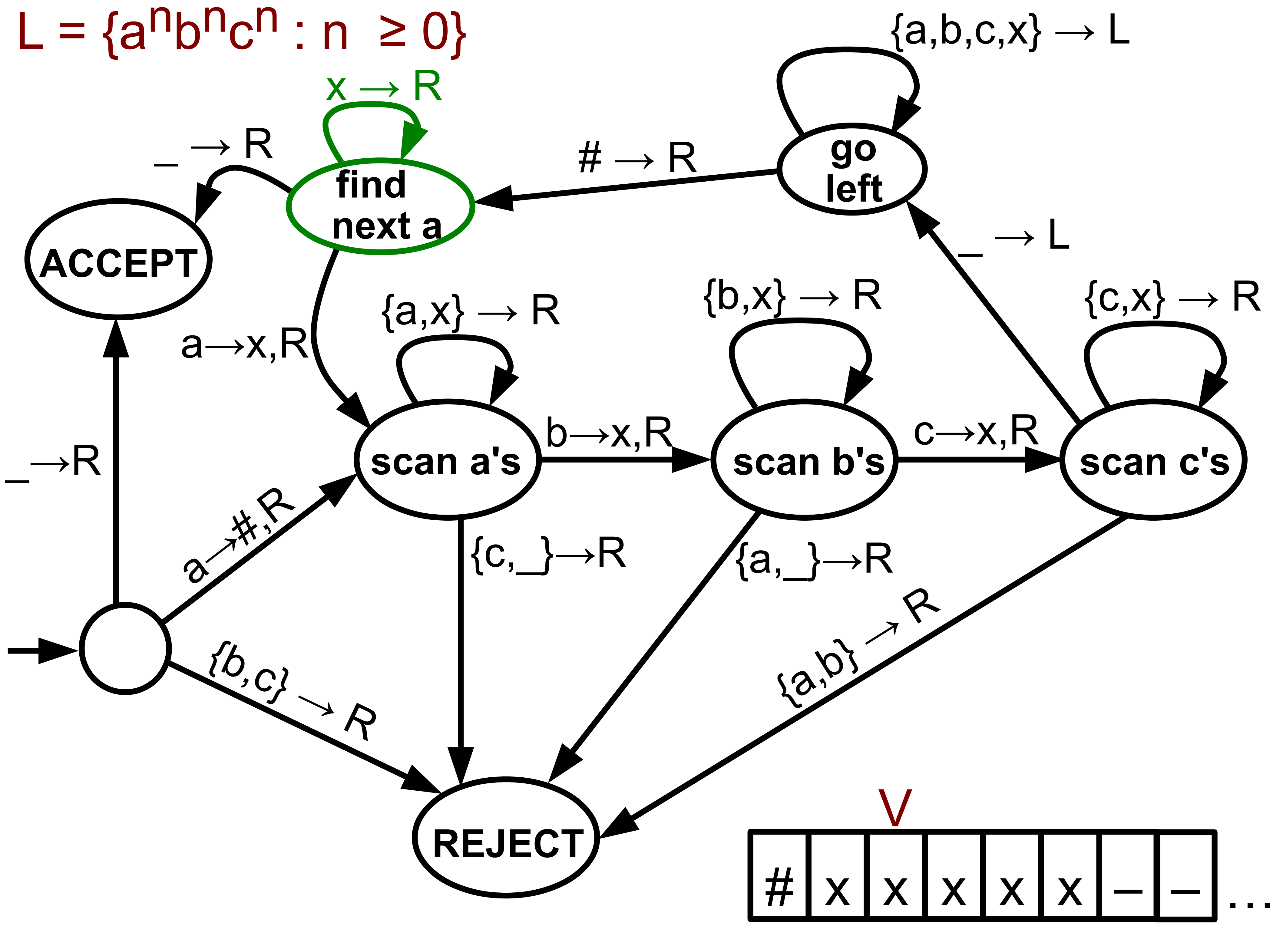
L = {aⁿbⁿcⁿ : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{a,x} → R
{c,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x x _ _ ...

$L = \{a^n b^n c^n : n \geq 0\}$

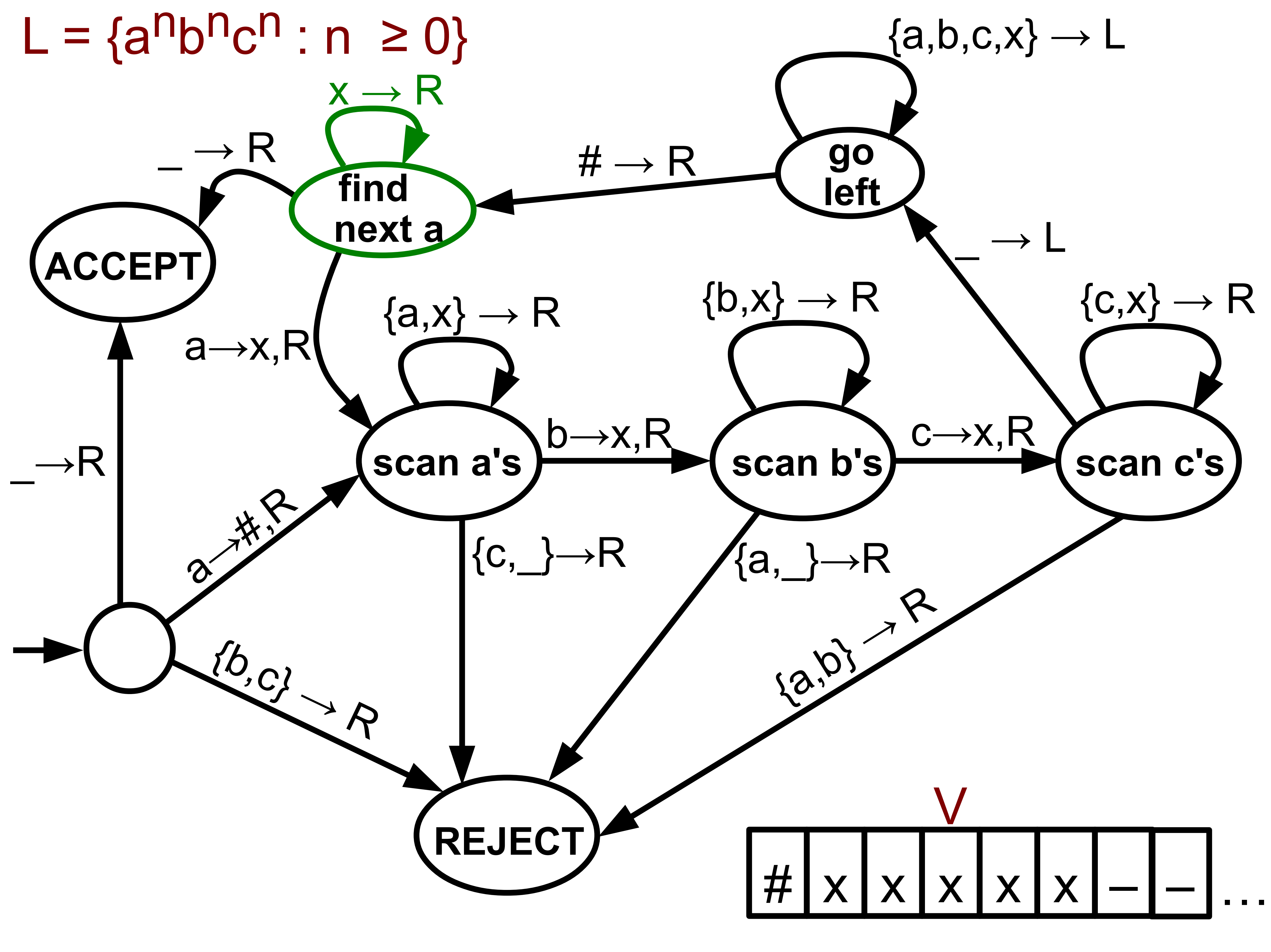


$L = \{a^n b^n c^n : n \geq 0\}$



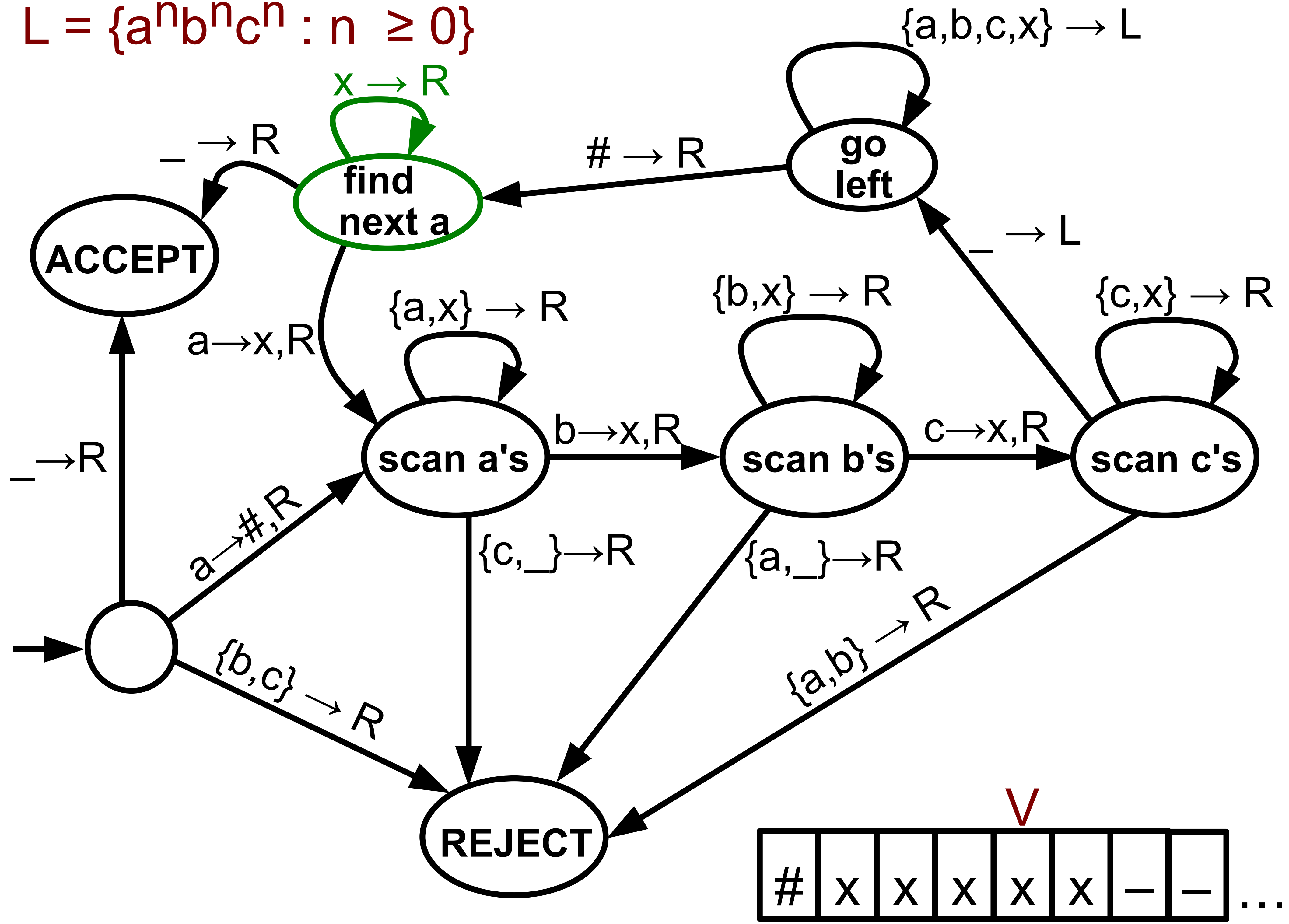


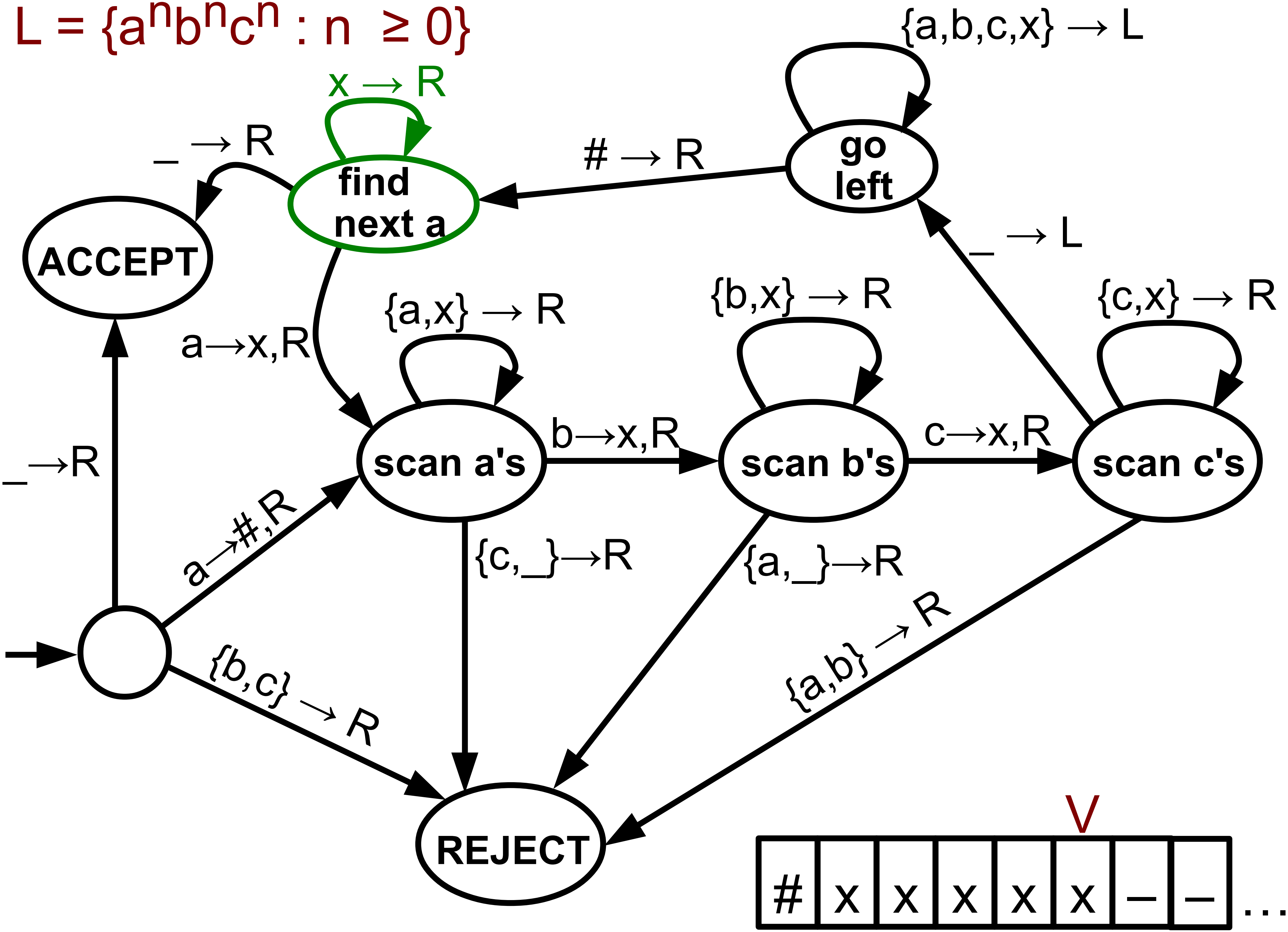
L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
find next a
go left
→ R
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
scan a's
b→x,R
scan b's
c→x,R
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x x _ _ ...



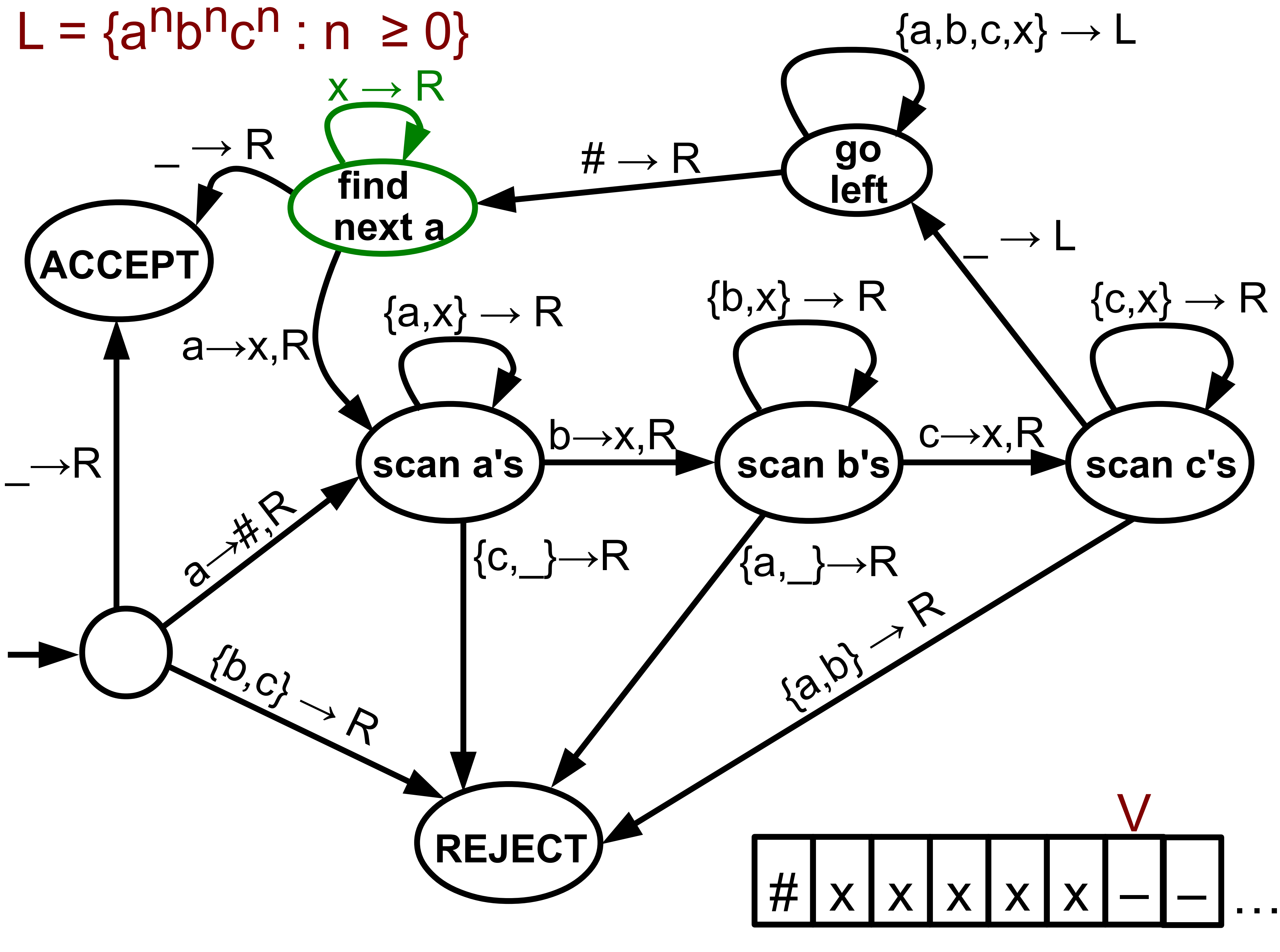
L = {a^n b^n c^n : n ≥ 0}
x → R
_ → R
find next a
→ R
go left
{a,b,c,x} → L
ACCEPT
a→x,R
{a,x} → R
_ → L
{b,x} → R
{c,x} → R
_→R
scan a's
b→x,R
scan b's
c→x,R
scan c's
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x x _ _ ...

$L = \{a^n b^n c^n : n \geq 0\}$

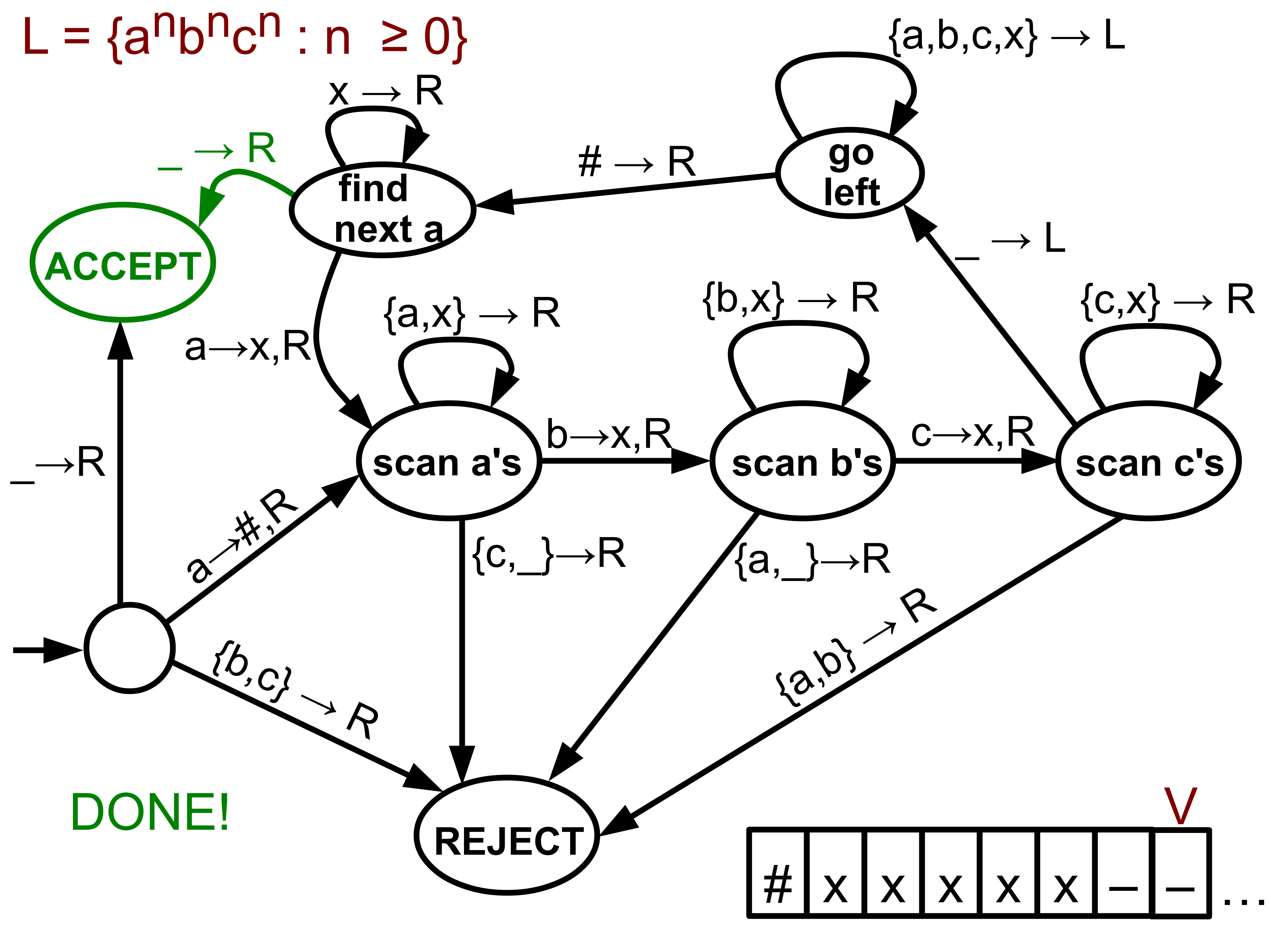


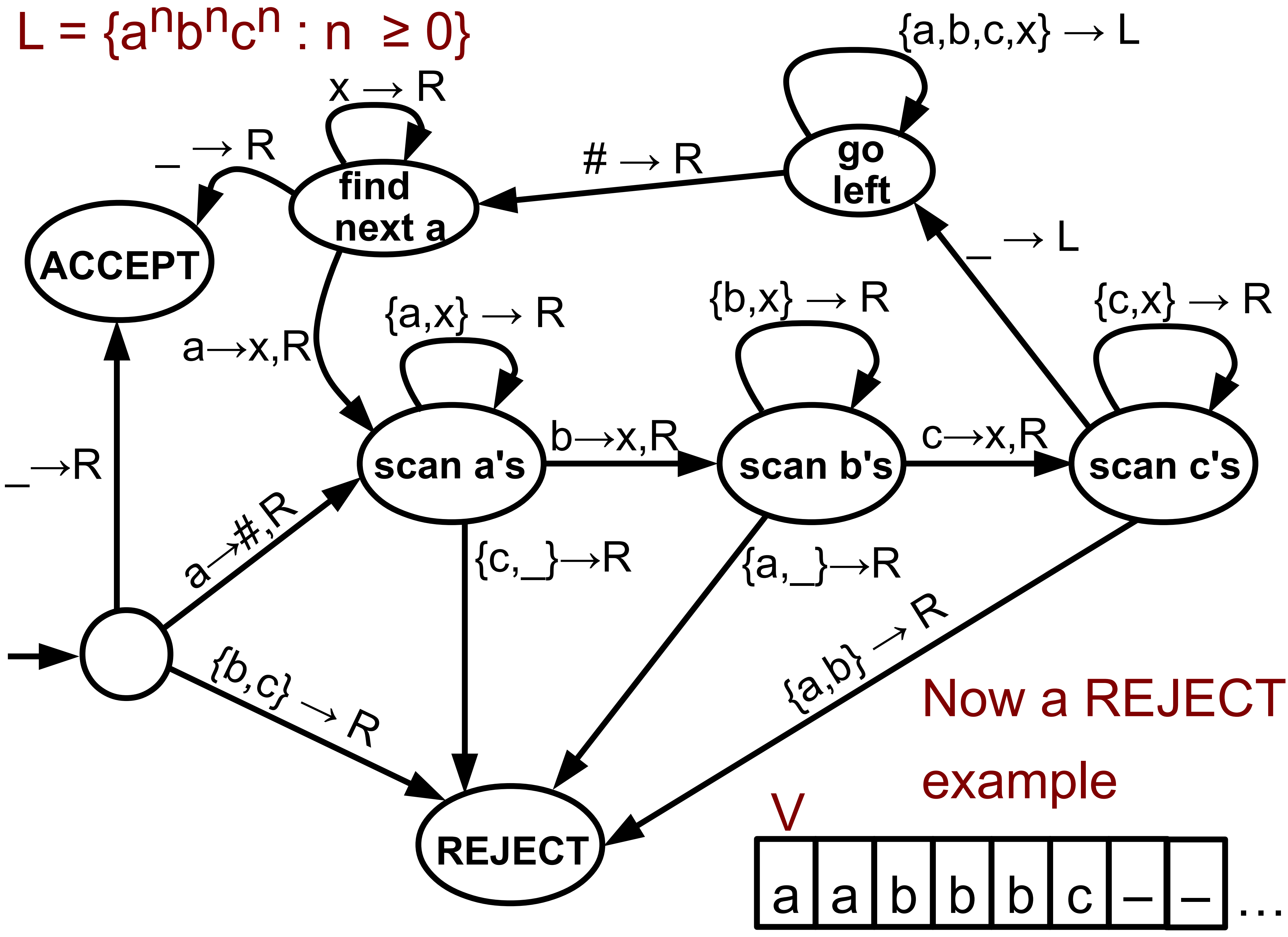


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x x _ _ ...

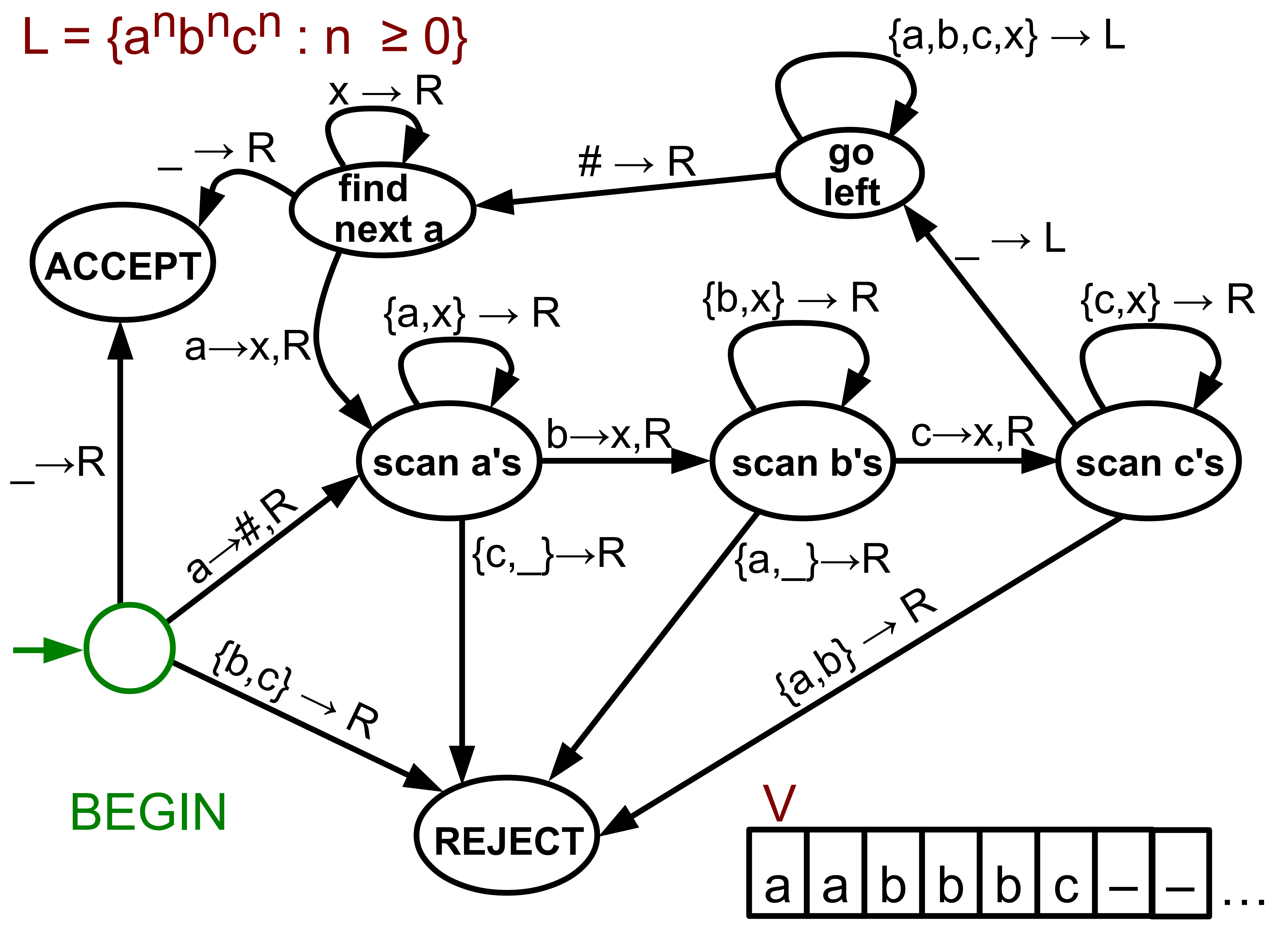


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x x x _ _ ...


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
DONE!
REJECT
V
x x x x x _ _ …

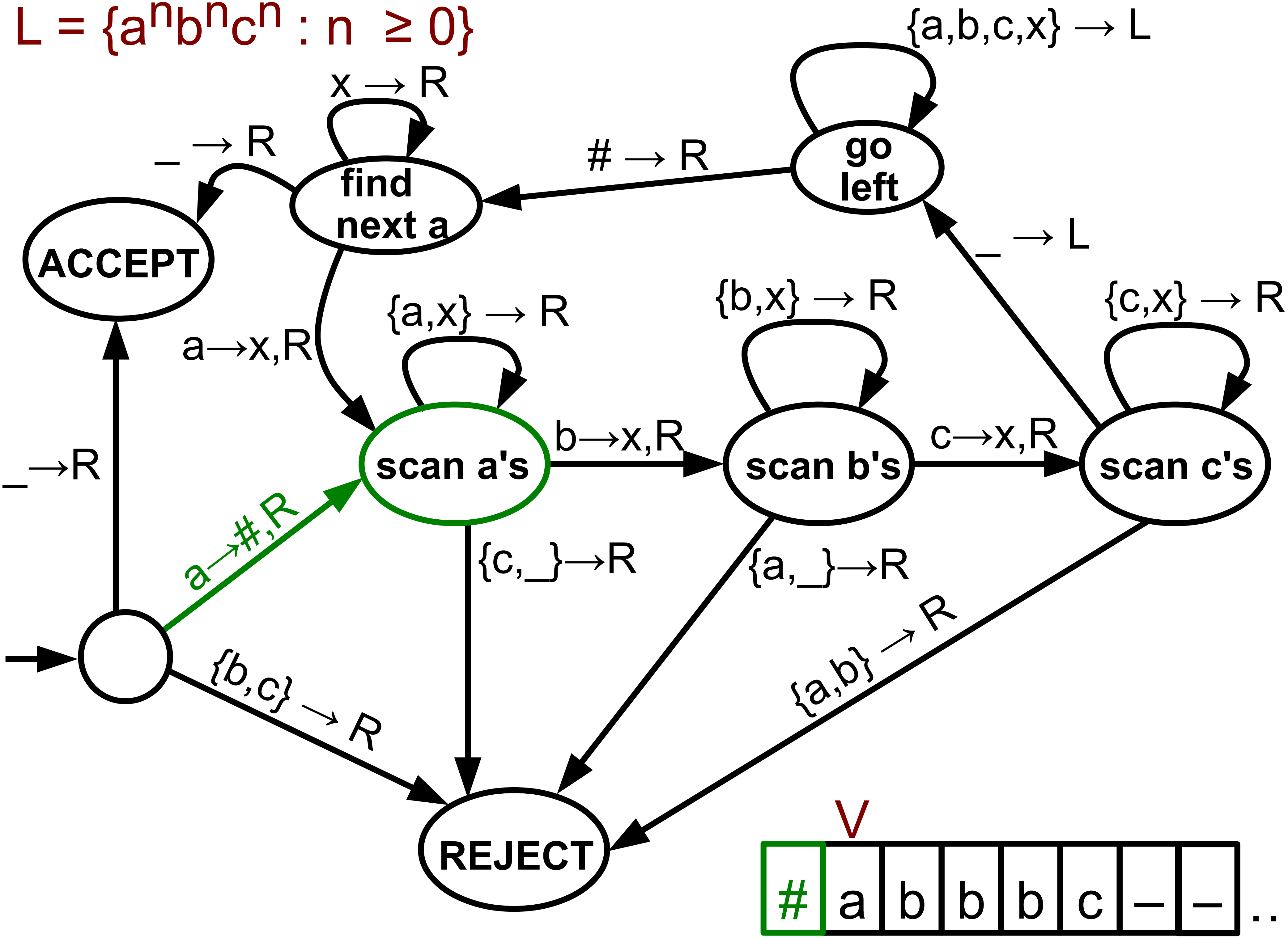


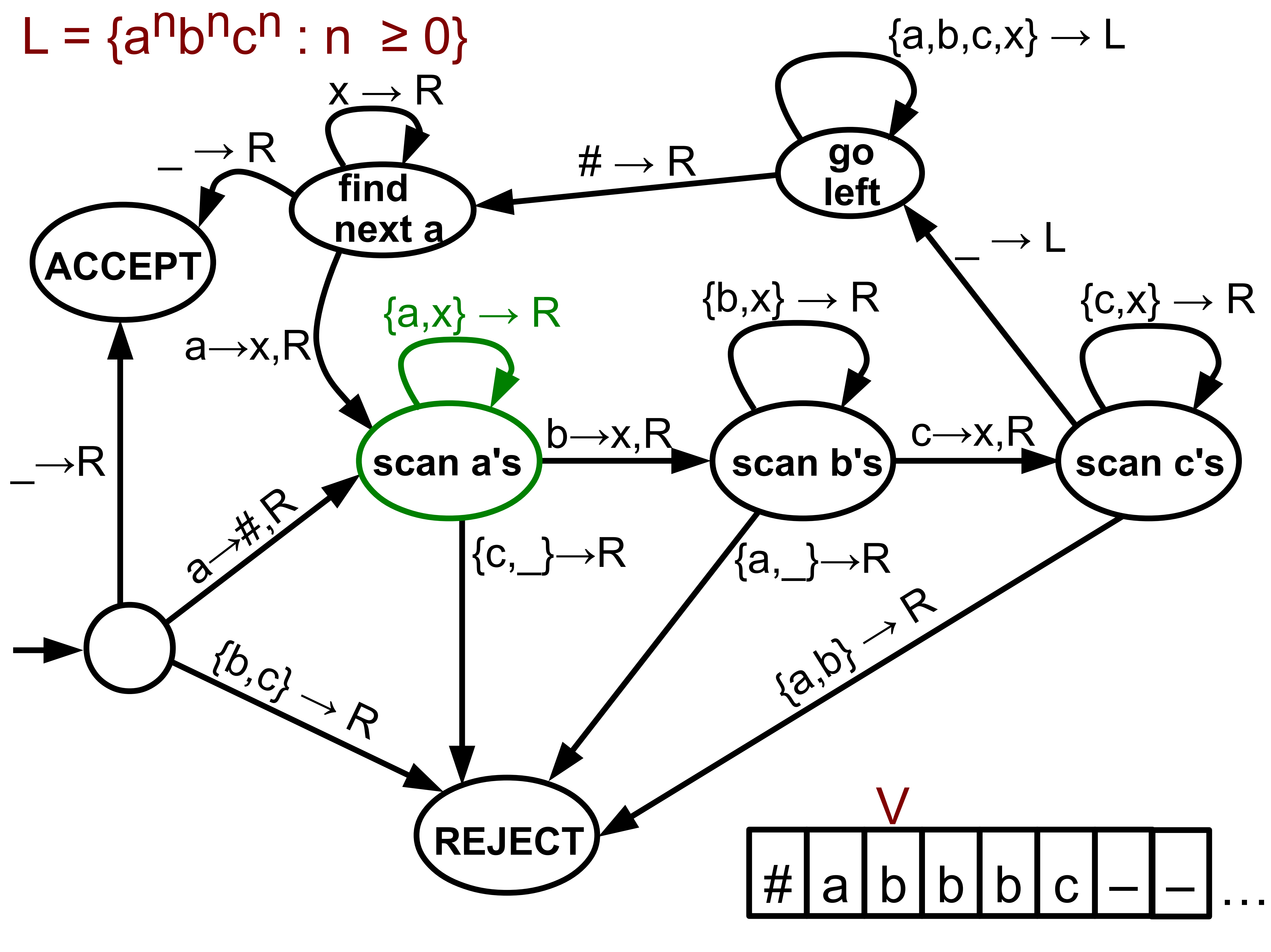
L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
Now a REJECT example
V
a a b b b c _ _ ...



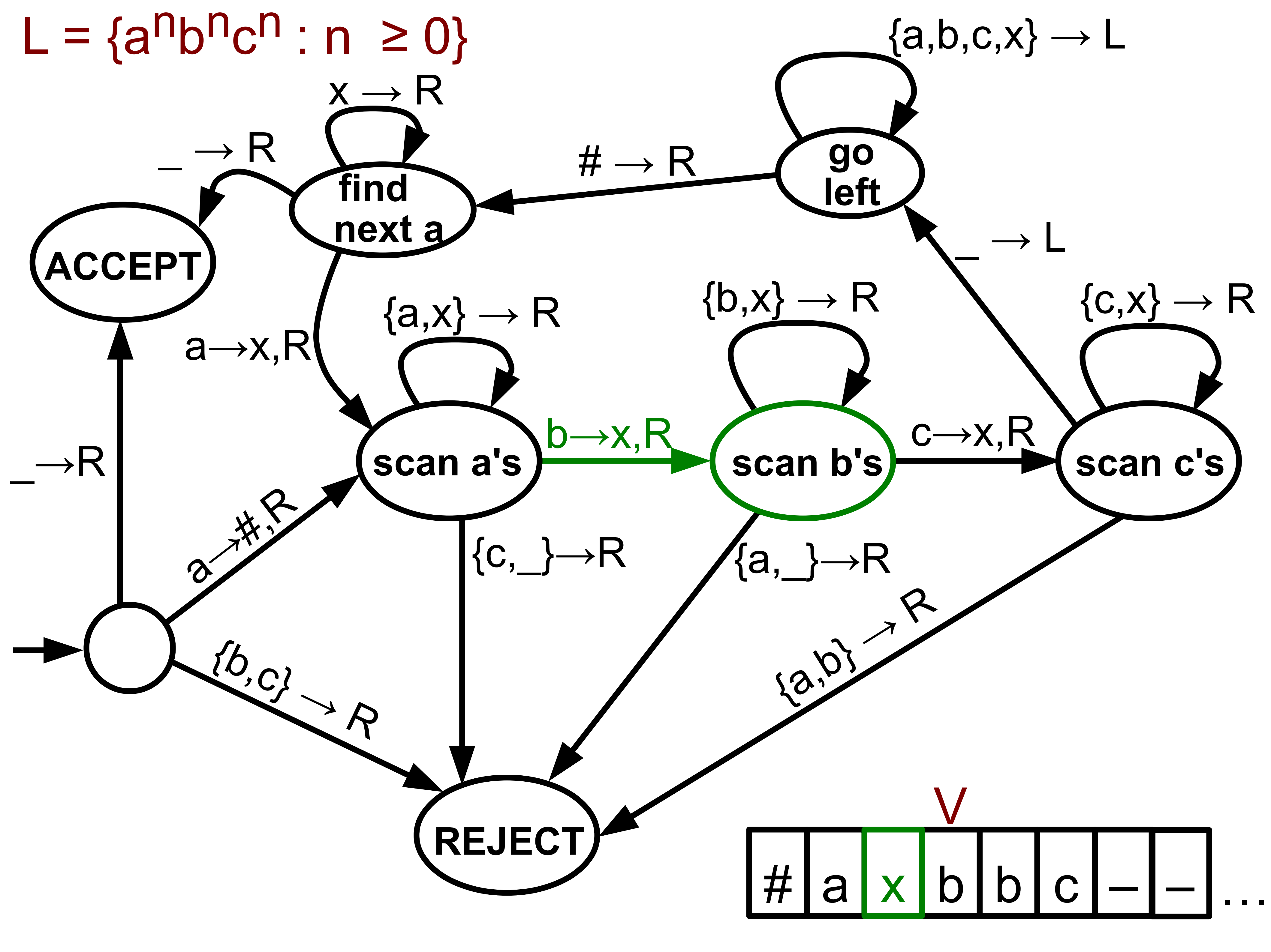
L = {aⁿbⁿcⁿ : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
BEGIN
REJECT
V
a a b b b c _ _ …

$L = \{a^n b^n c^n : n \geq 0\}$



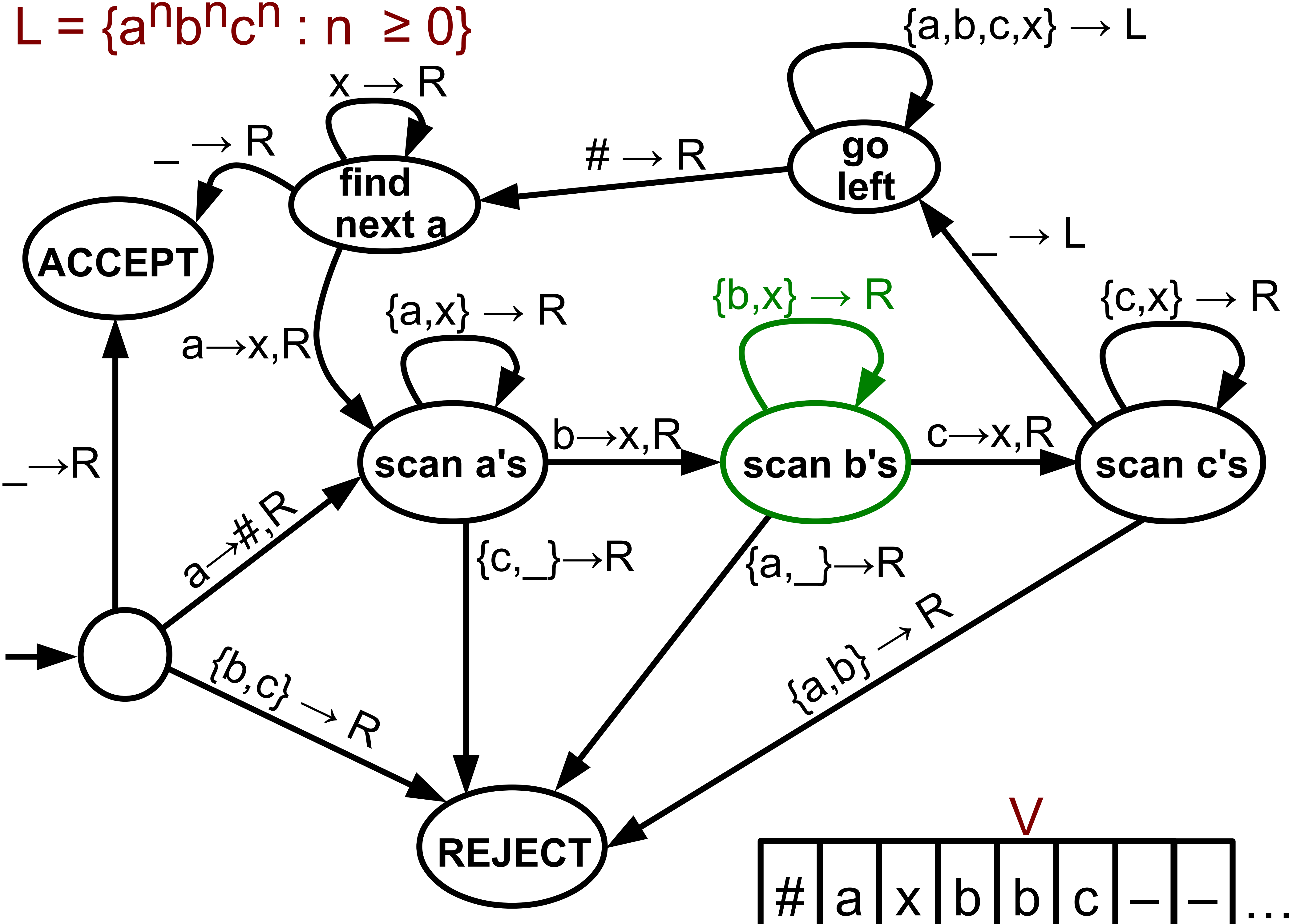


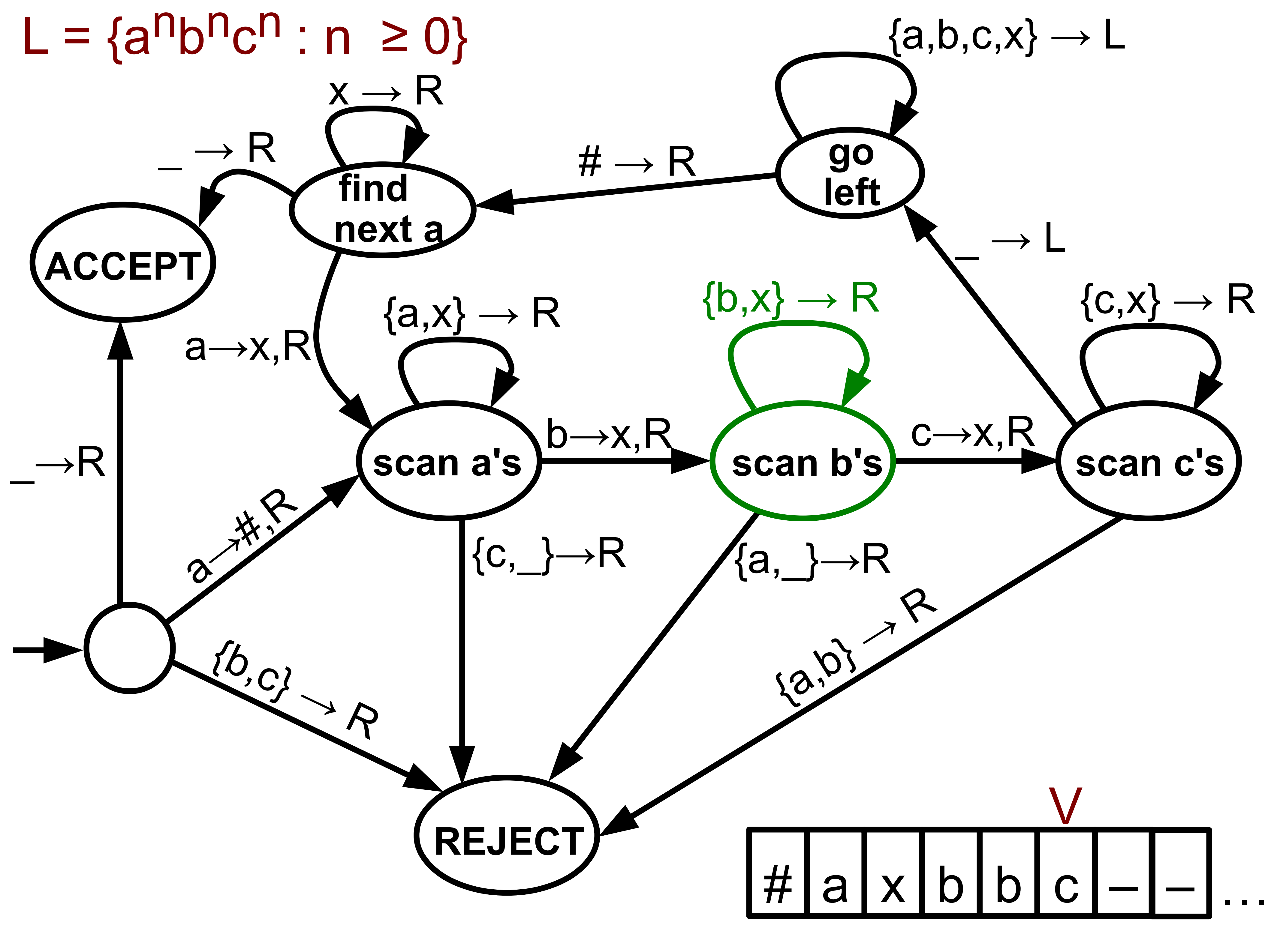
L = {aⁿbⁿcⁿ : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a b b b c _ _ ...

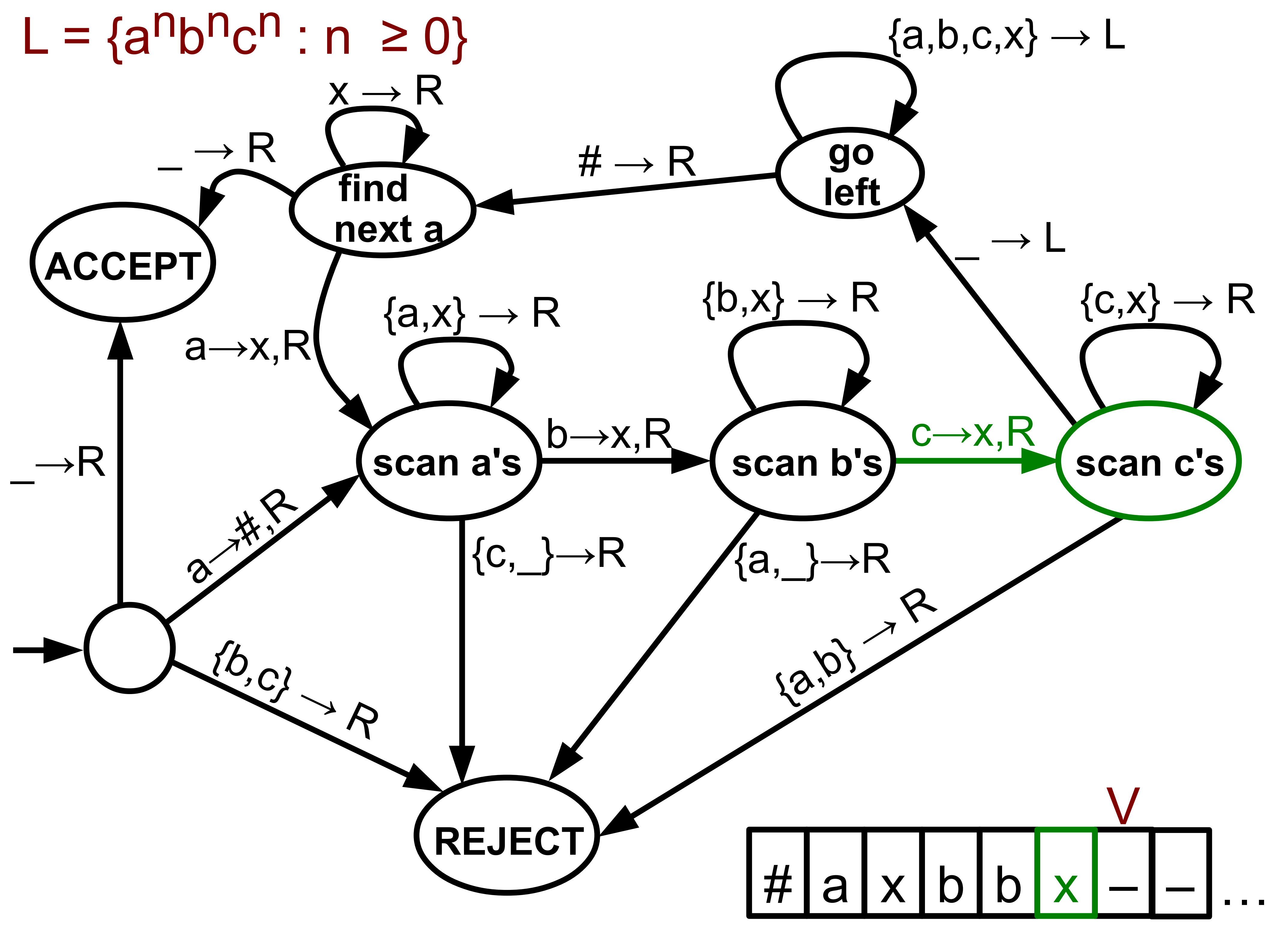


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b b c _ _ ...

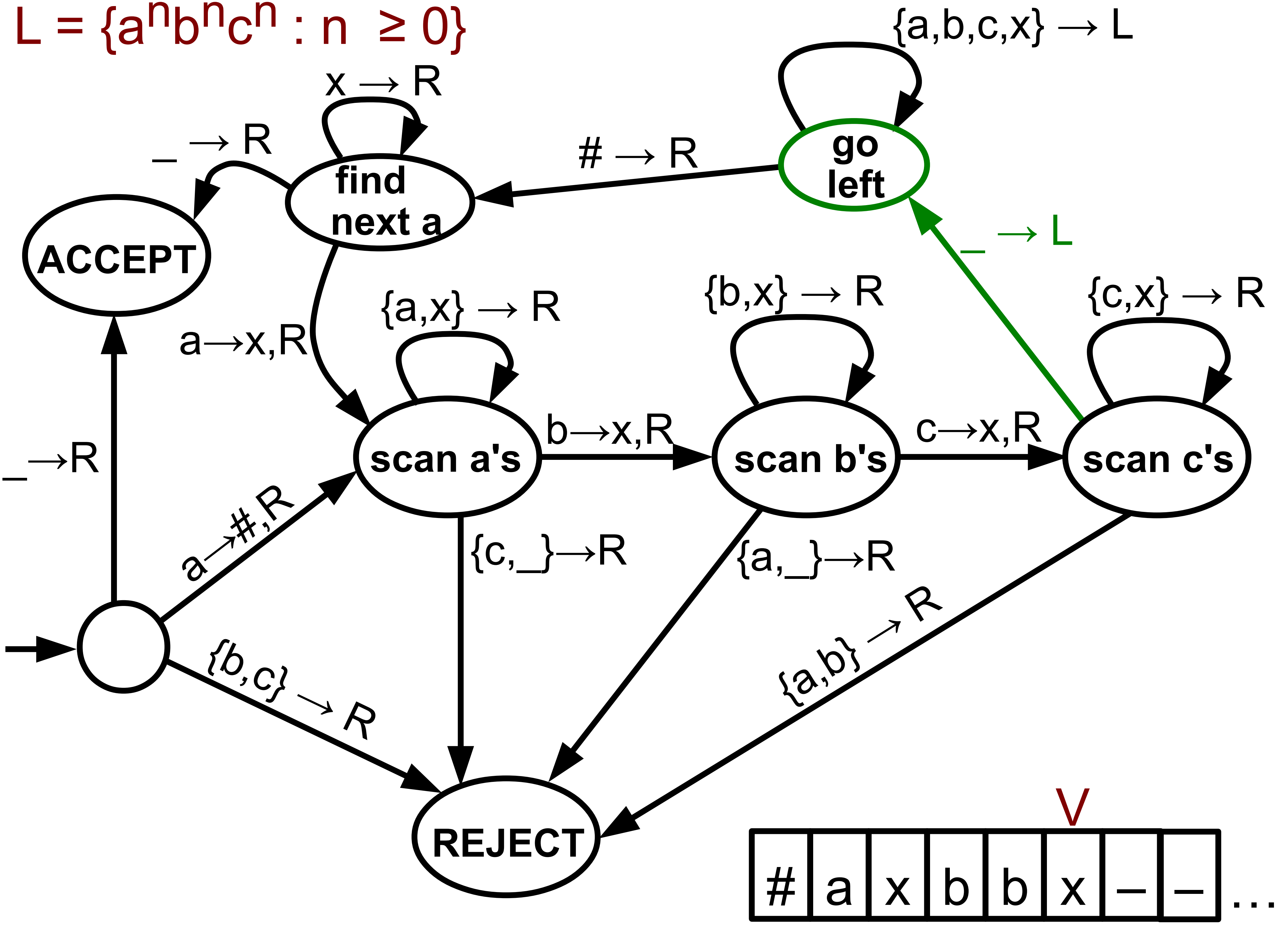
L = {$a^n b^n c^n$: n ≥ 0}


{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{a,x} → R
{c,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b b c _ _ ...

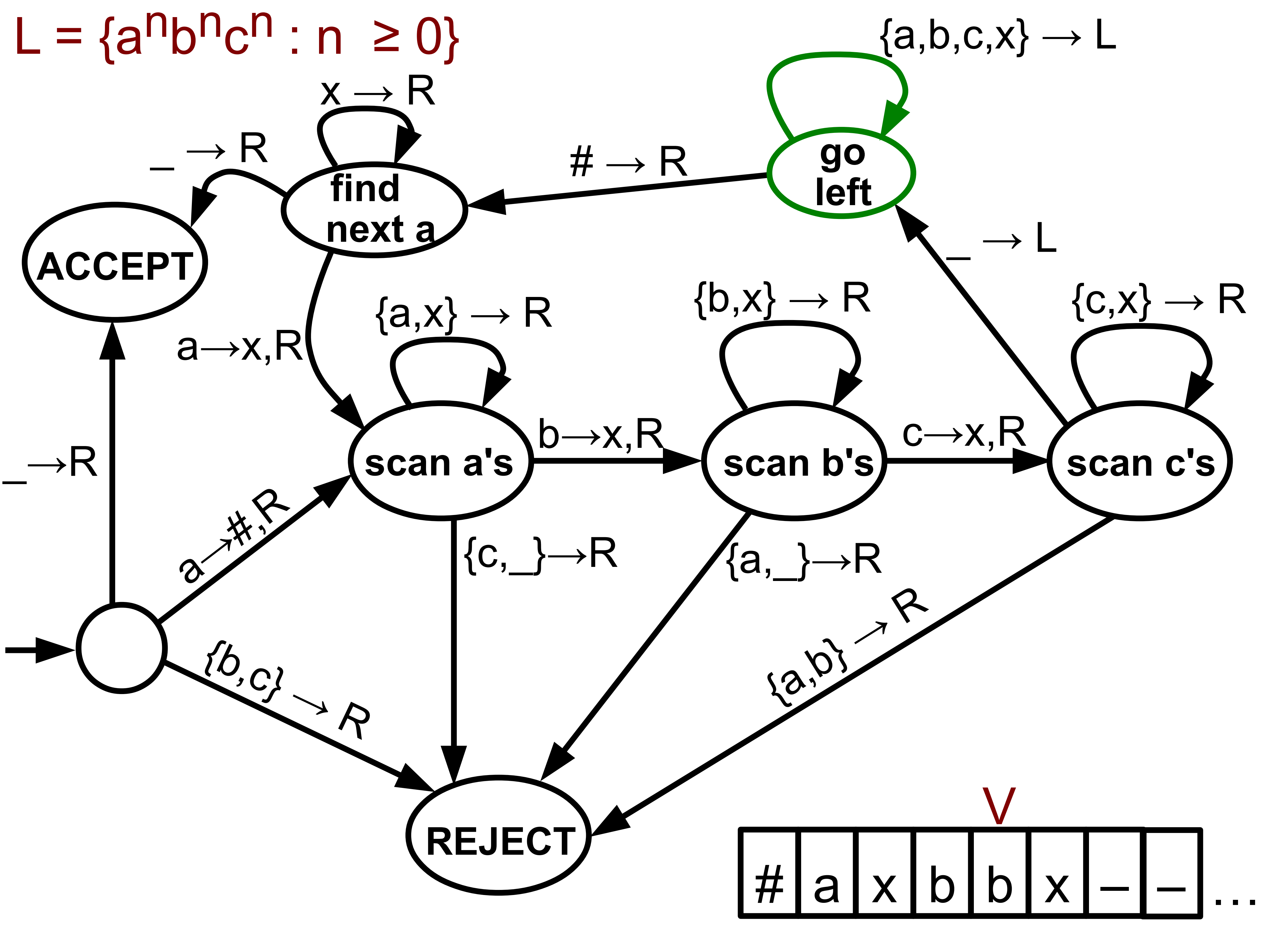

L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
scan a's
b→x,R
scan b's
c→x,R
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b b c _ _ ...


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
a x b b x _ _ ...

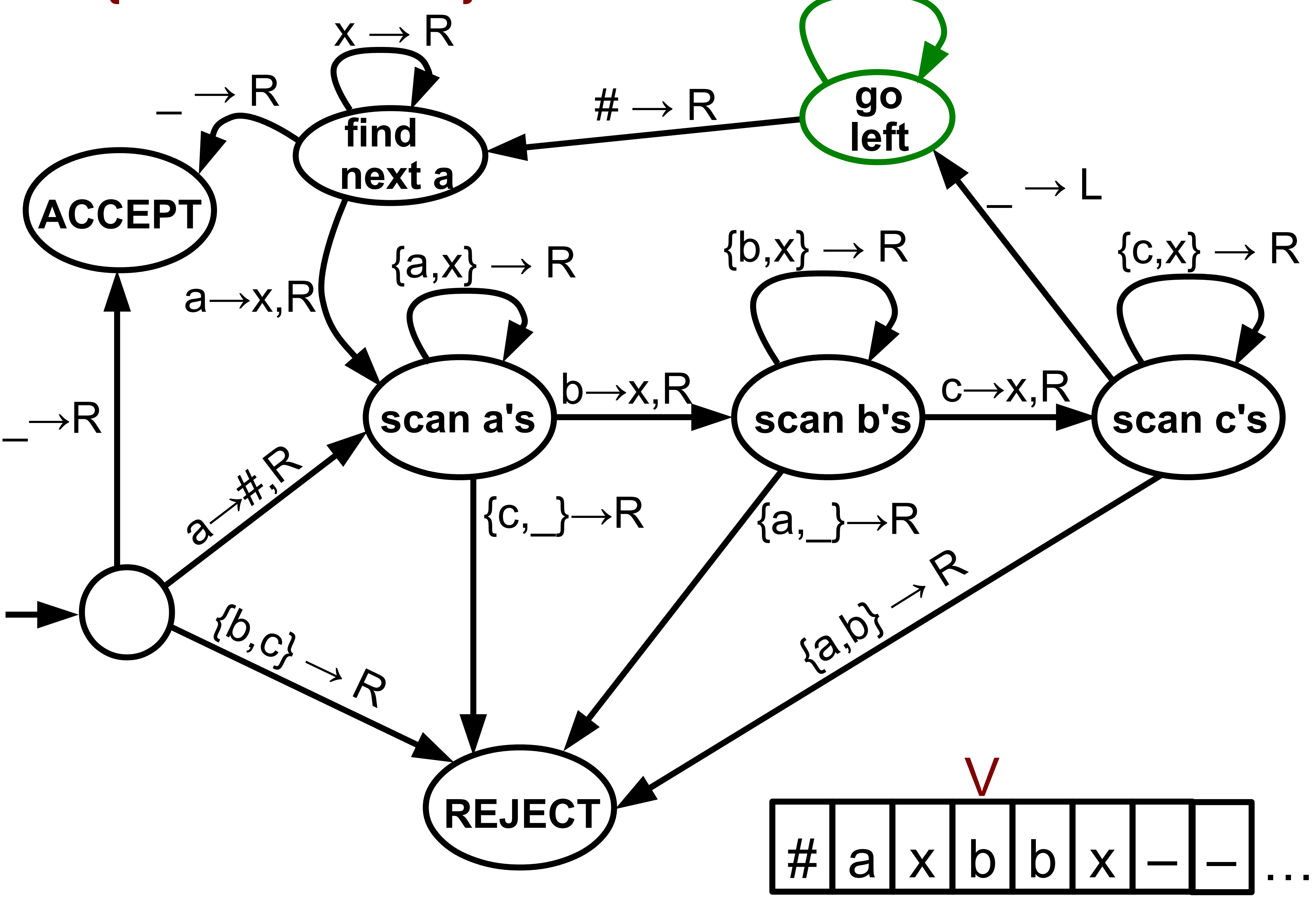
$L = \{a^n b^n c^n : n \geq 0\}$



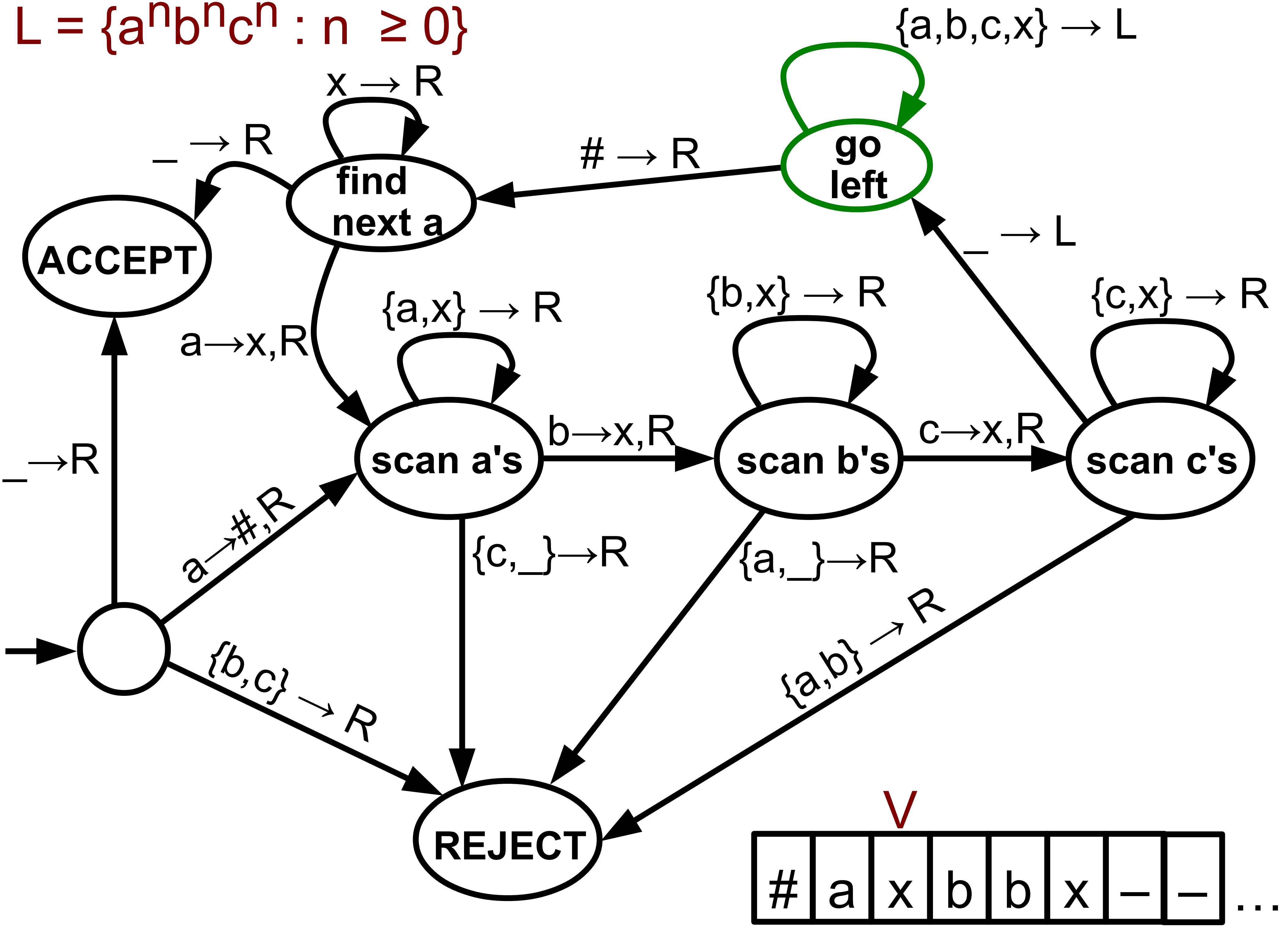
$L = \{a^n b^n c^n : n \geq 0\}$



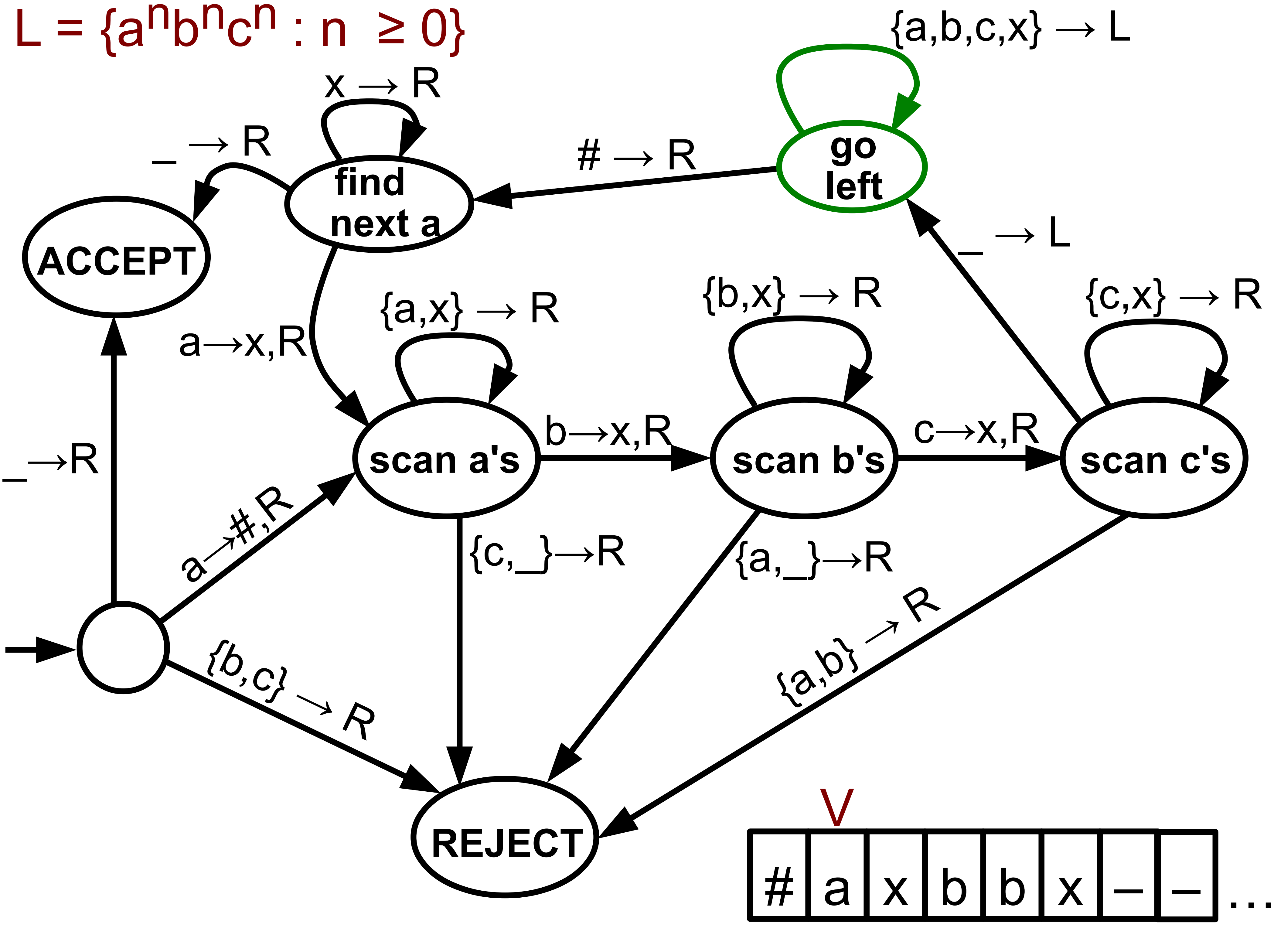
$L = \{a^n b^n c^n : n \geq 0\}$



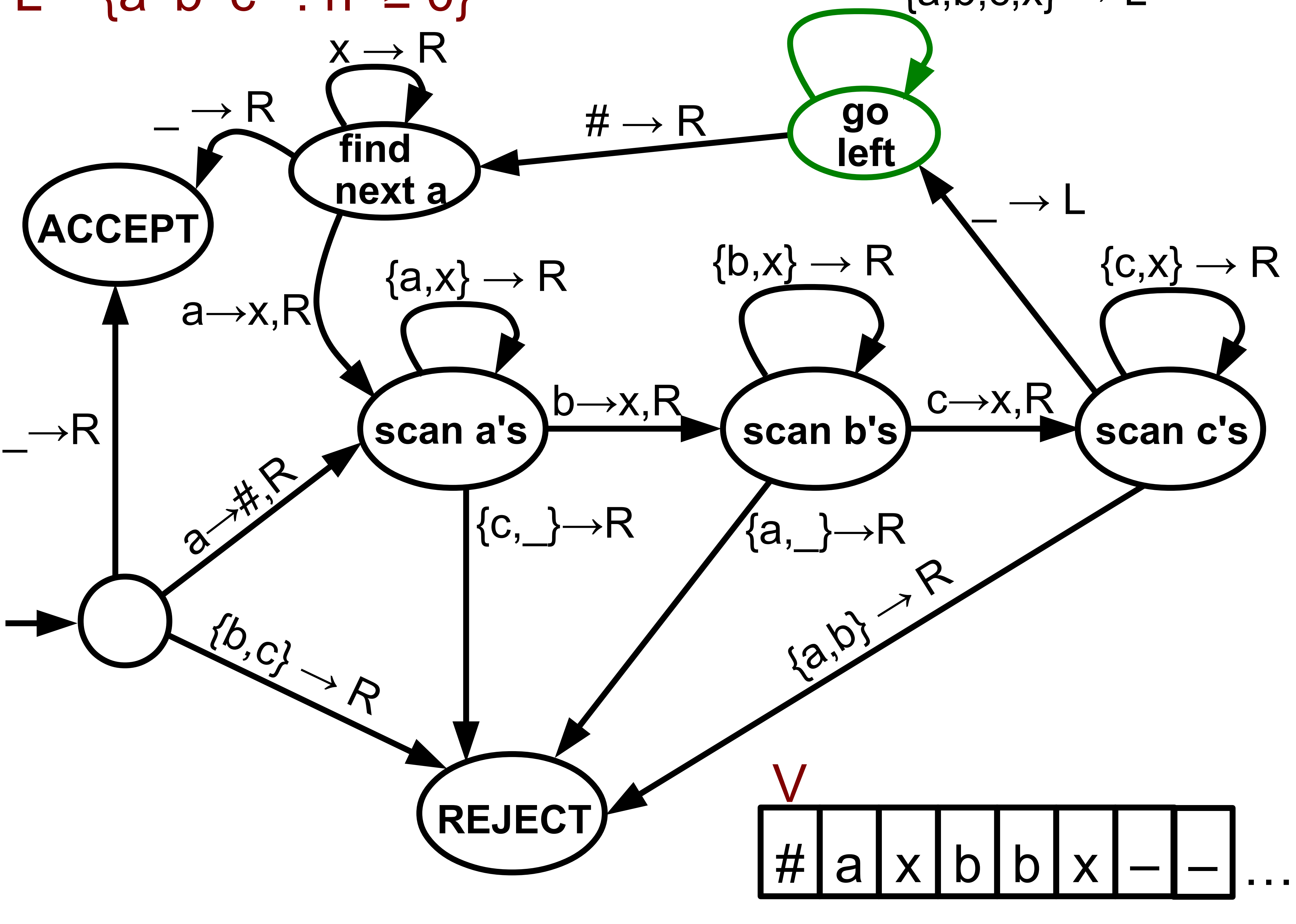
$L = \{a^n b^n c^n : n \geq 0\}$



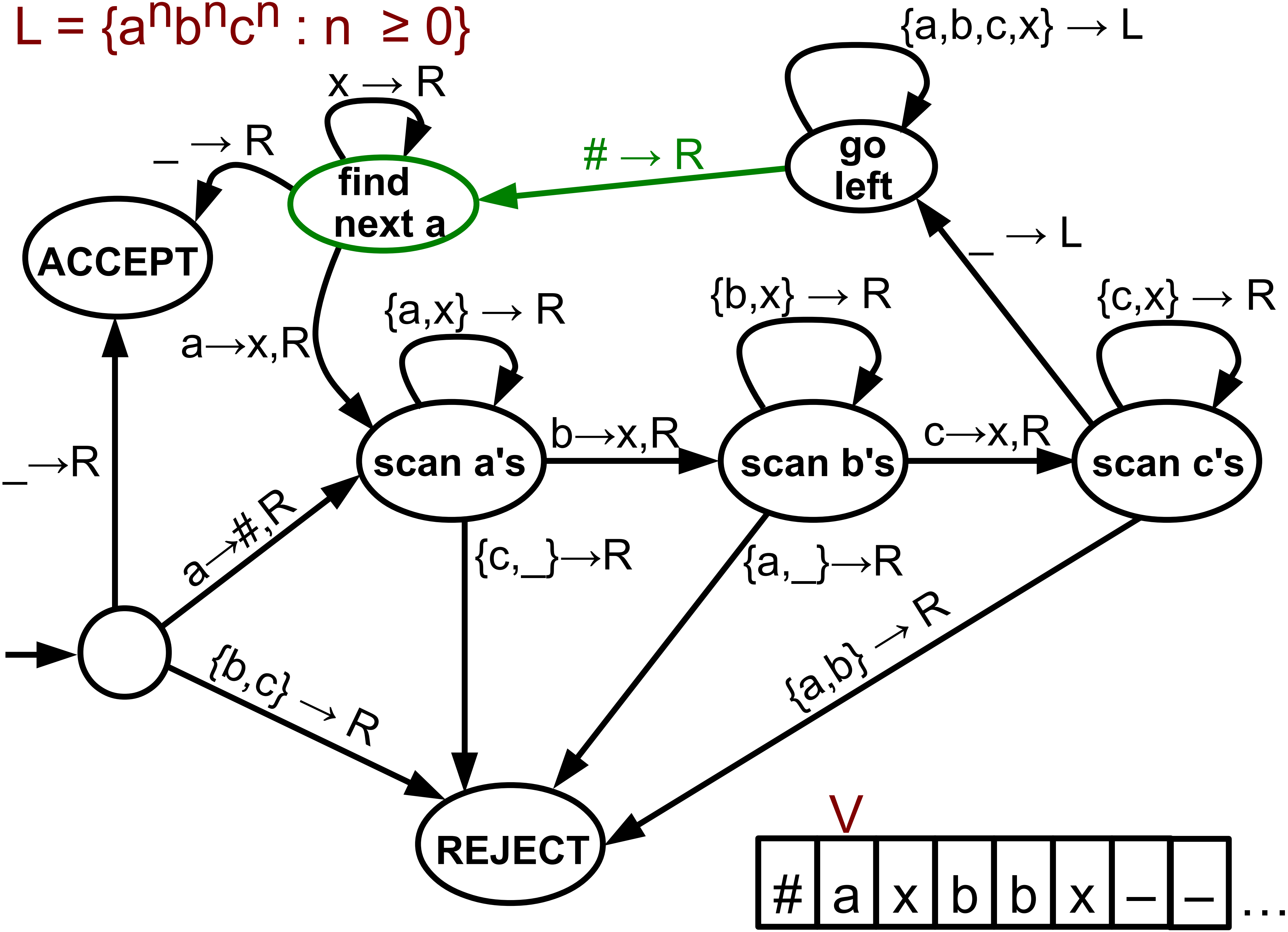
$L = \{a^n b^n c^n : n \geq 0\}$



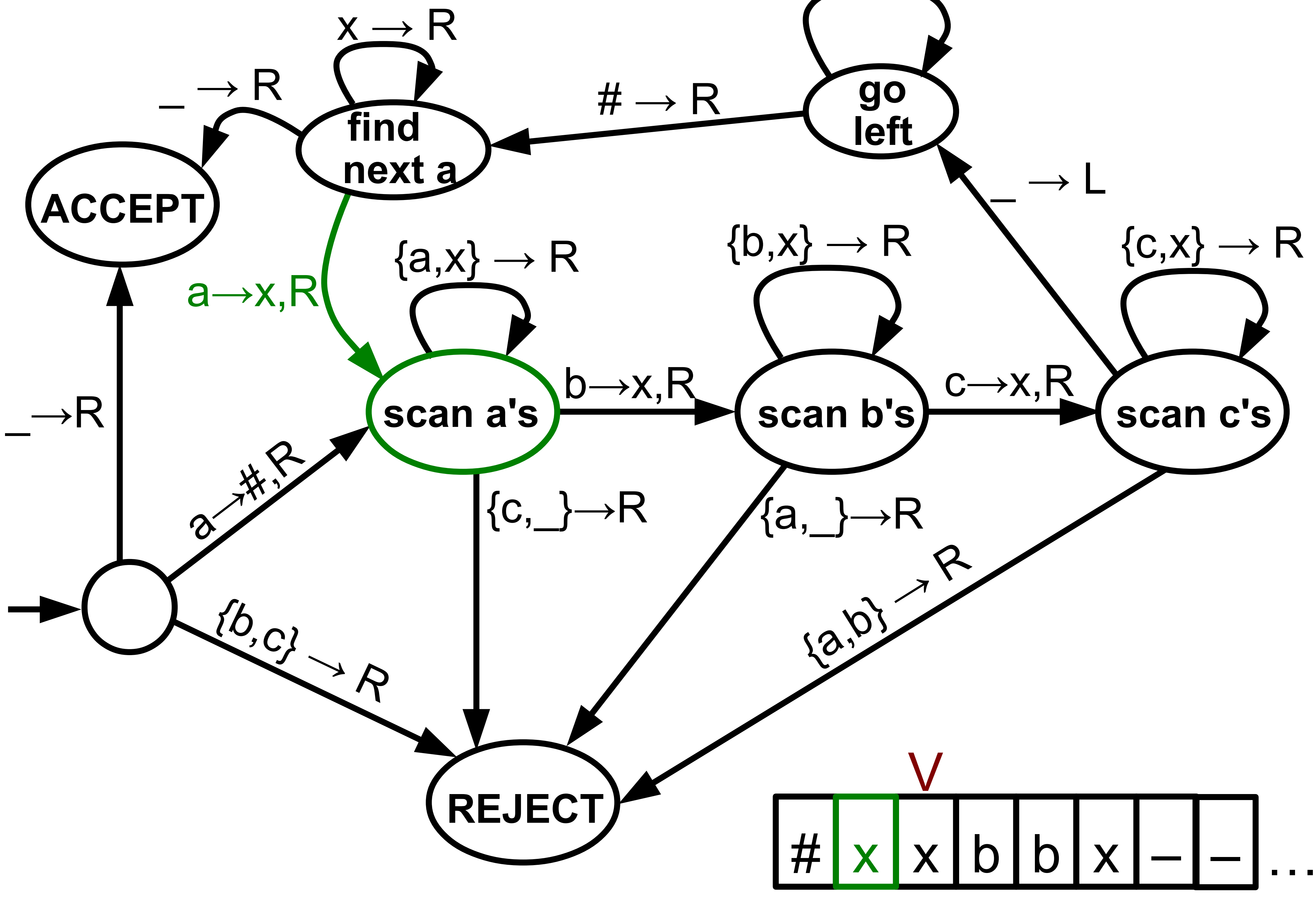
$L = \{a^n b^n c^n : n \geq 0\}$



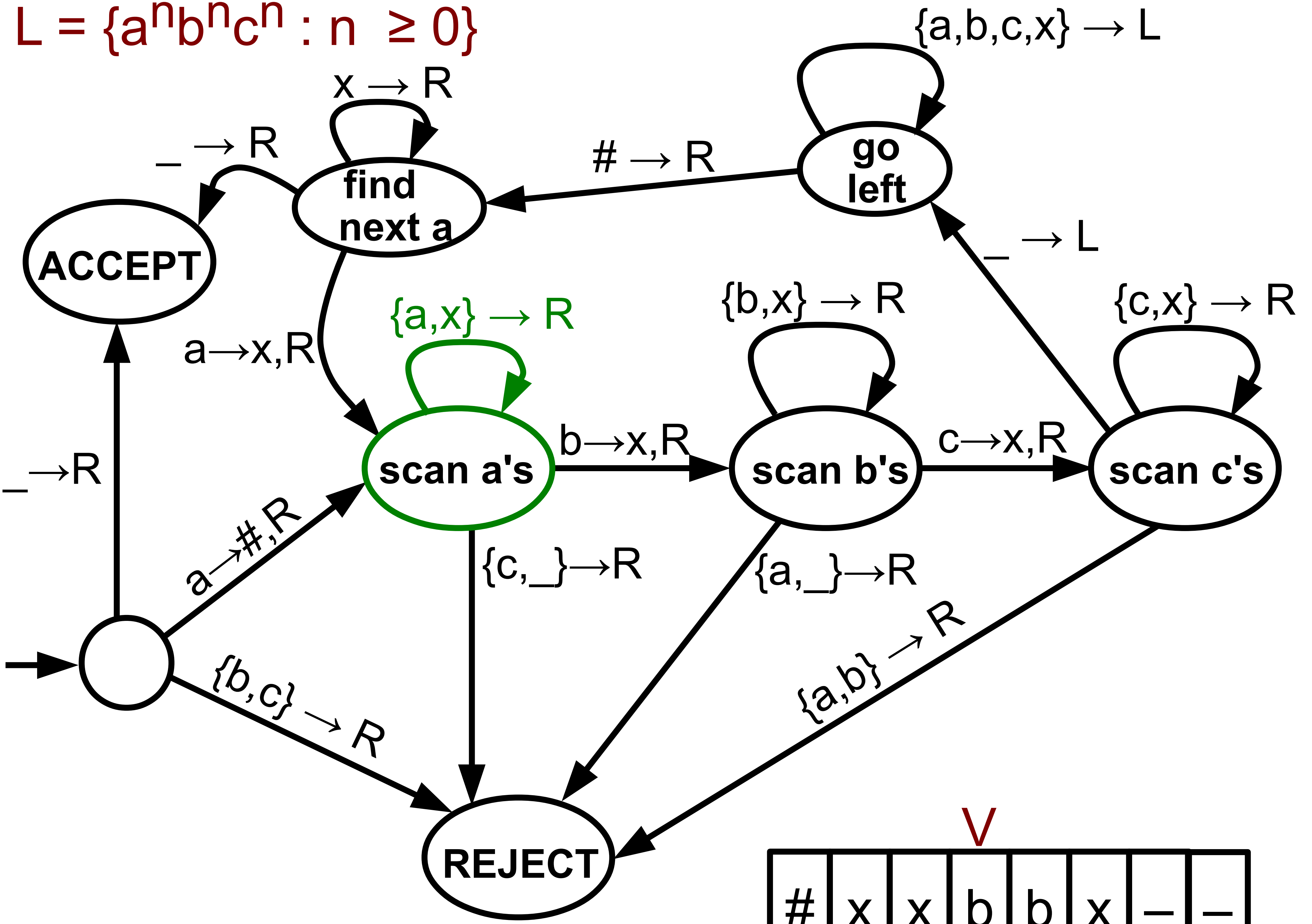
$L = \{a^n b^n c^n : n \geq 0\}$



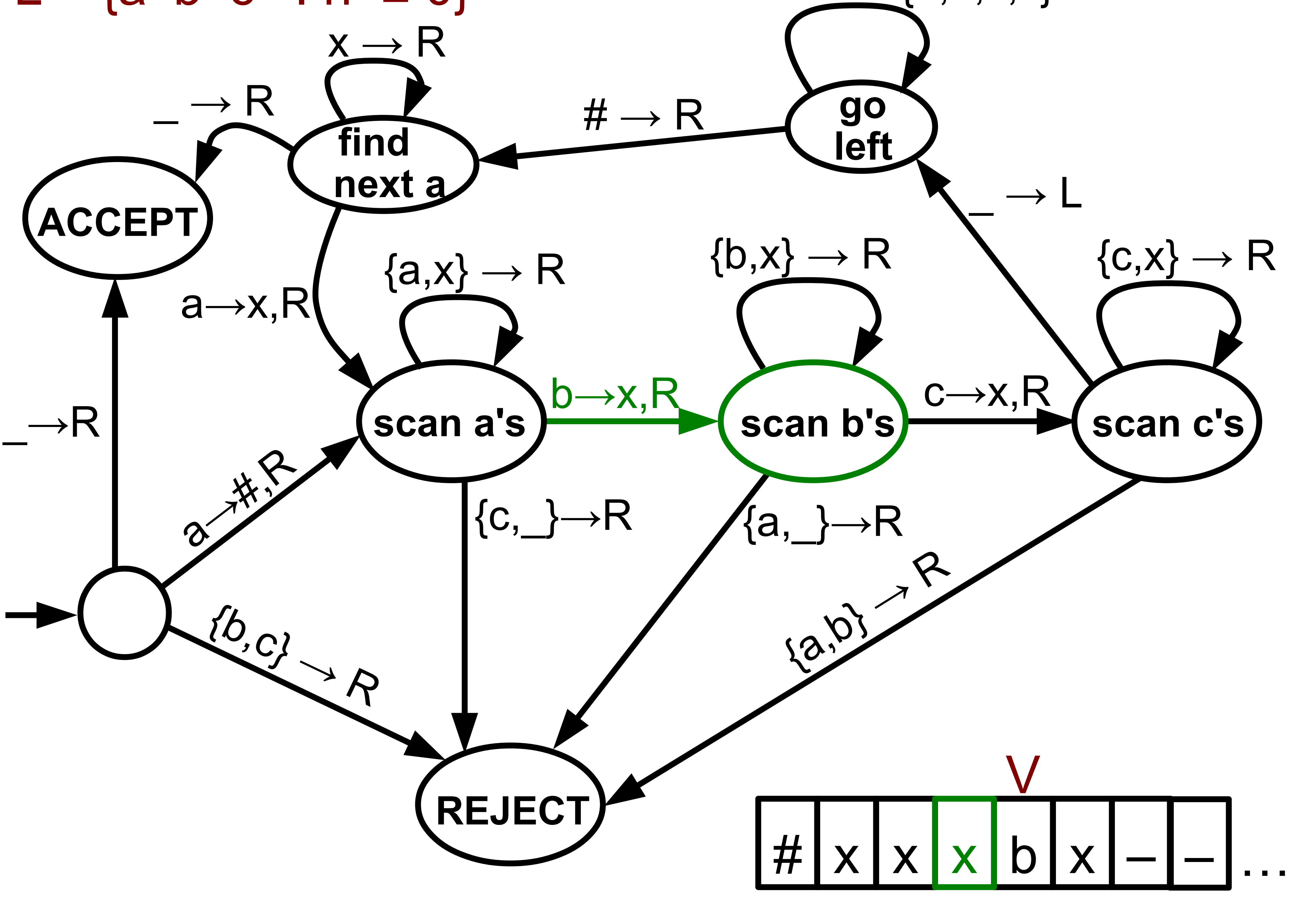
$L = \{a^n b^n c^n : n \geq 0\}$

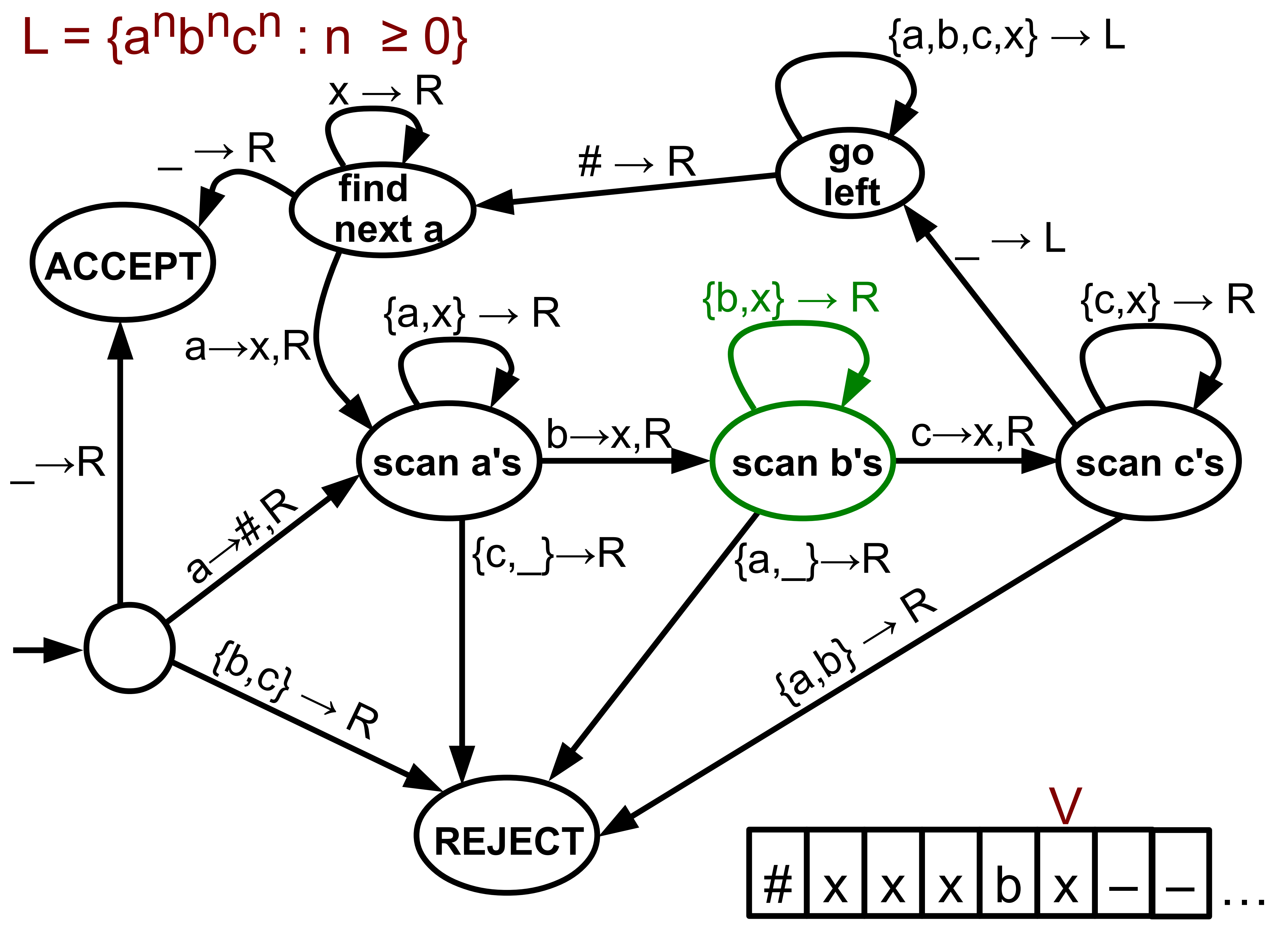


$L = \{a^n b^n c^n : n \geq 0\}$



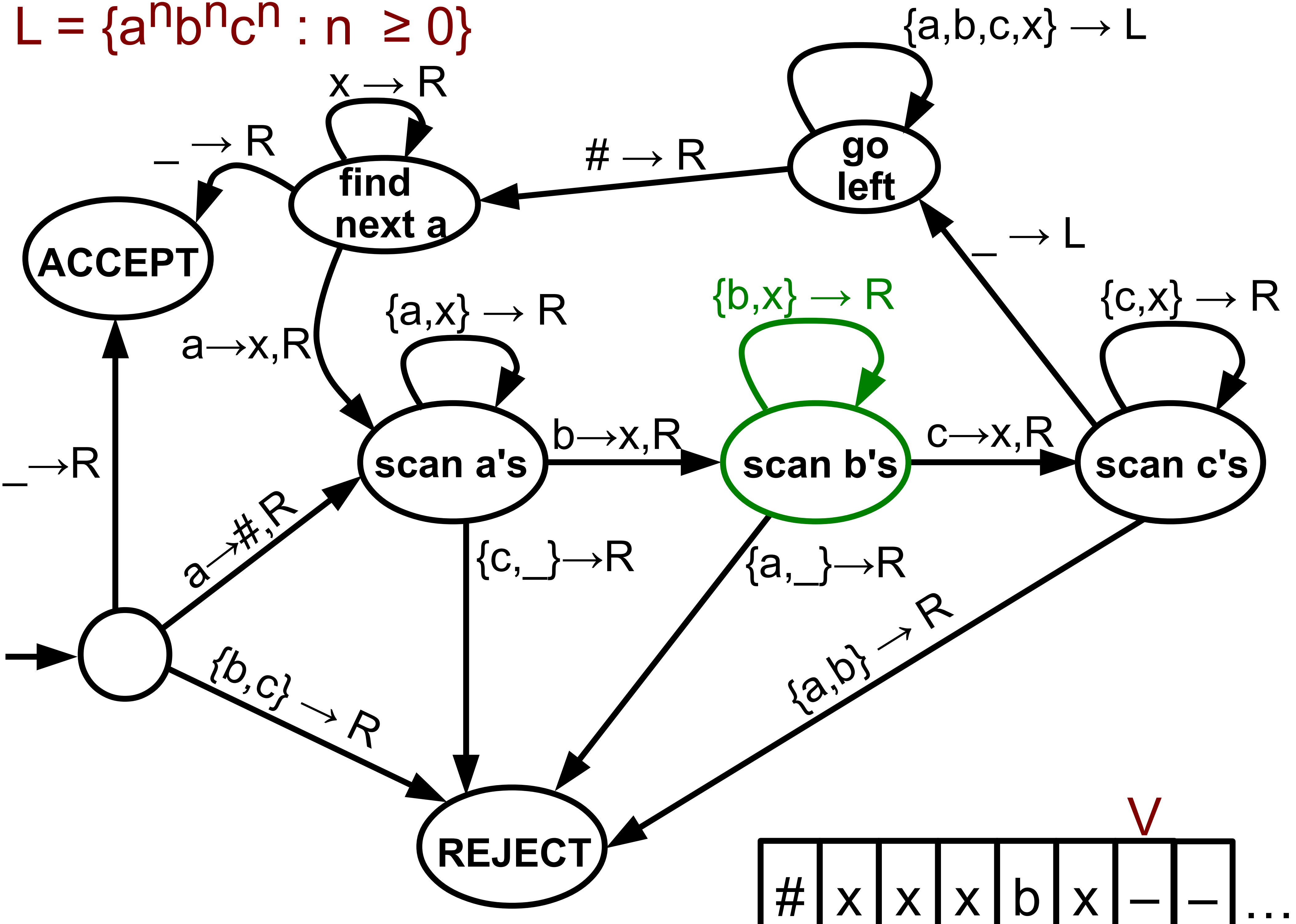
$L = \{a^n b^n c^n : n \geq 0\}$



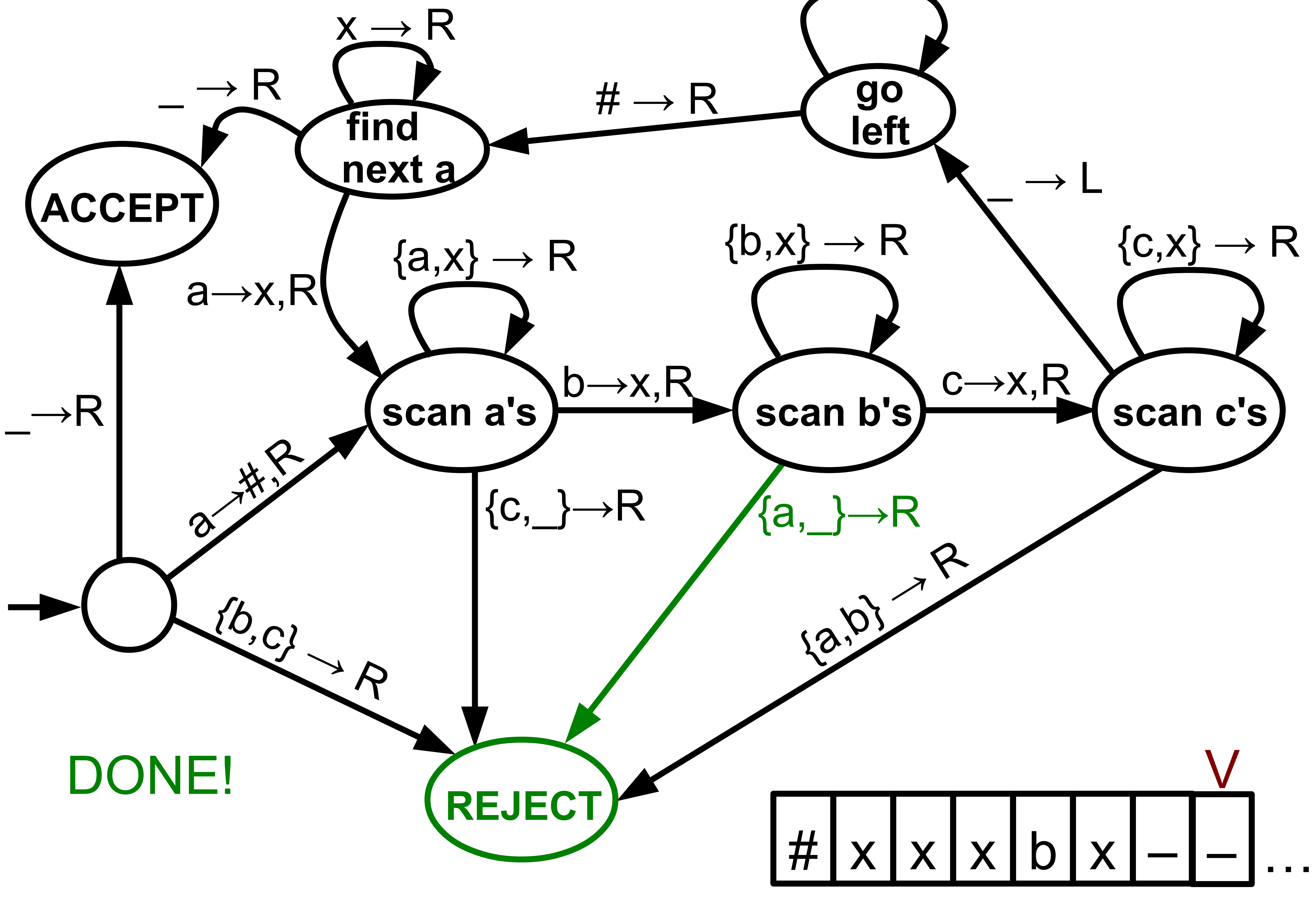


L = {a^n b^n c^n : n ≥ 0}
{a,b,c,x} → L
x → R
_ → R
→ R
go left
find next a
ACCEPT
_ → L
{b,x} → R
{c,x} → R
{a,x} → R
a→x,R
b→x,R
c→x,R
scan a's
scan b's
scan c's
_→R
a→#,R
{c,_}→R
{a,_}→R
{a,b} → R
{b,c} → R
REJECT
V
x x x b x _ _ …

$L = \{a^n b^n c^n : n \geq 0\}$



$L = \{a^n b^n c^n : n \geq 0\}$



Example: TM for $L = \{a^{2^n} : n \geq 0\}$

$= \{a, aa, aaaa, aaaaaaaa, \ldots\}$

M := "On input w,

1) if only one a, ACCEPT

2) cross off every other a on the tape

3) if the number of a's is odd, REJECT

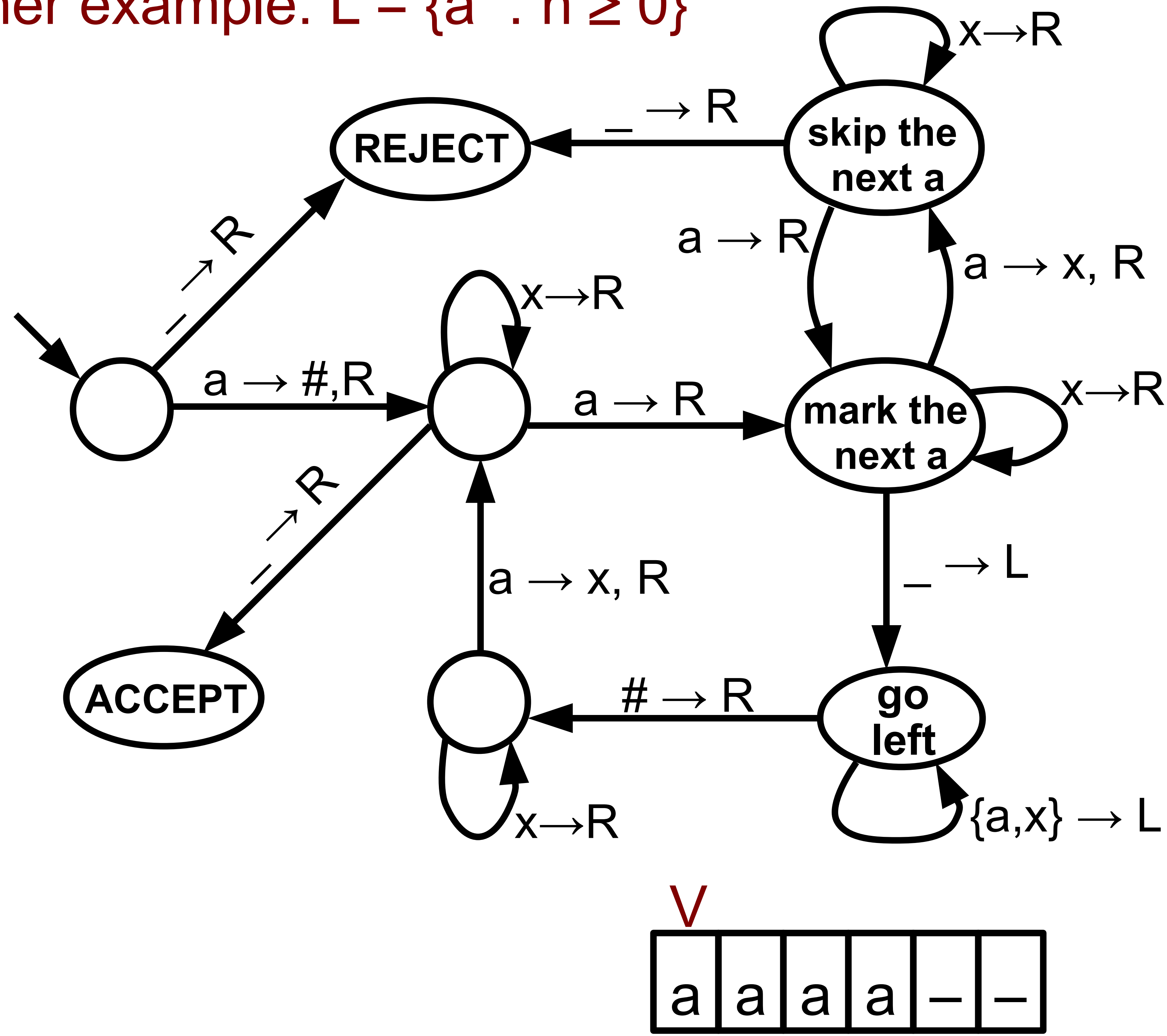
4) Go back to 1)"

For instance:

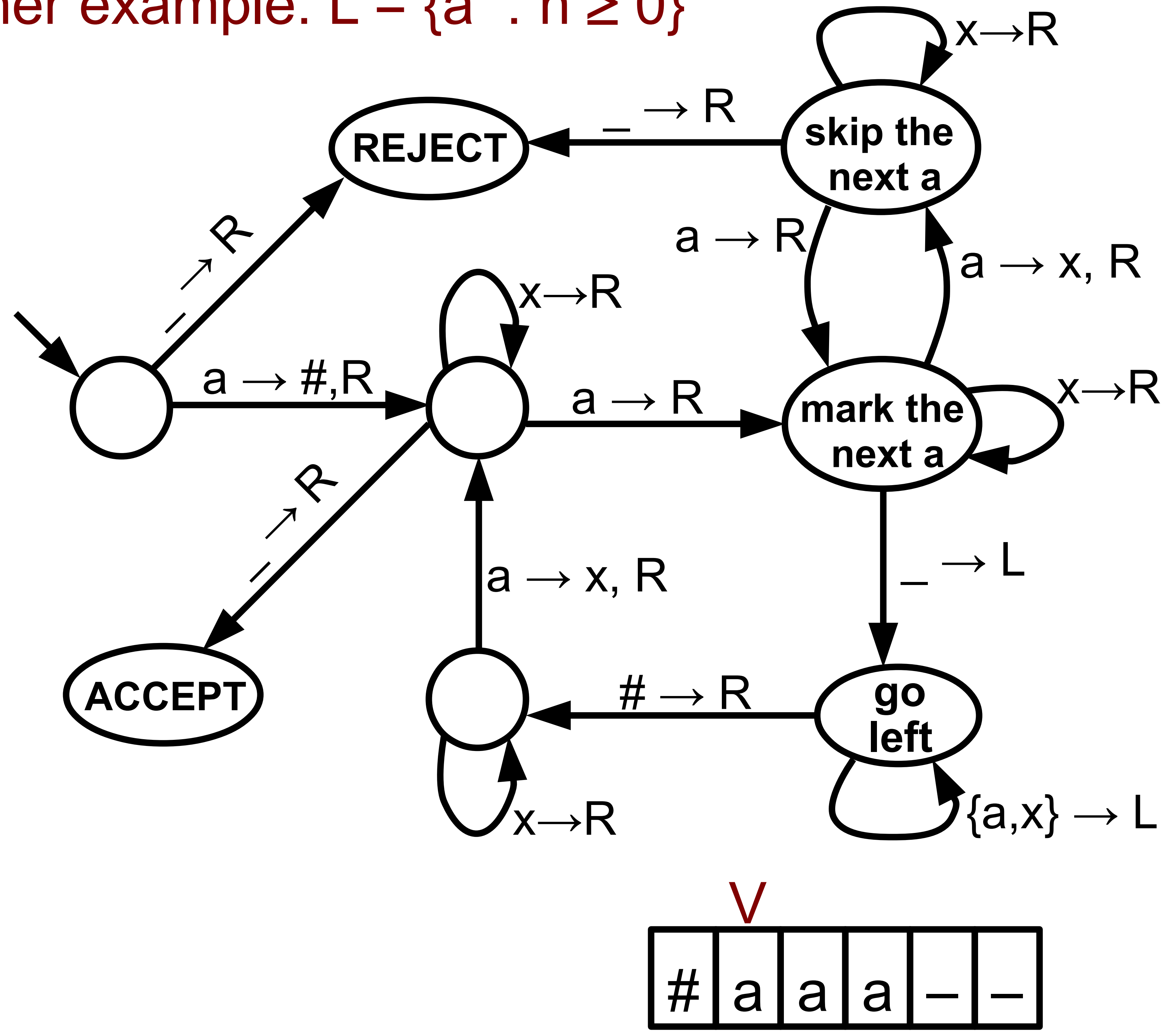
8 a's → 4 a's → 2 a's → 1 a → ACCEPT

12 a's → 6 a's → 3 a's → REJECT

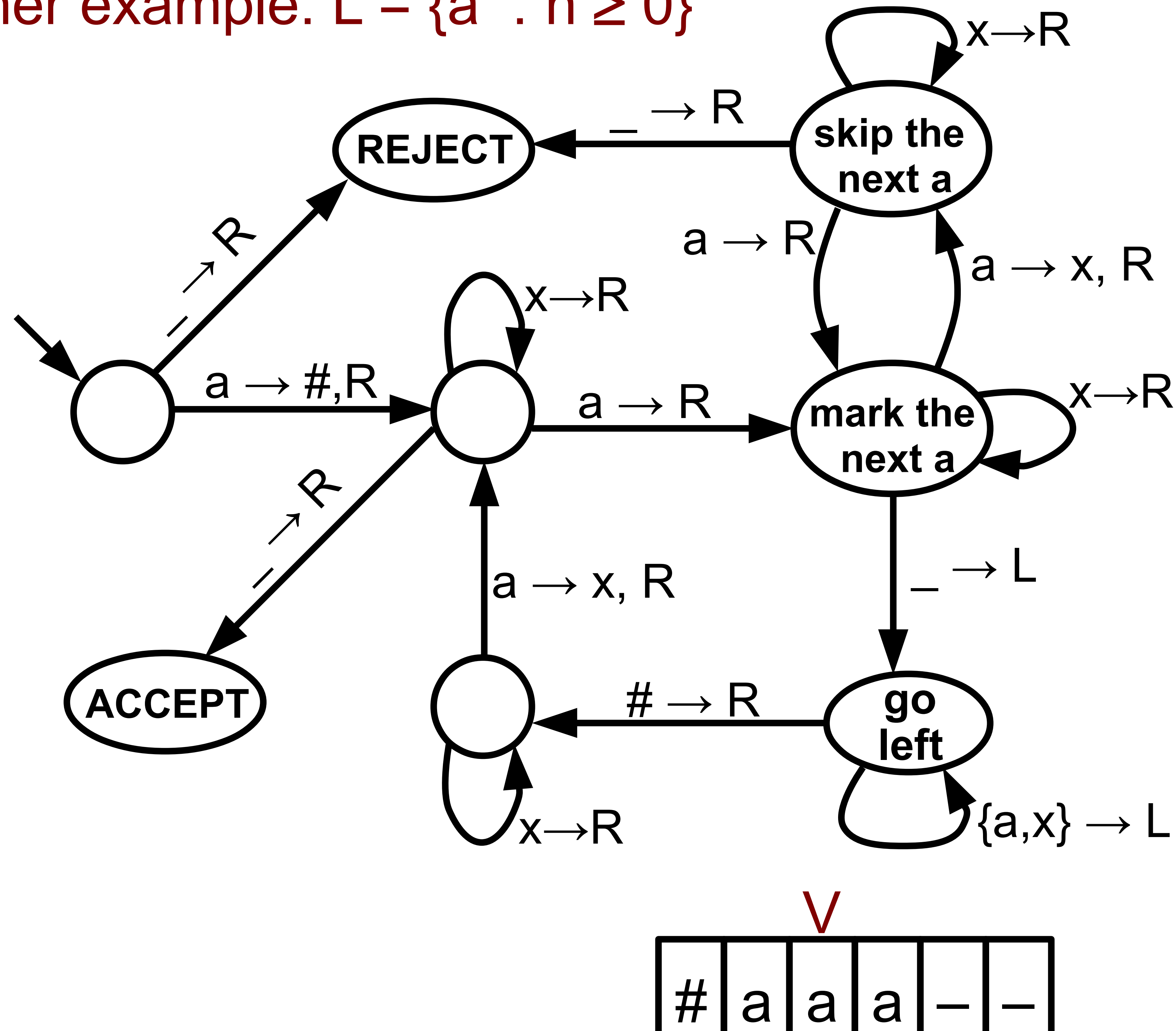
Another example: L = $\{a^{2^n} : n \geq 0\}$



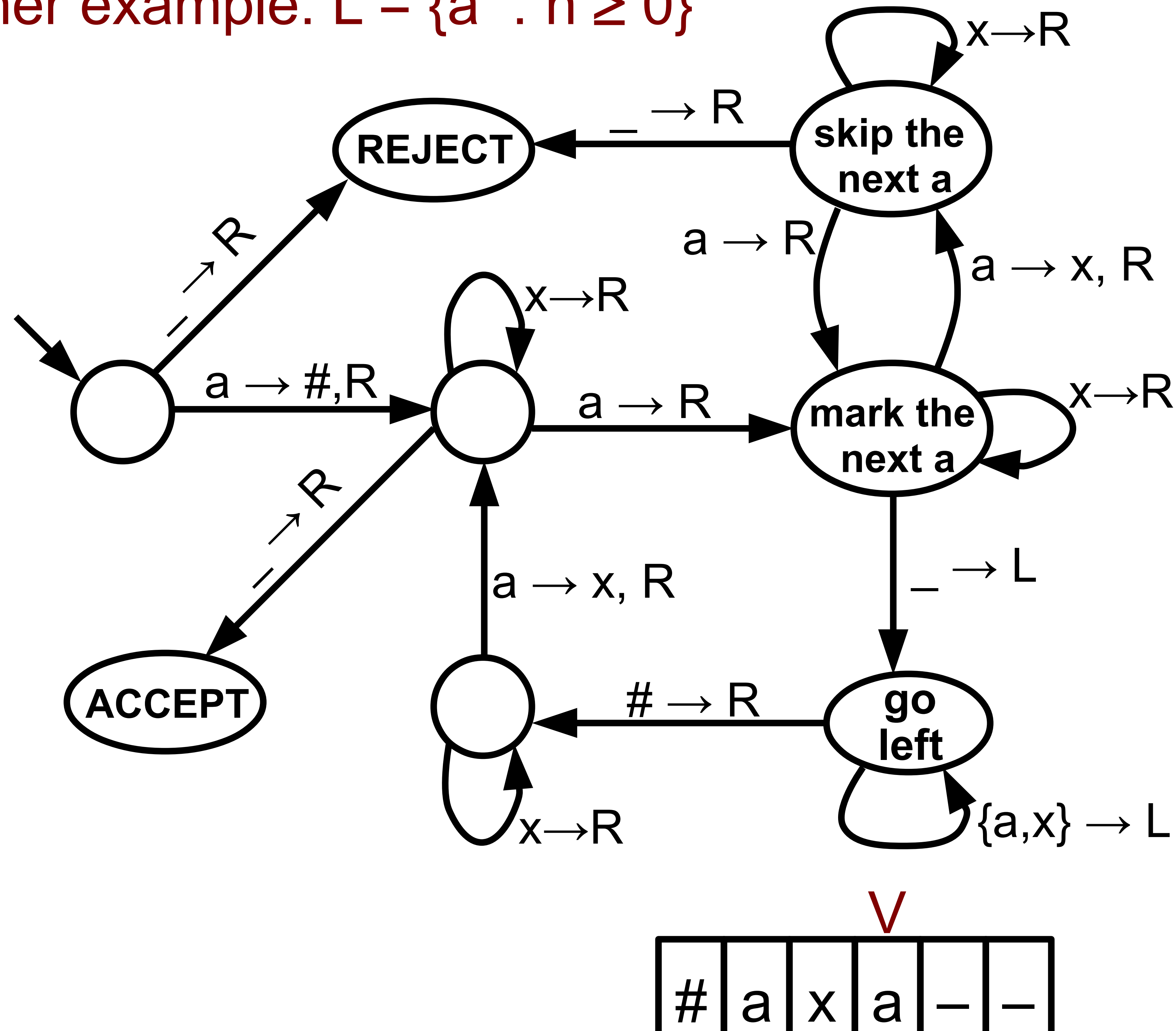
Another example: $L = \{a^{2^n} : n \geq 0\}$



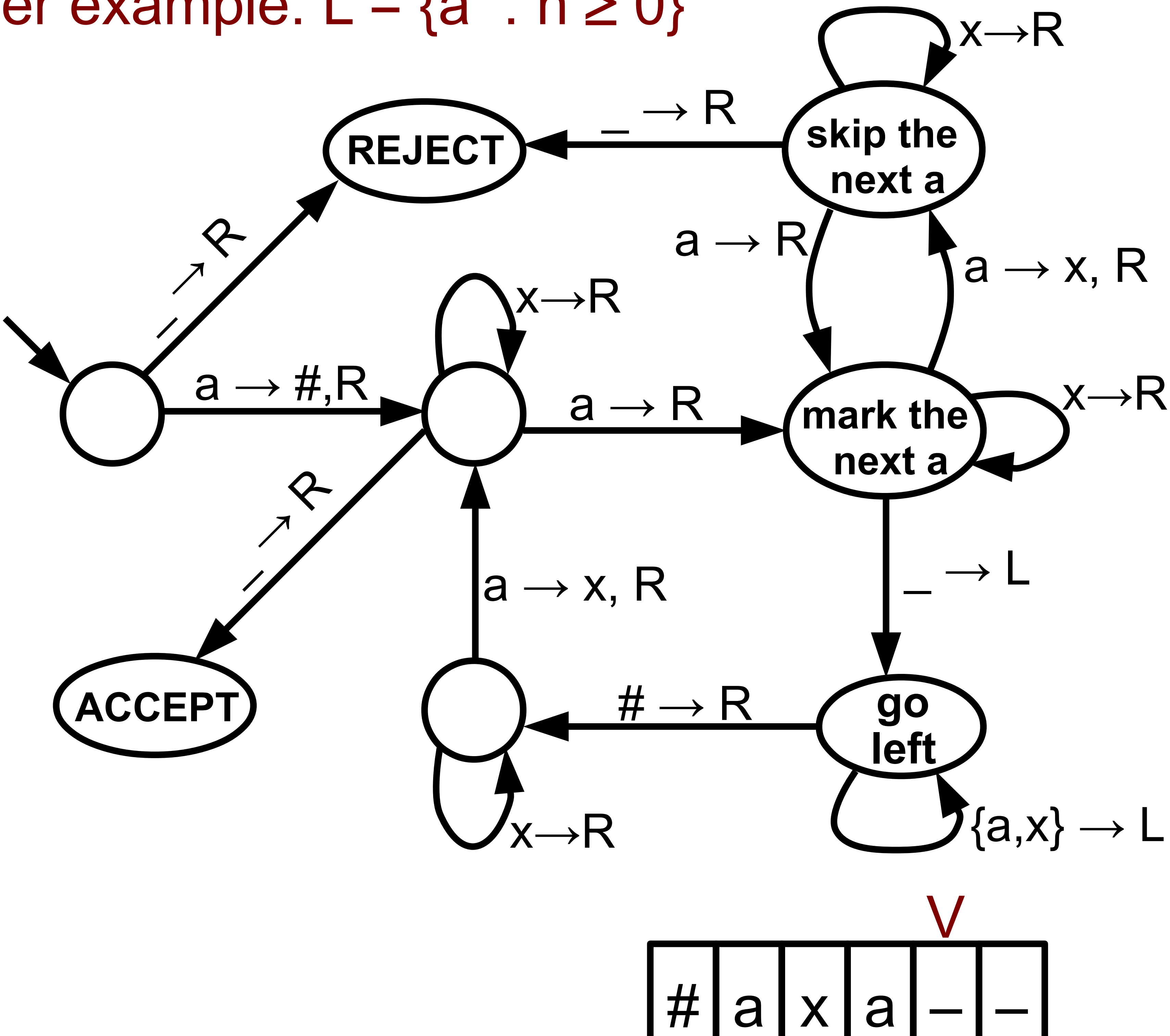
Another example: $L = \{a^{2^n} : n \geq 0\}$



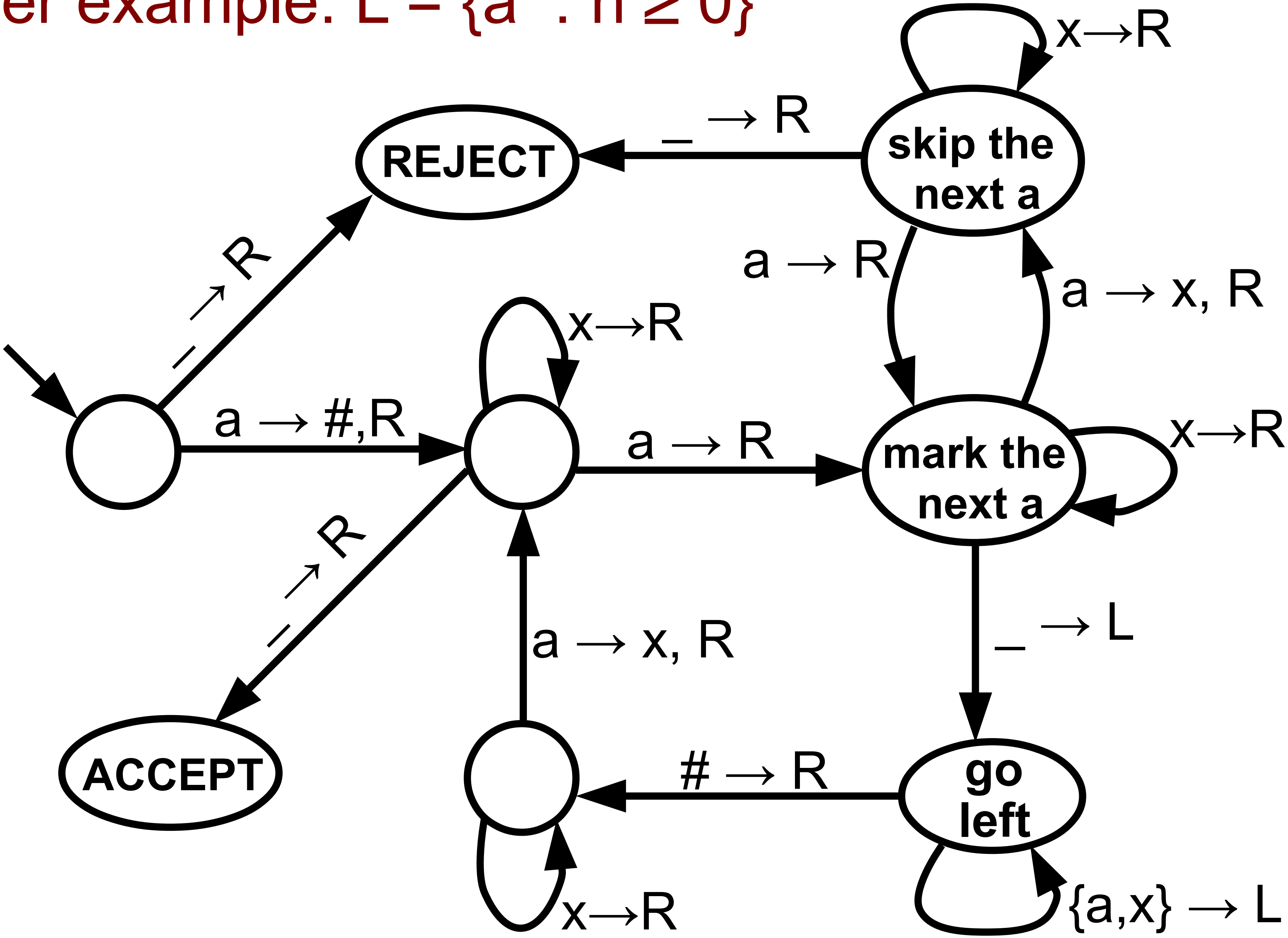
Another example: $L = \{a^{2^n} : n \geq 0\}$

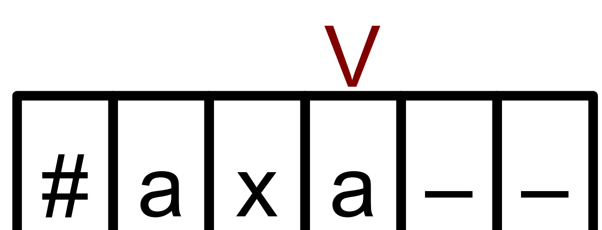


Another example: $L = \{a^{2^n} : n \geq 0\}$

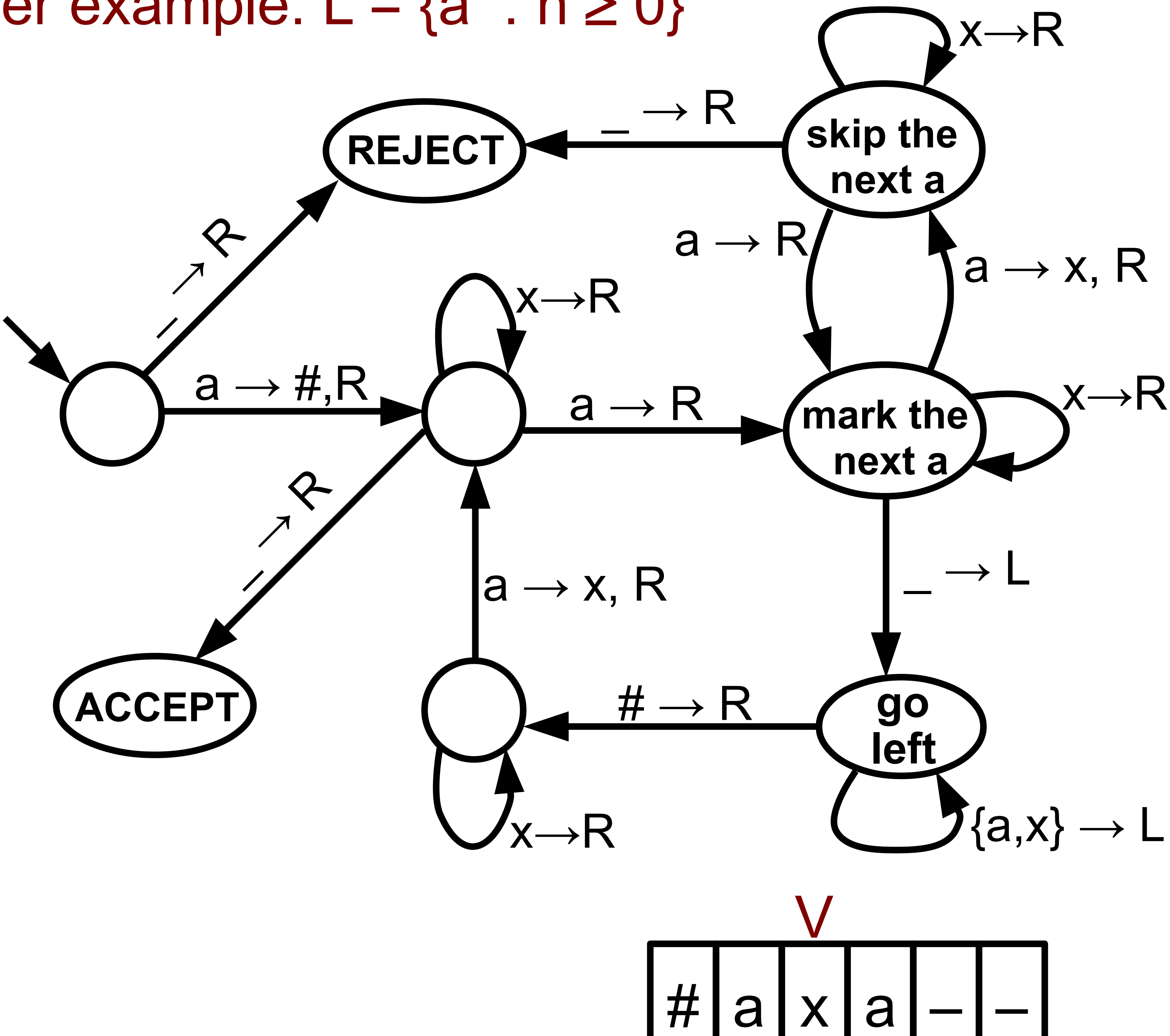


Another example: $L = \{a^{2^n} : n \geq 0\}$

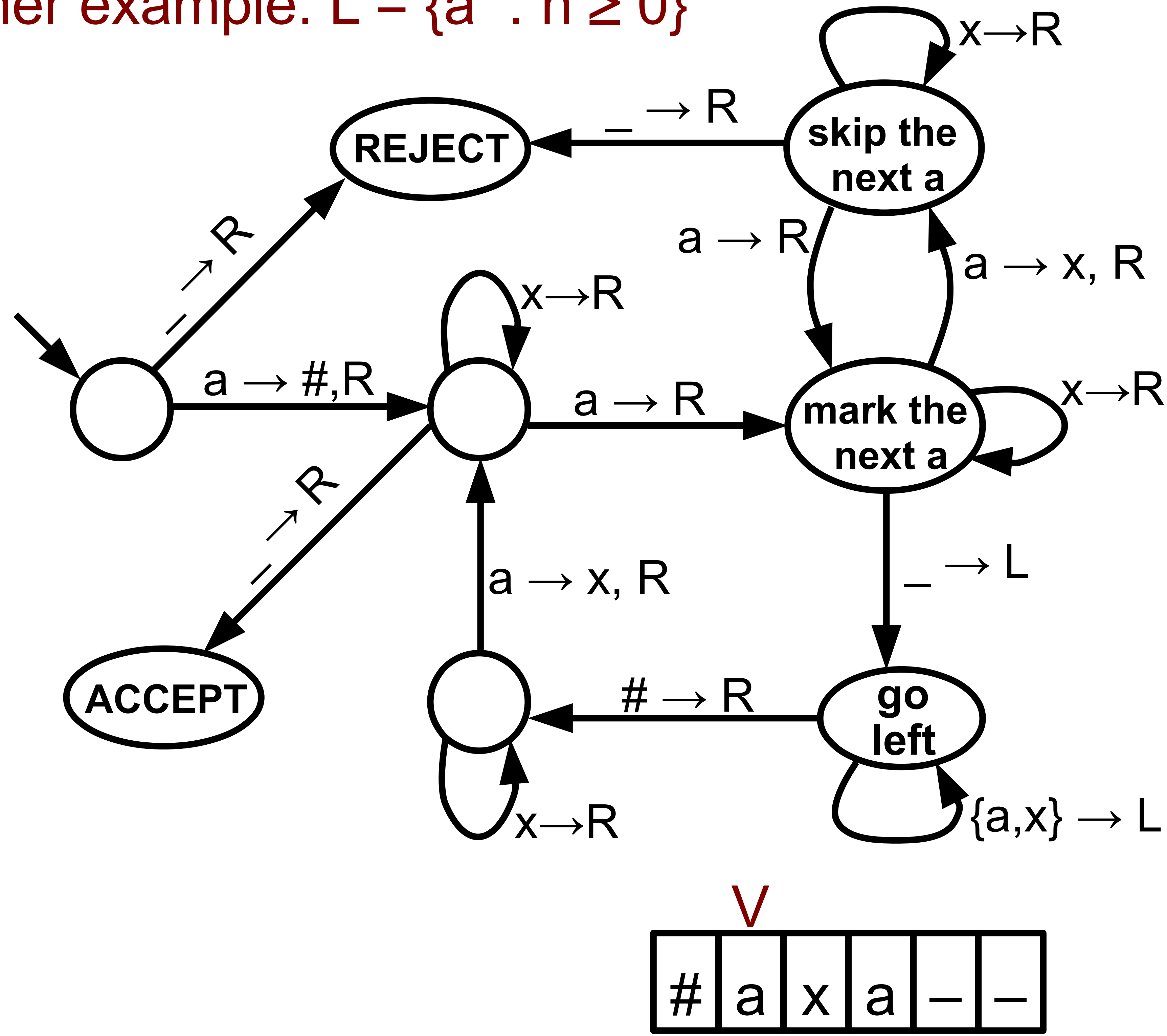




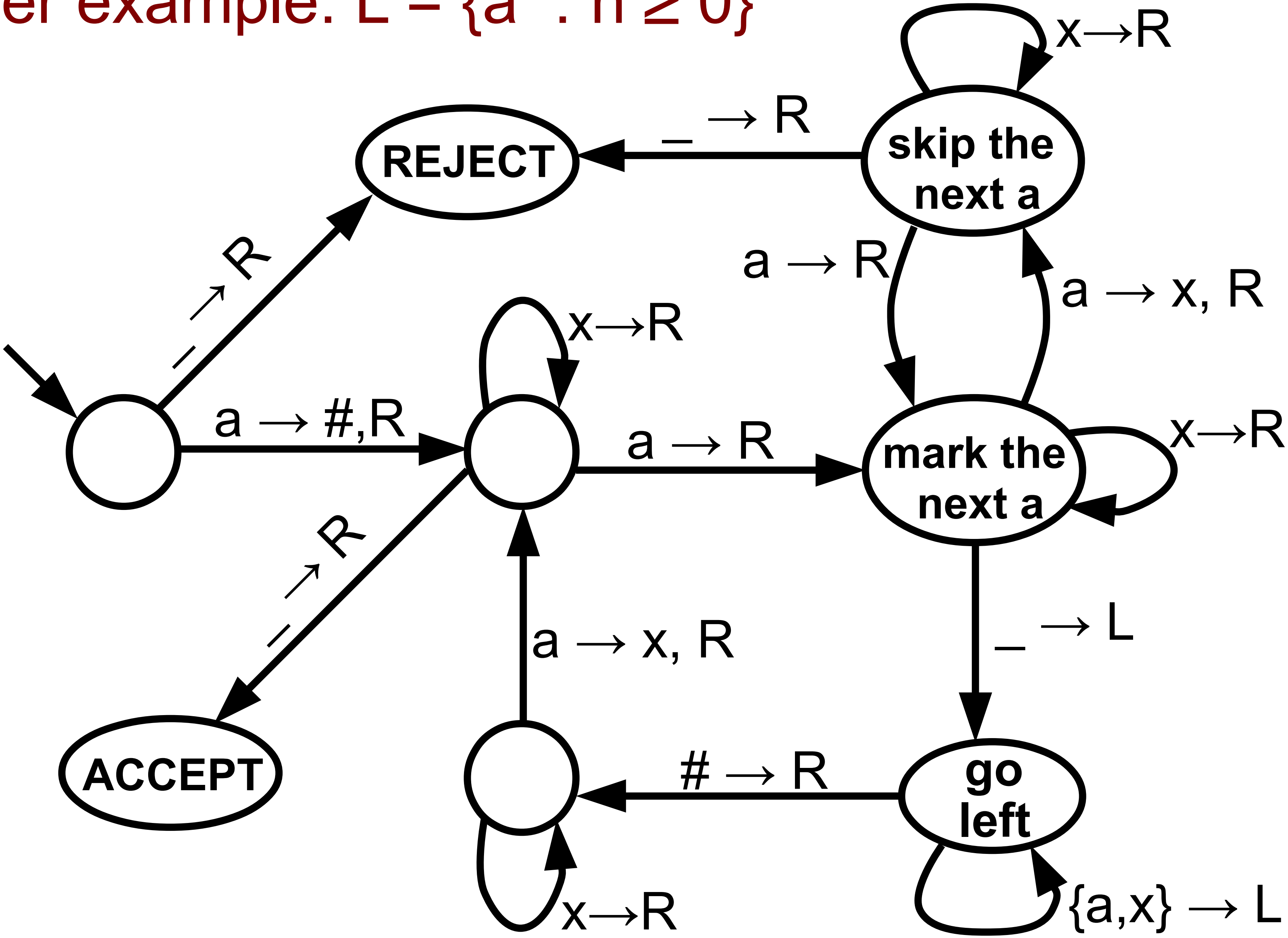
Another example: L = {a^{2^n} : n ≥ 0}

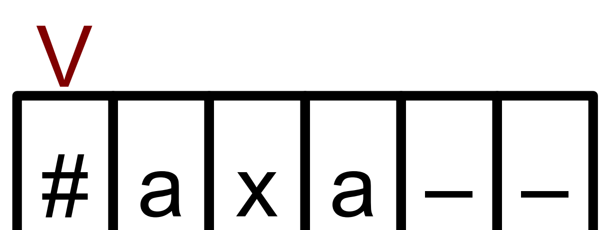


Another example: $L = \{a^{2^n} : n \geq 0\}$

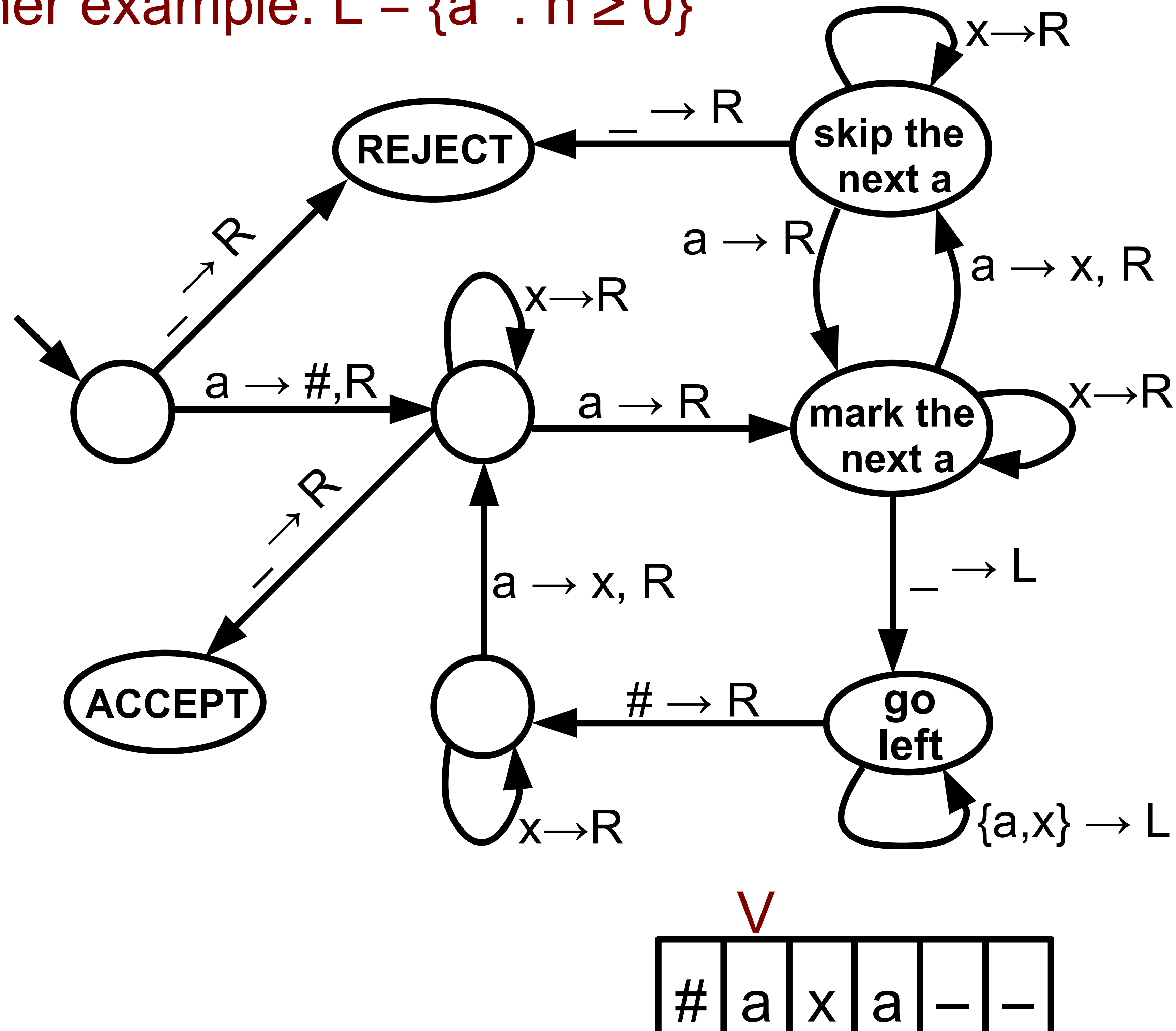


Another example: $L = \{a^{2^n} : n \geq 0\}$

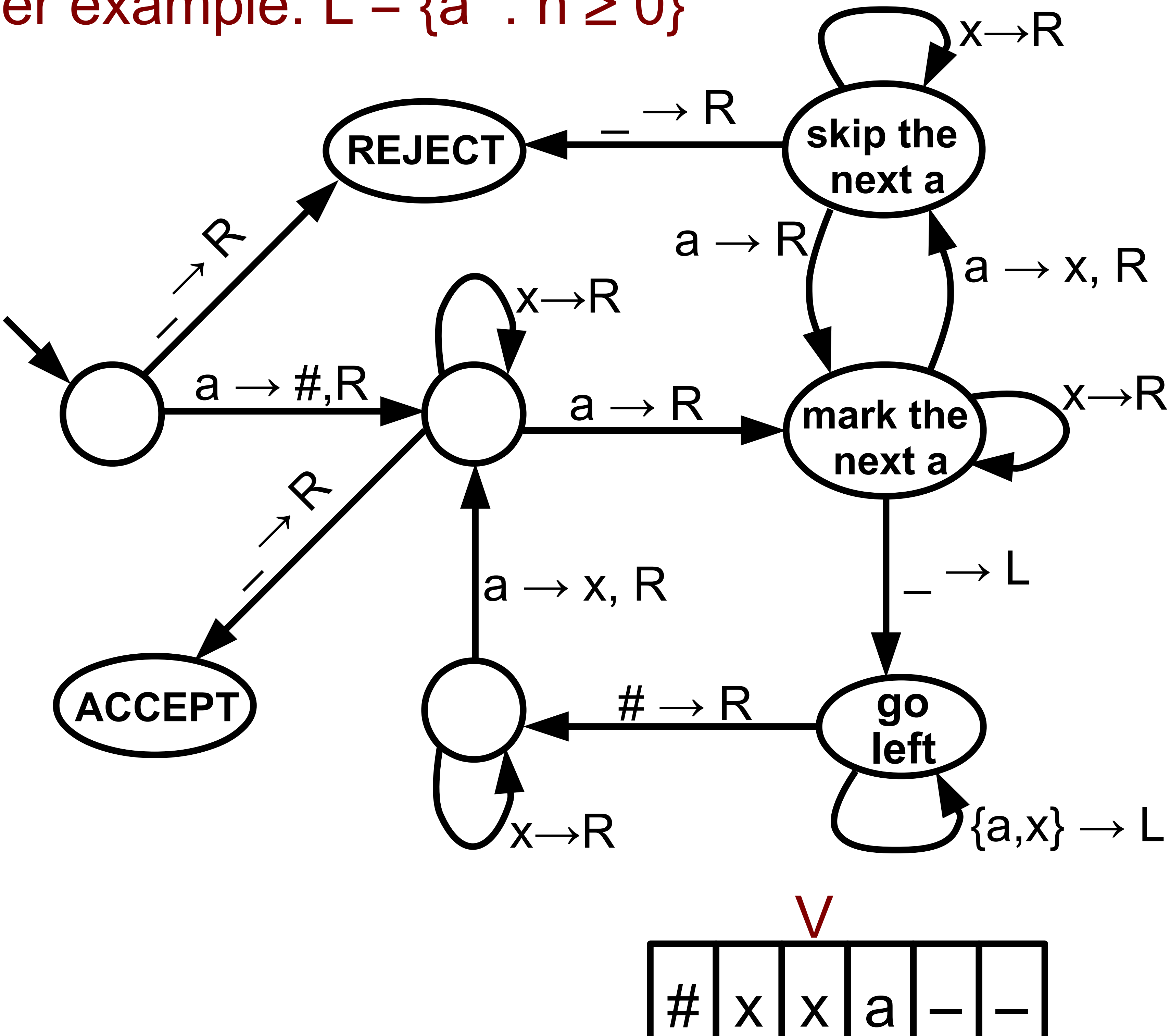




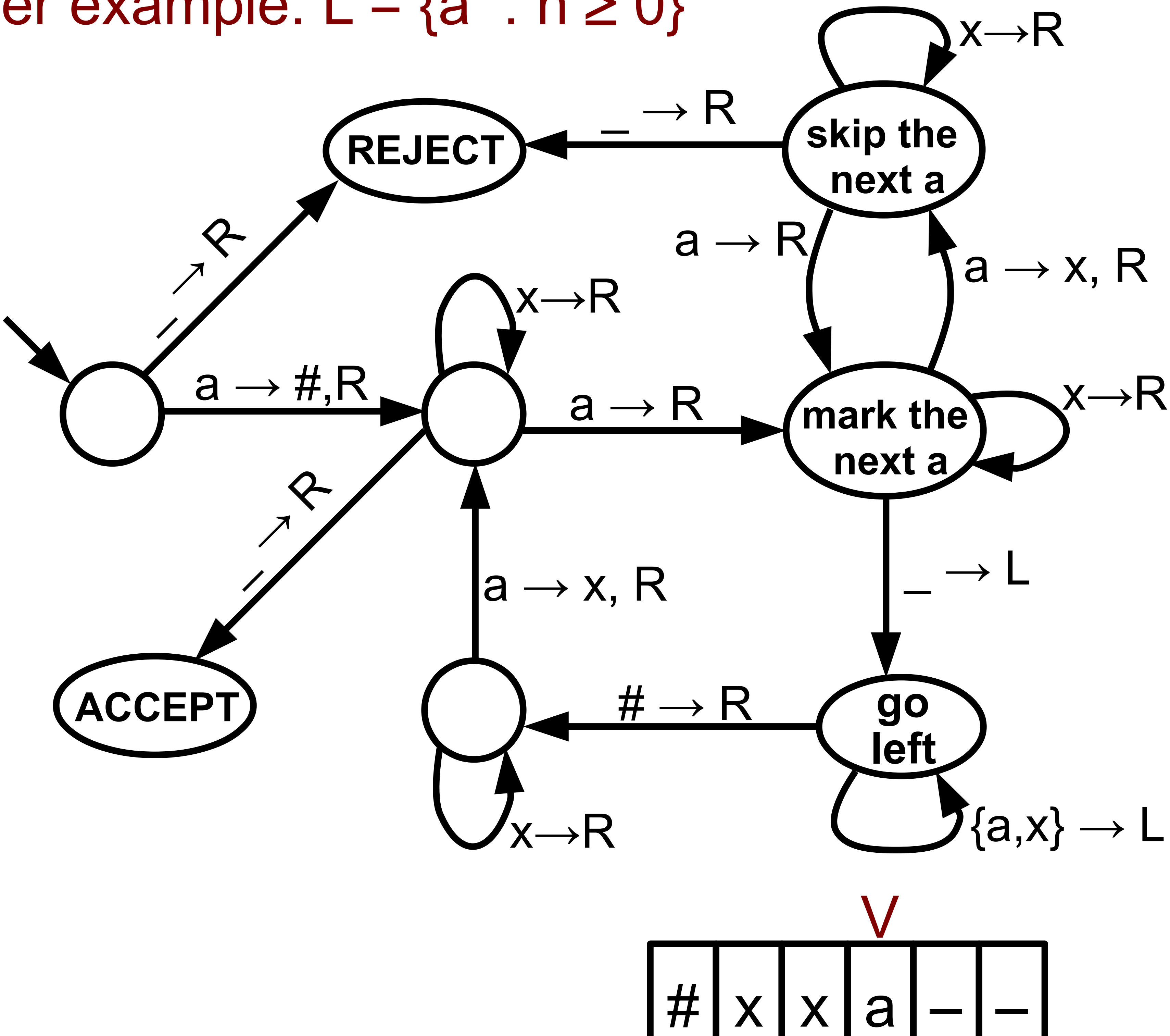
Another example: L = $\{a^{2^n} : n \geq 0\}$



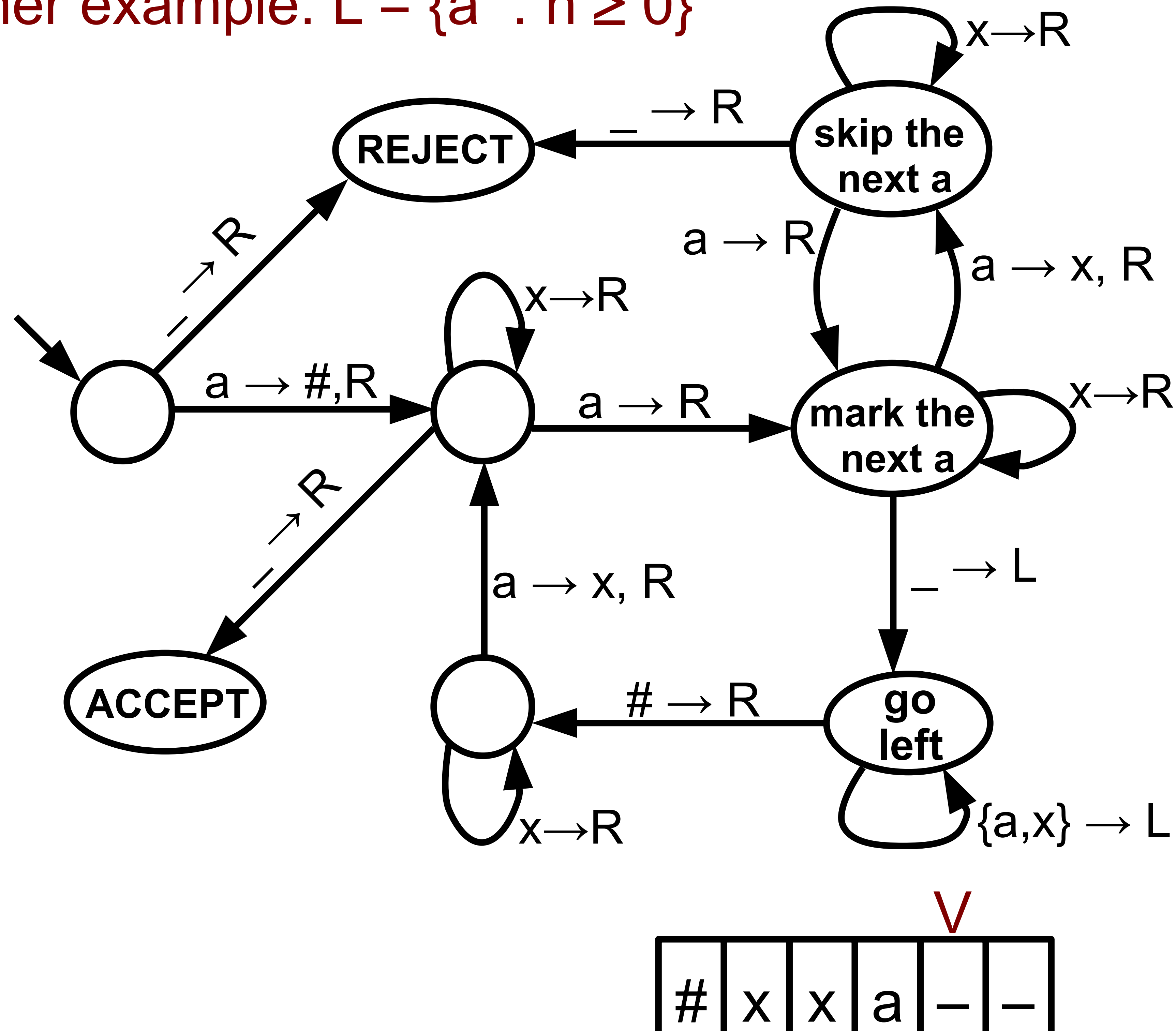
Another example: $L = \{a^{2^n} : n \geq 0\}$



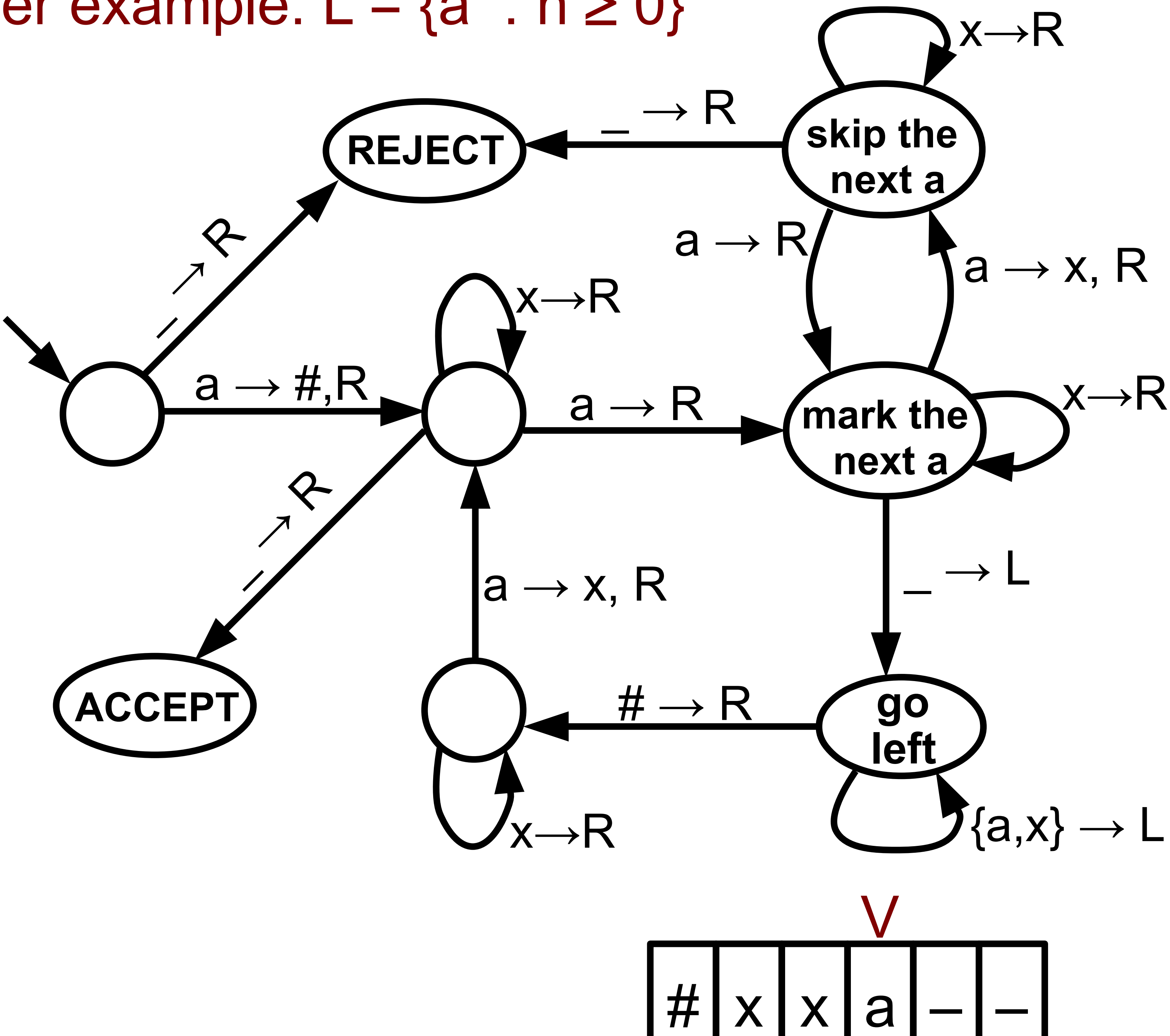
Another example: $L = \{a^{2^n} : n \geq 0\}$



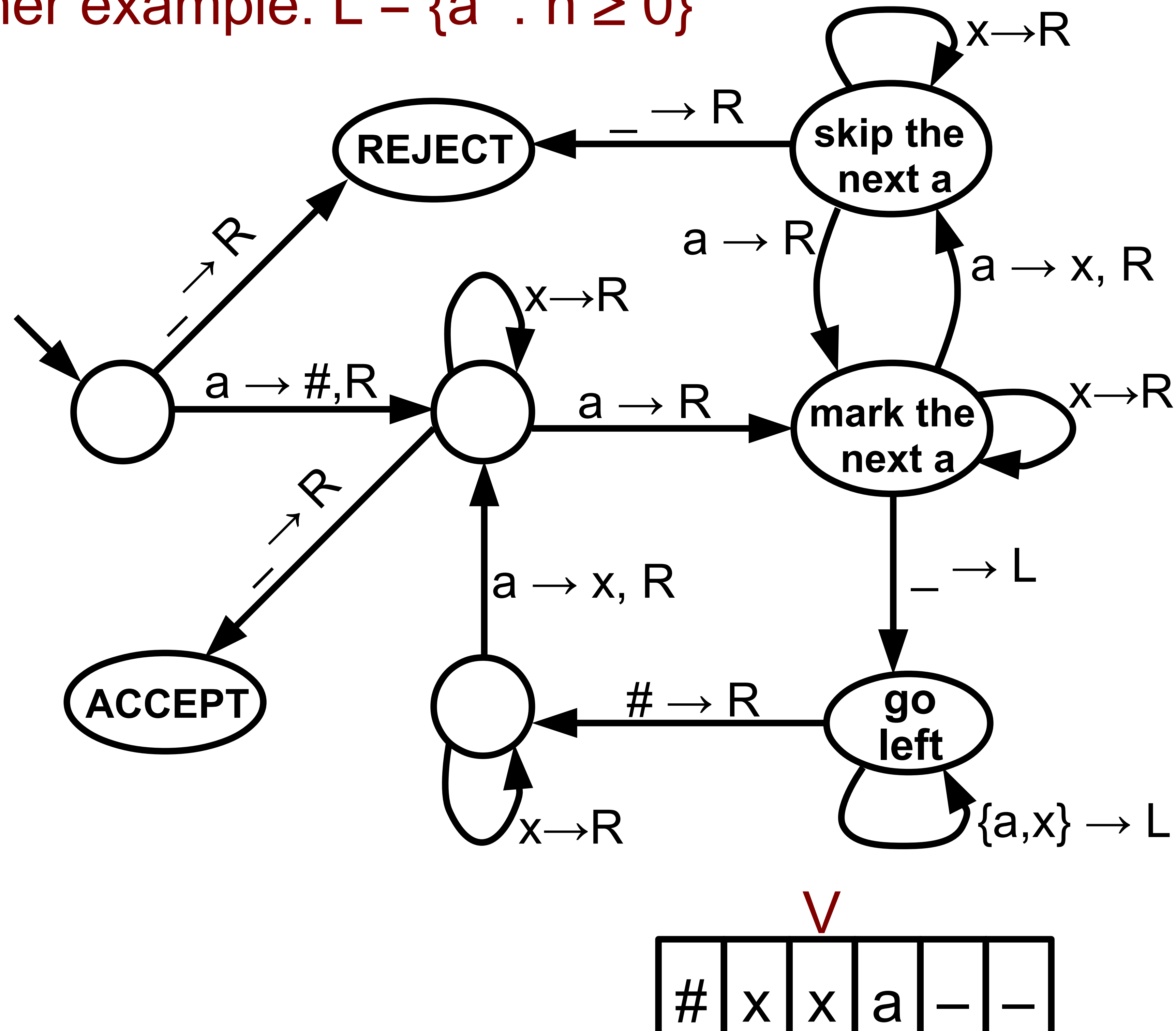
Another example: $L = \{a^{2^n} : n \geq 0\}$



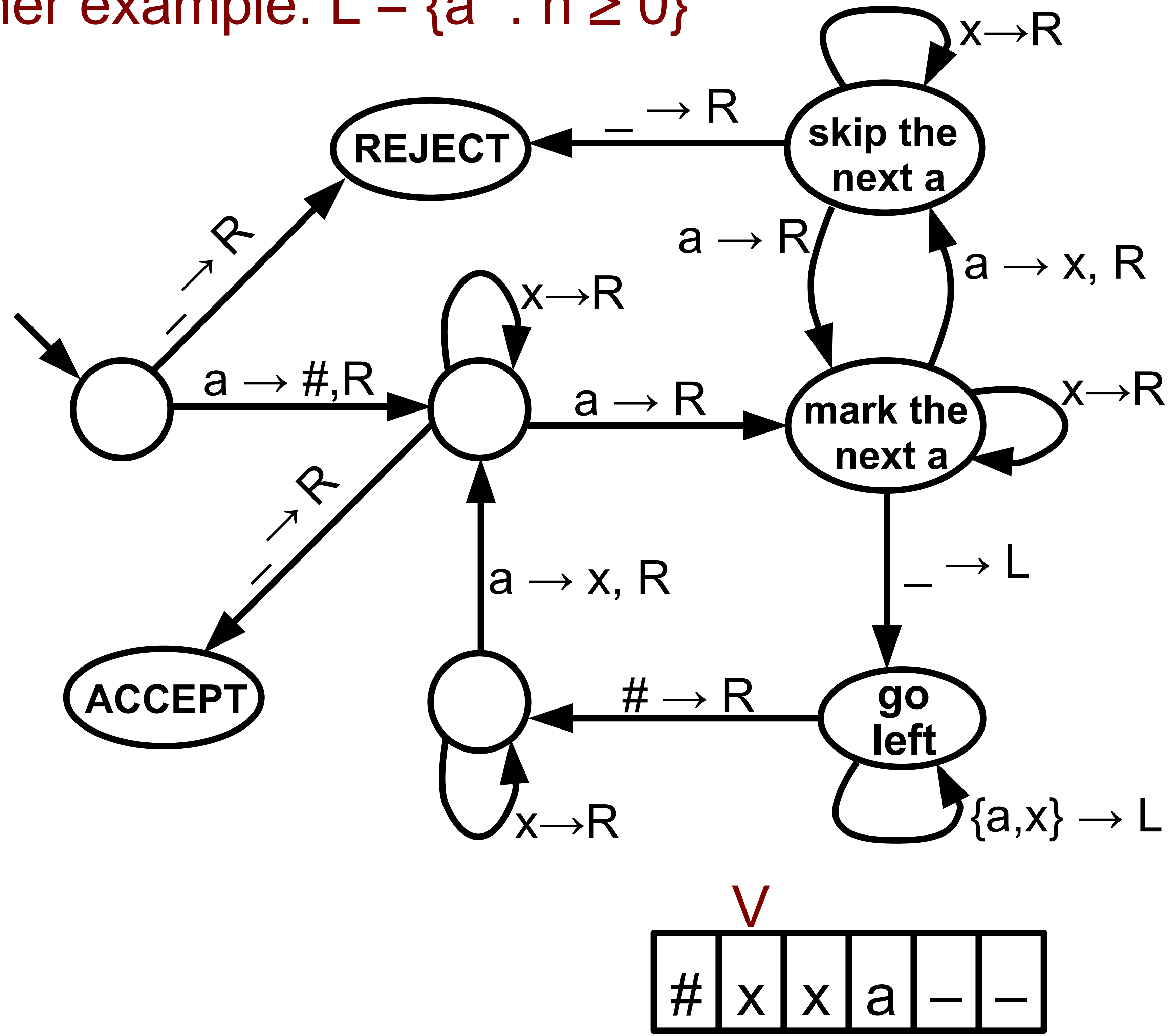
Another example: $L = \{a^{2^n} : n \geq 0\}$



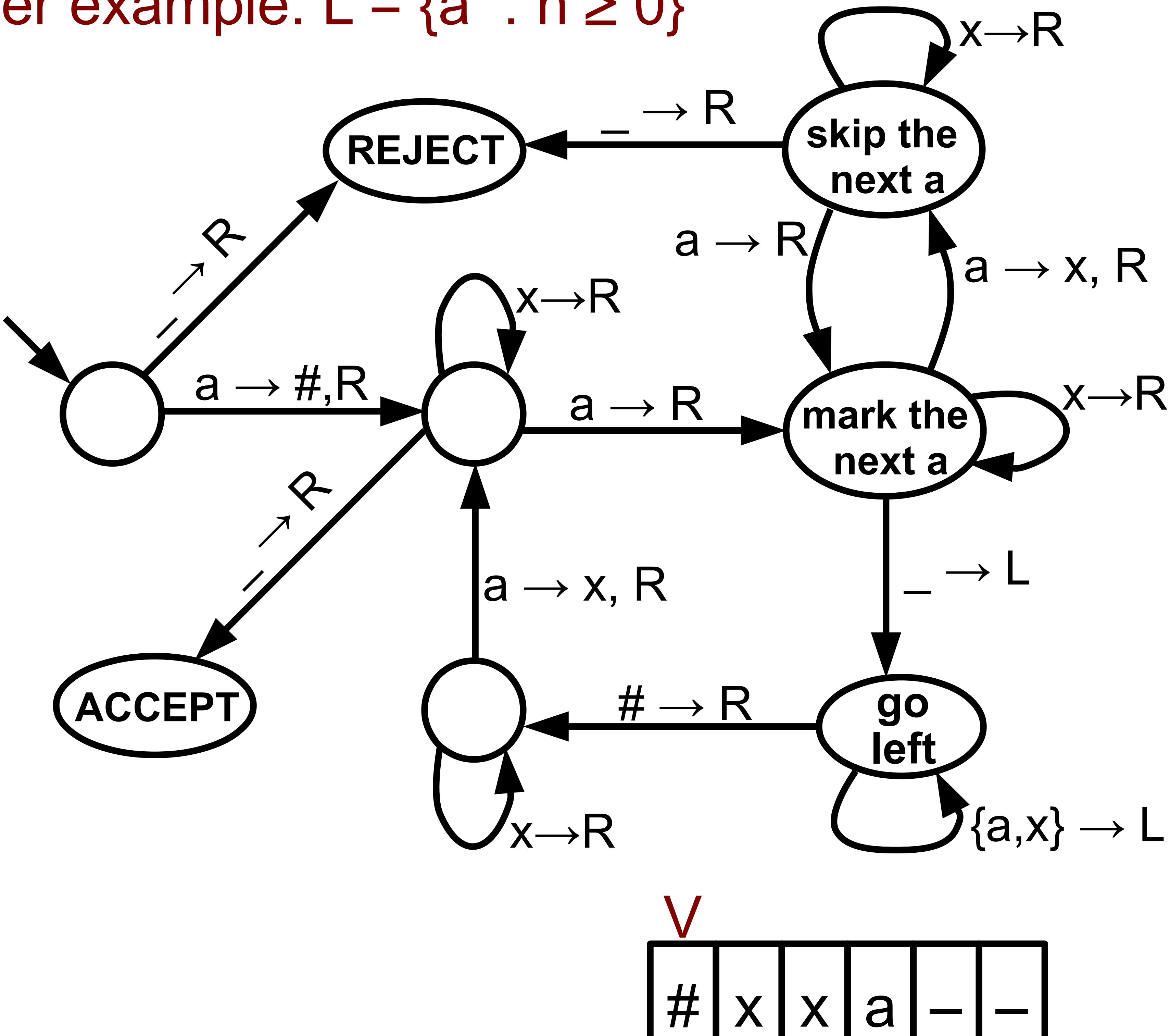
Another example: $L = \{a^{2^n} : n \geq 0\}$



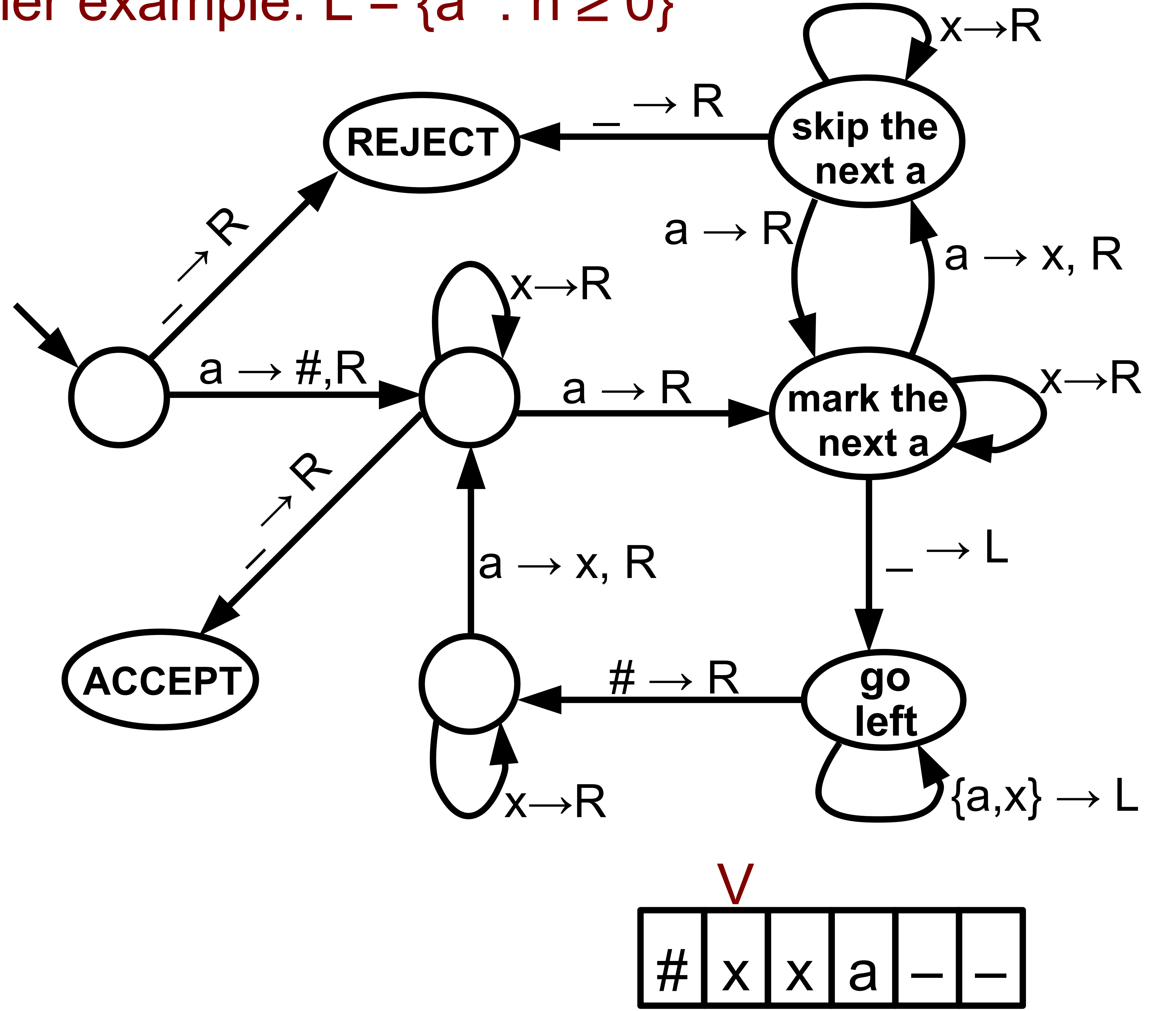
Another example: $L = \{a^{2^n} : n \geq 0\}$



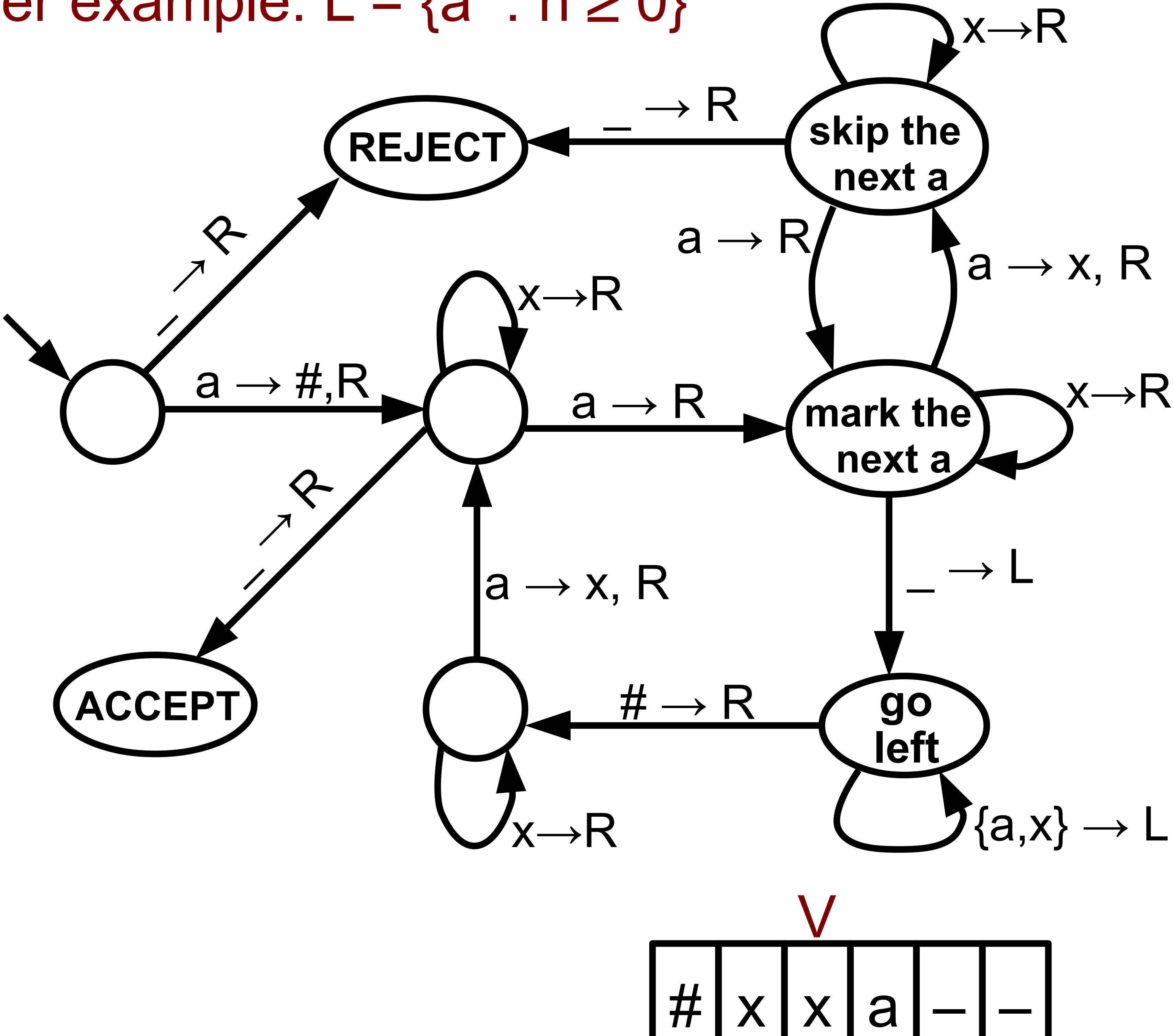
Another example: $L = \{a^{2^n} : n \geq 0\}$



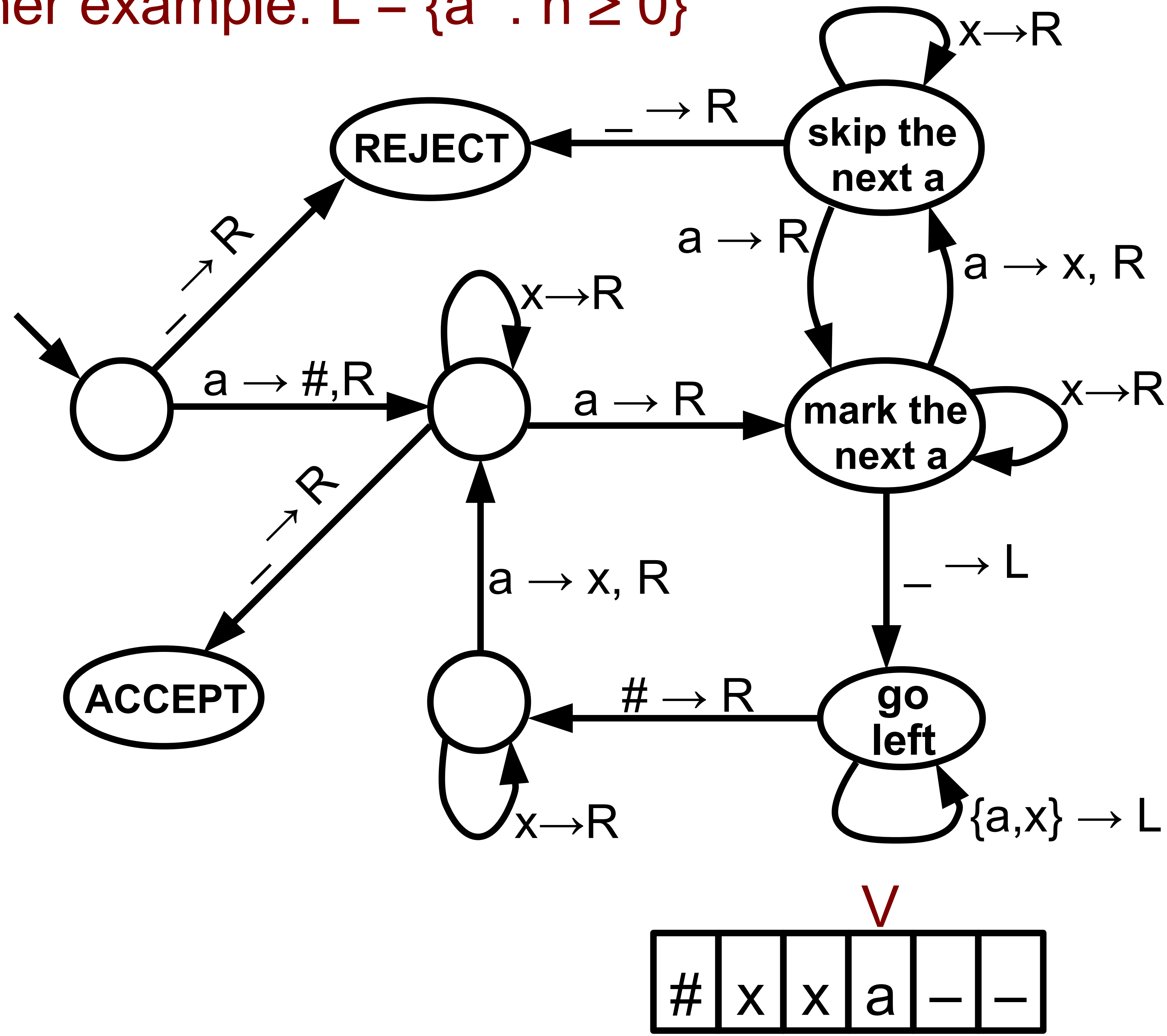
Another example: $L = \{a^{2^n} : n \geq 0\}$



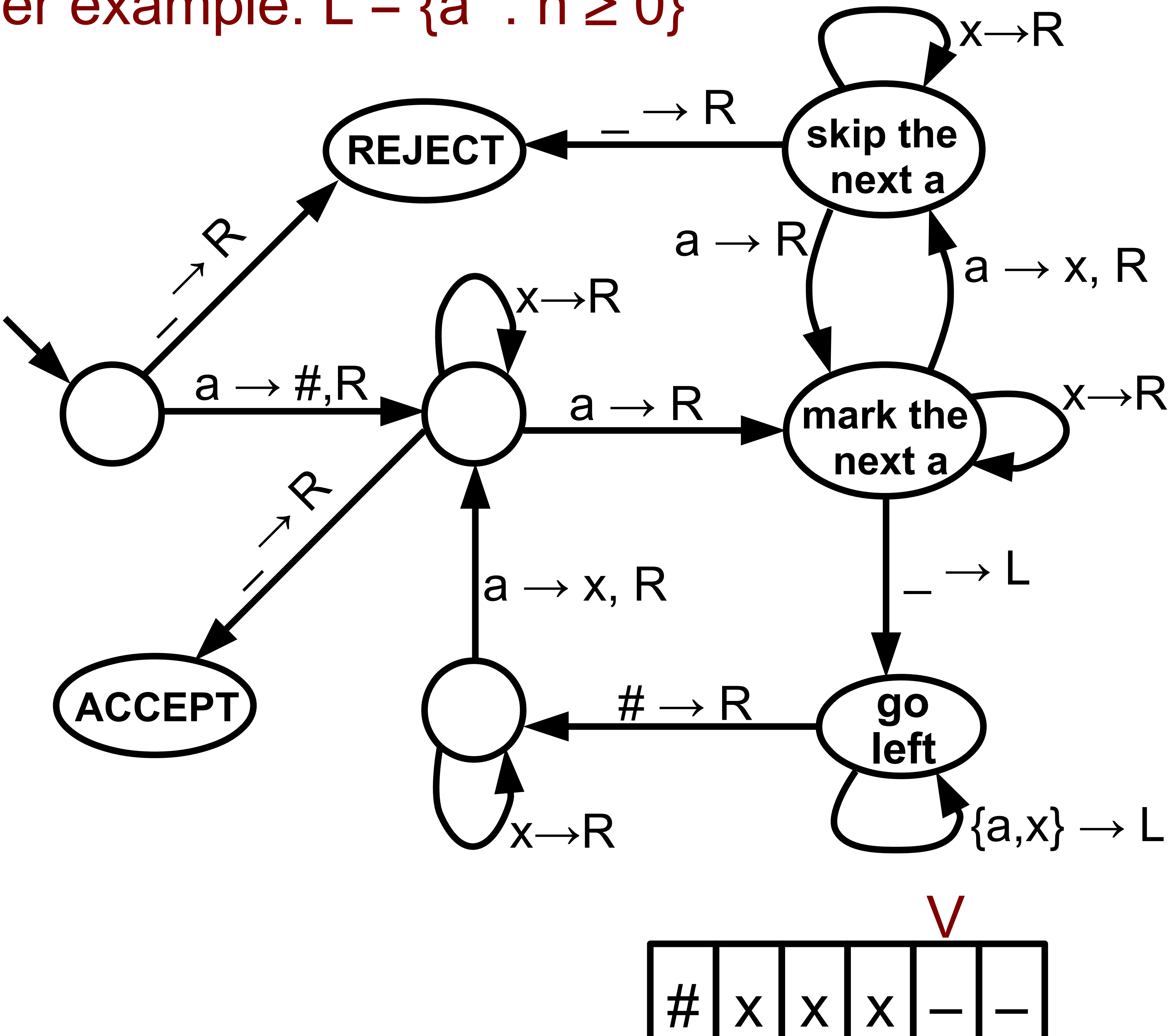
Another example: $L = \{a^{2^n} : n \geq 0\}$



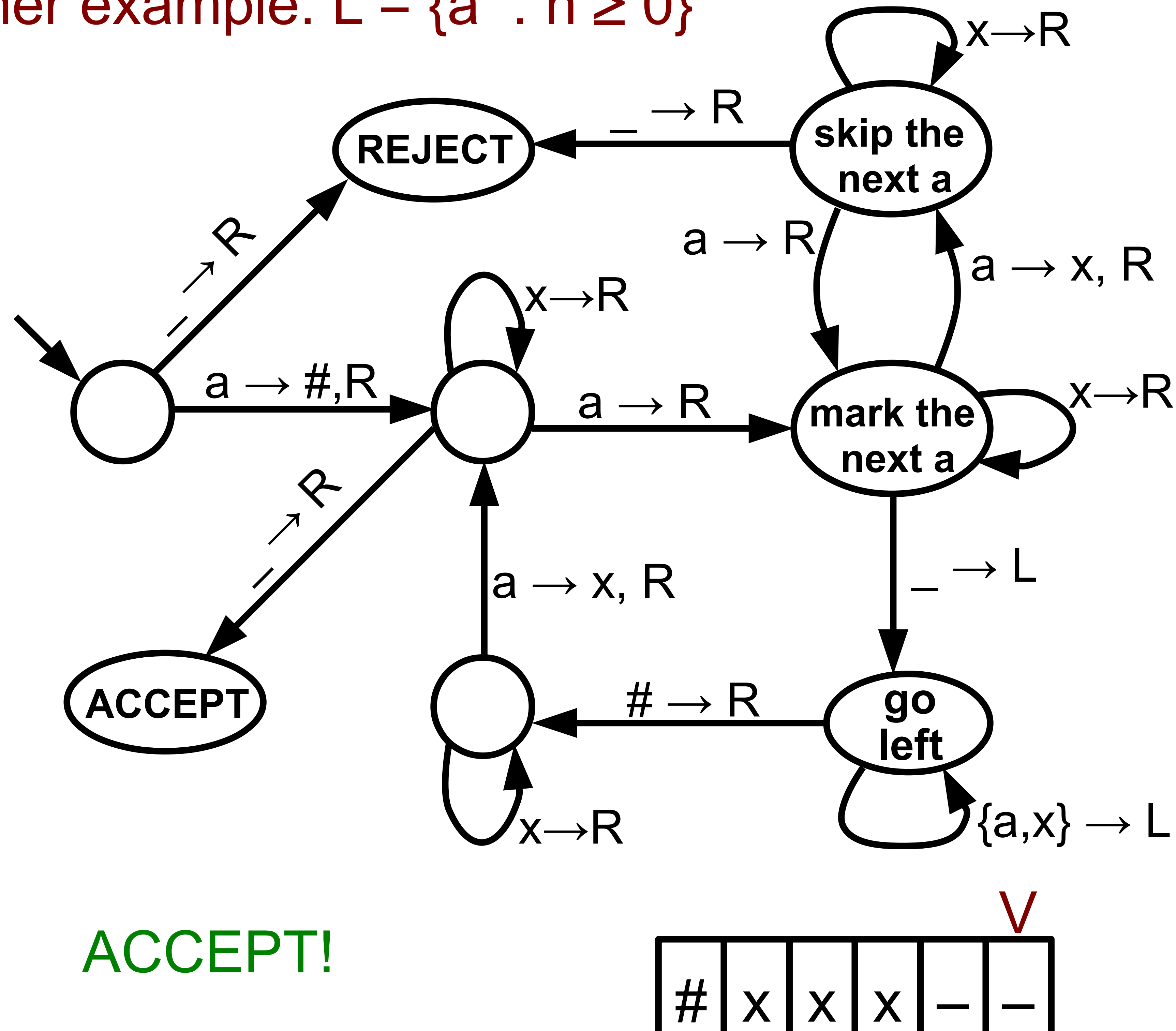
Another example: $L = \{a^{2^n} : n \geq 0\}$



Another example: $L = \{a^{2^n} : n \geq 0\}$



Another example: L = $\{a^{2^n} : n \geq 0\}$



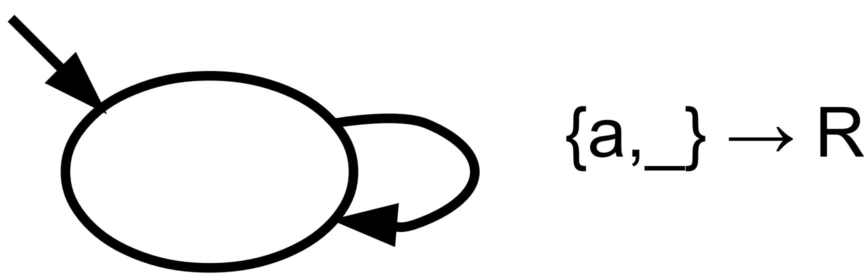
ACCEPT!

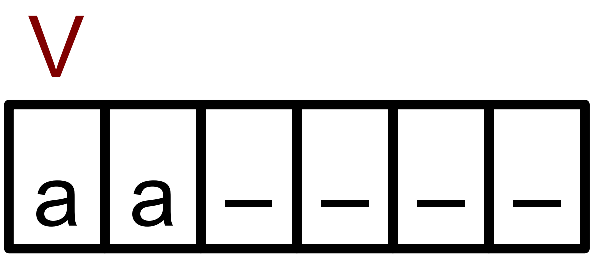
Unlike DFA and PDA,

TM computation may never halt. That is, it may continue forever without entering accept/reject states

This is when your computer "freezes"

Example



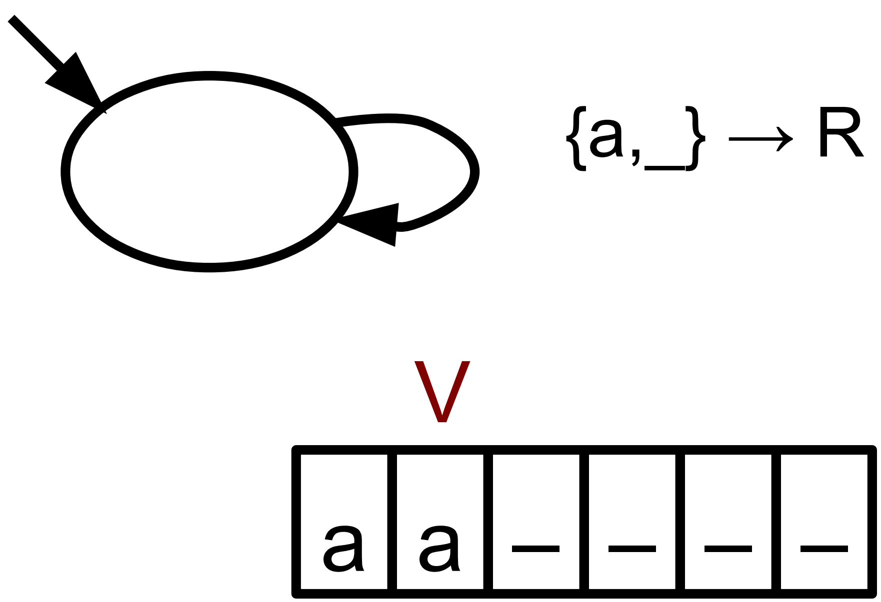


Unlike DFA and PDA,

TM computation may never halt. That is, it may continue forever without entering accept/reject states

This is when your computer "freezes"

Example

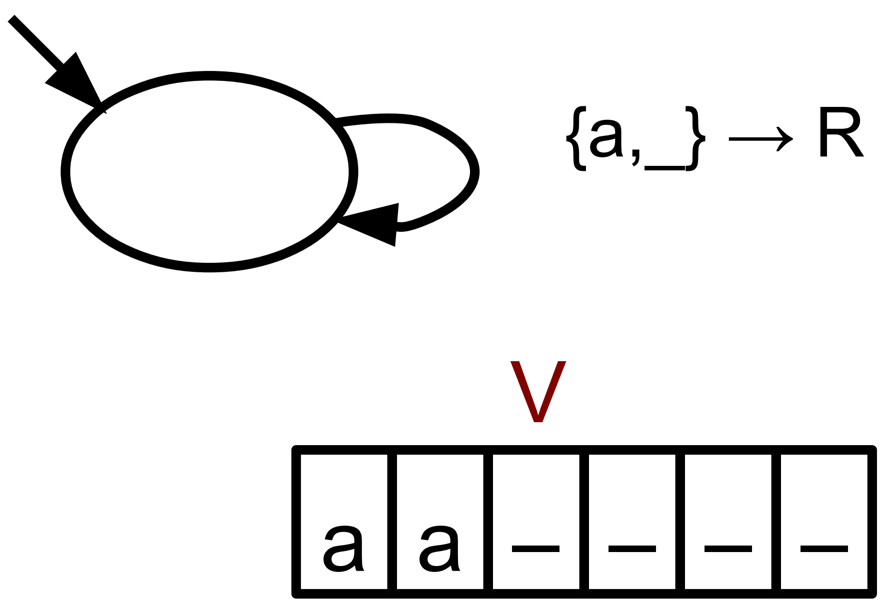


Unlike DFA and PDA,

TM computation may never halt. That is, it may continue forever without entering accept/reject states

This is when your computer "freezes"

Example

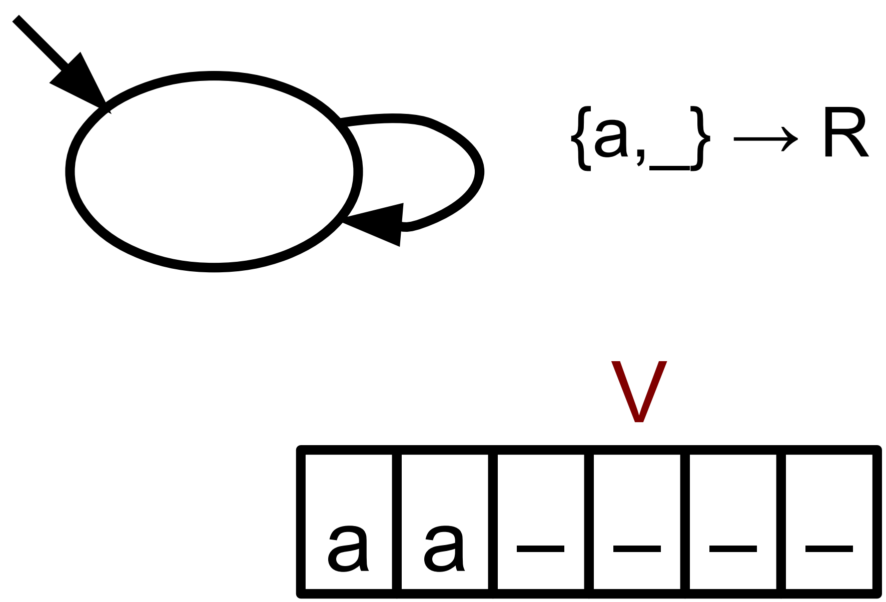


Unlike DFA and PDA,

TM computation may never halt. That is, it may continue forever without entering accept/reject states

This is when your computer "freezes"

Example



And so on!

- Definition: A Turing Machine TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

- Q is a finite, non-empty set of states
- Σ is the input alphabet. Blank symbol $\_ \notin \Sigma$
- Γ is the tape alphabet, $\Sigma \subseteq \Gamma$ and $\_ \in \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state; $q_{accept} \neq q_{reject}$

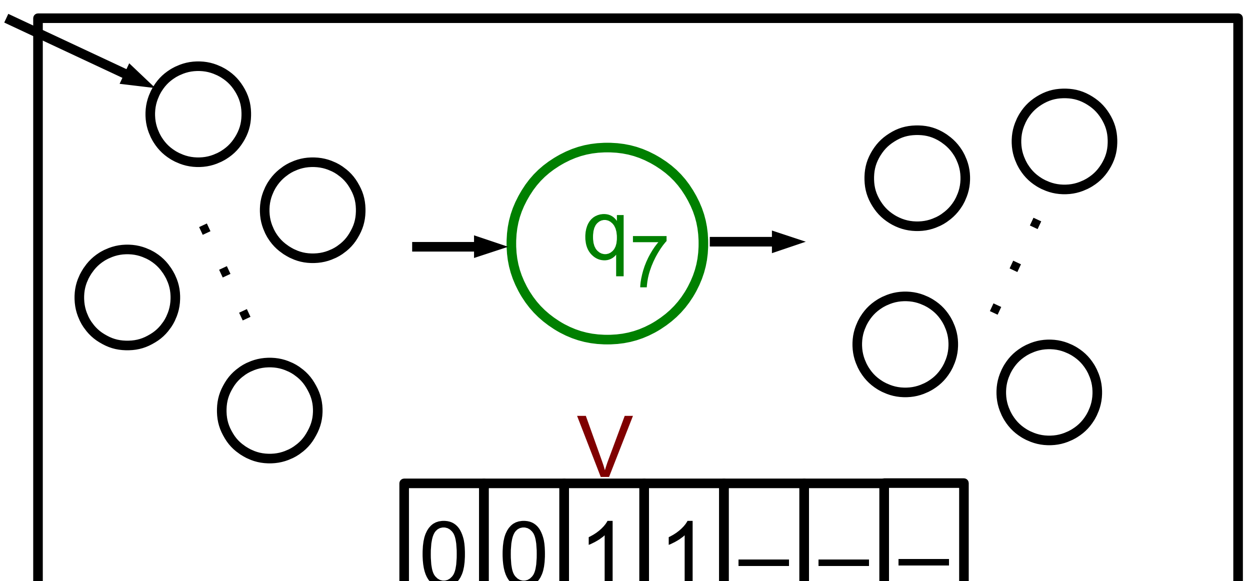
- Definition: A configuration of a TM specifies contents of tape, state, head location

It is written as u q v where $q \in Q$, $u, v \in \Gamma^*$

Meaning: 1) TM in state q

2) head is on first symbol of v.

3) Tape contains uv, blanks not shown



⇔ ?

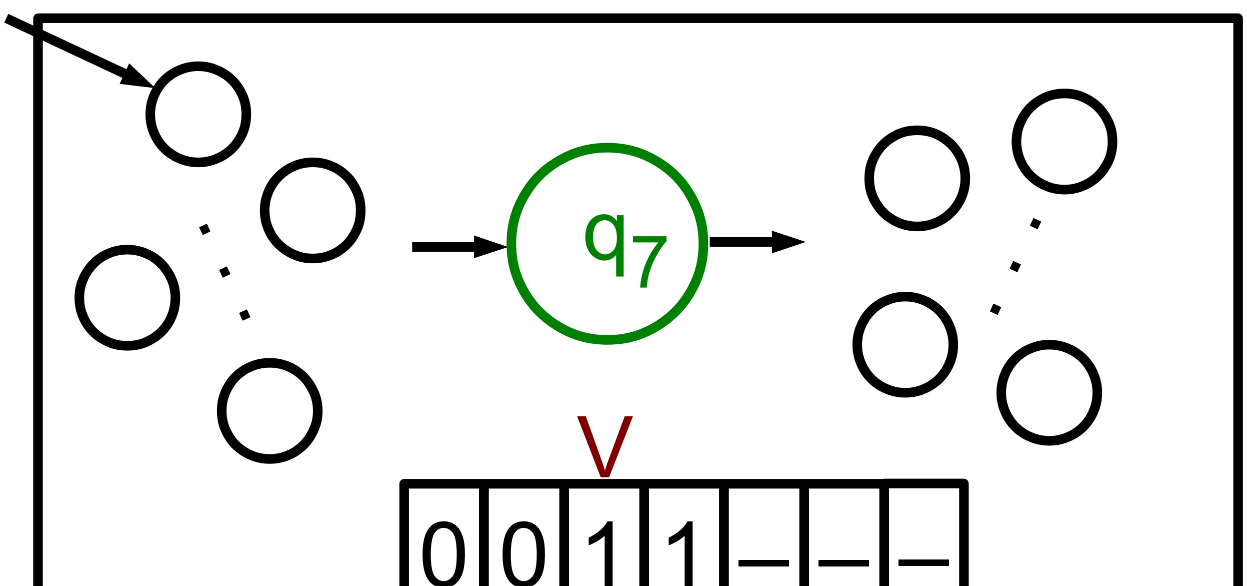
- Definition: A configuration of a TM specifies contents of tape, state, head location

It is written as u q v where $q \in Q$, $u, v \in \Gamma^*$

Meaning: 1) TM in state q

2) head is on first symbol of v.

3) Tape contains uv, blanks not shown



$\Leftrightarrow$ $00q_7 11$

- Definition: A configuration C yields a configuration C' if TM goes from C to C' in one step:

- u a q b v yields u q' a c v if δ (q,b) = (q',c,L)
- u a q b v yields u a c q' v if δ (q,b) = (q',c,R)
- u a q is treated like u a q _
- q b v yields q' c v if δ (q,b) = (q', c, L)
- q b v yields c q' v if δ (q,b) = (q', c, R)

- Definition:

  Start configuration of TM on input w is q_0w

  Accept configuration: any configuration with q_{accept}

  Reject configuration: any configuration with q_{reject}

  Halt (stop) configur.: Accept U Reject configur.

- Definition: TM M accepts (rejects, halts on) input w if $\exists$ configurations $C_1, C_2, ..., C_k$:

  C_1 is start configuration

  C_i yields C_{i+1} $\forall i < k$

  C_k is accept (reject, halt) configuration

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

q_1 aaa

#a q_2 aa

Example: L = {a^{2^n} : n ≥ 0}

q_0 aaaa

q_1 aaa

#a q_2 aa

#ax q_3 a

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

Example: L = $\{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

q_1 aaa

#a q_2 aa

#ax q_3 a

#axa q_2

#ax q_4 a

#a q_4 xa

q_4 axa

q_4 #axa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

q_1 aaa

#a q_2 aa

#ax q_3 a

#axa q_2

#ax q_4 a

#a q_4 xa

q_4 axa

q_4 #axa

q_5 axa

#x q_1 xa

#xx q_1 a

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

\#x q_4 xa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

\#x q_4 xa

\# q_4 xxa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

\#x q_4 xa

\# q_4 xxa

q_4 #xxa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

\#x q_4 xa

\# q_4 xxa

q_4 #xxa

\# q_5 xxa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

\#x q_4 xa

\# q_4 xxa

q_4 #xxa

\# q_5 xxa

\#x q_5 xa

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

\#x q_4 xa

\# q_4 xxa

q_4 #xxa

\# q_5 xxa

\#x q_5 xa

\#xx q_5 a

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa

\# q_1 aaa

\#a q_2 aa

\#ax q_3 a

\#axa q_2

\#ax q_4 a

\#a q_4 xa

\# q_4 axa

q_4 #axa

\# q_5 axa

\#x q_1 xa

\#xx q_1 a

\#xxa q_2

\#xx q_4 a

\#x q_4 xa

\# q_4 xxa

q_4 #xxa

\# q_5 xxa

\#x q_5 xa

\#xx q_5 a

\#xxx q_1

Example: $L = \{a^{2^n} : n \geq 0\}$

q_0 aaaa
q_1 aaa
#a q_2 aa
#ax q_3 a
#axa q_2
#ax q_4 a
#a q_4 xa
q_4 axa
q_4 #axa
q_5 axa
#x q_1 xa
#xx q_1 a
#xxa q_2
#xx q_4 a
#x q_4 xa
q_4 xxa
q_4 #xxa
q_5 xxa
#x q_5 xa
#xx q_5 a
#xxx q_1
#xxx_ q_{ACCEPT}

- Definition: A language L is decidable

  if $\exists$ a TM M such that for every input w

  $w \in L \Rightarrow$ M accepts w

  $w \notin L \Rightarrow$ M rejects w

- Is this the same as

  $w \in L \Leftrightarrow$ M accepts w

  ???

- Definition: A language L is decidable
  if ∃ a TM M such that for every input w
  $w \in L \Rightarrow$ M accepts w
  $w \notin L \Rightarrow$ M rejects w

- This is NOT the same as
  $w \in L \Leftrightarrow$ M accepts w
  because M may LOOP FOREVER (freeze, crash,...)

- We ask something more: TM halts on every input
  Such a TM is called a decider

- Definition: The language of TM M is
  $L(M) = \{w : M \text{ accepts } w\}$

- Recall this means $w \in L(M) \Leftrightarrow$ M accepts w

- Definition: A language L is recognizable if $\exists$ TM M :
  $L = L(M)$

- However we are more interested in decidable L

- So far, DFA, CFG, TM recognize languages

- Since TM can write on tape, they can also compute functions

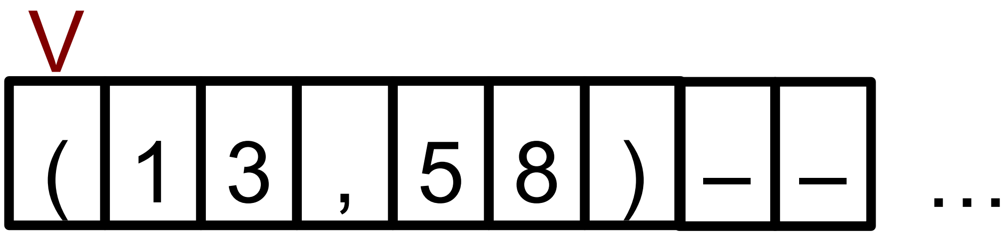
- Definition: A function $f : \Sigma^* \rightarrow \Sigma^*$ is computable if $\exists$ a TM M such that on every input $w \in \Sigma^*$ TM halts with f(w) on the tape

- All common functions such as +, x, /, etc.
  are computable

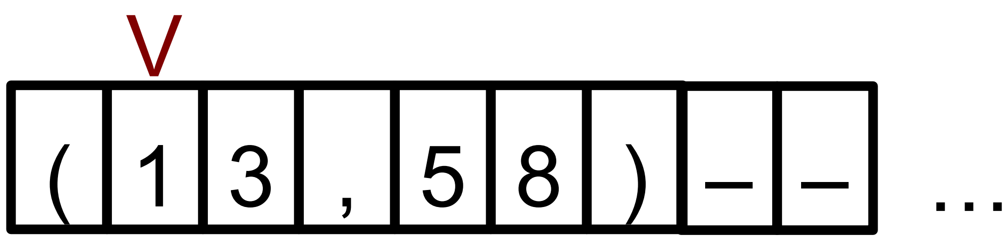
- Note: Consider for example $+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
  $+(2,9) = 11$
  $+(15,8) = 23$
  How to represent an input pair $(a,b) \in \mathbb{N} \times \mathbb{N}$?

- Any reasonable representation will do
  For example, use extra symbols for ( ) and ,

- Example: Computing + : ℕx ℕ→ ℕ

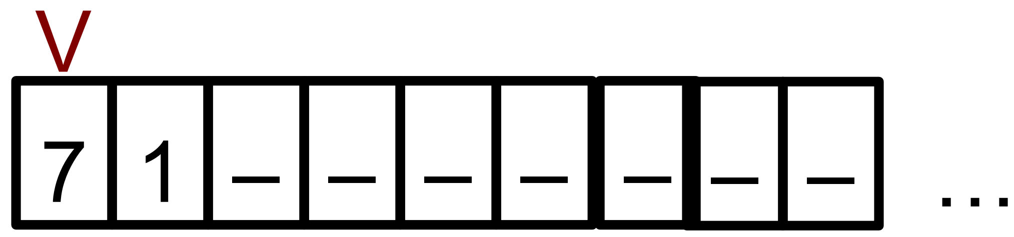


- Example: Computing $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$



Goes on for many steps until...

- Example: Computing + : ℕx ℕ→ ℕ



- How powerful are TM?
- One can show that regular, and context-free languages are decidable
- We saw TM decide some non-context free languages
- We saw TM also compute functions
- What else can a TM do?

- Suprisingly, TM are very powerful

- Pick your favorite programming language, say JAVA

- Theorem: For every language L :

  L decidable in JAVA ⇔ L decidable in TM

- Everything you program, you can do on a TM

  Program, algorithm, TM, etc. all mean the same!

- So why not use JAVA? Who cares about TM?

- JAVA and TM are equivalent. However,

- To design programs, JAVA is more convenient.
  Higher-level, shorter programs, human readable
  You do this in the Algorithms class

- To understand fundamental limits of computing TM
  is more convenient.
  Simpler description, configurations, head movement
  You do this in This class

- Main reason why TM better than JAVA for our aims

- TM computation is local:

  all action happens in tape symbols adjacent to head

- Not true for JAVA:

  no head, any tape (memory) symbol can change

- Locality is exploited in several results we will see

- Let us now make this more precise

- Fact: [Locality of TM computation]
  TM configuration C_i yields C_{i+1}
  ⇔ ∀ j , the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
  are consistent with TM transition function δ

- Example C_i = # a q_2 a a a b c x _ _
  C_{i+1} = # a x q_3 a a b c x _ _

Consistent: $\delta(q_2, \sigma) = (q, x, R)$ for some $\sigma \in \Gamma$, $q \in Q$

Note: σ = a here, but that is not among the 6 symbols

- Fact: [Locality of TM computation]
  TM configuration C_i yields C_{i+1}
  $\Leftrightarrow \forall j$, the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
  are consistent with TM transition function δ

- Example C_i = # a q_2 a a a b c x _ _
  C_{i+1} = # a x q_3 a a b c x _ _

Consistent: $\delta(q_2, a) = (q_3, x, R)$

Note: Only one choice here!

- Fact: [Locality of TM computation]
  TM configuration C_i yields C_{i+1}
  ⇔ ∀ j , the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
  are consistent with TM transition function δ

- Example C_i = # a q_2 a a a b c x _ _
  C_{i+1} = # a x q_3 a a b c x _ _

Consistent: $\delta(q_2, a) = (q_3, x, R)$

Note: Again only one choice here!

- Fact: [Locality of TM computation]
  TM configuration C_i yields C_{i+1}
  $\Leftrightarrow \forall j$, the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
  are consistent with TM transition function δ

- Example C_i = # a q_2 a a a b c x _ _
  C_{i+1} = # a x q_3 a a b c x _ _

Consistent: $\delta(q, a) = (q_3, \sigma , R)$ for some $q \in Q$, $\sigma \in \Gamma$

Note: $q = q_2$, but that is not among the 6 symbols

- Fact: [Locality of TM computation]
  TM configuration C_i yields C_{i+1}
  $\Leftrightarrow \forall j$, the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
  are consistent with TM transition function δ

- Example

C_i =	#	a	q_2	a	a	a	b	c	x	_	_
C_{i+1} =	#	a	x	q_3	a	a	b	c	x	_	_

Consistent: $\forall j$, hence C_i yields C_{i+1}

- **Fact**: [Locality of TM computation]
  TM configuration C_i yields C_{i+1}
  $\Leftrightarrow \forall j$, the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
  are consistent with TM transition function δ

- **Example** $C_i \;\; = \# \; a \; q_2 \; a \; a \; a \; b \; c \; x \; \_ \; \_$
  $C_{i+1} = \# \; a \; q_2 \; q_3 \; a \; a \; b \; a \; x \; \_ \; \_$

- **Not consistent**

- Is there anything beyond JAVA / TM?

- Church-Turing Thesis:
  Anything that is “effectively computable”
  is computable on a TM

- This is not a theorem. It is the belief that every computational model humans may ever consider (DNA computing, quantum computing, etc.) will still be equivalent to TM

- So far, simple-looking languages like $\{0^n 1^n : n \geq 0\}$, $\{ w : w \in \{0,1\}^* \}$, $\{a^i b^j c^k : i \leq j \leq j\}$

- Next: { D : D is a DFA and .... }

  { (M,w) : M is a TM and ... }

  { G : G is a graph and … }

- How to represent D, (M,w), G is not important

  Any reasonable representation will do!

- Example, use formal definitions over $\Sigma = \{a,b,c,....\}$

- Is there anything a TM cannot do?

- Definition:ATM = {(M,w) : M is a TM and M accepts w}

- We are going to prove ATM undecidable:

- Interpretation: Your friend comes to you with a piece a code M and some input w and says: M accepts w!

- Nobody can tell! …can't you just run M on w?

- Is there anything a TM cannot do?

- Definition:ATM = {(M,w) : M is a TM and M accepts w}

- We are going to prove ATM undecidable:

- Interpretation: Your friend comes to you with a piece a code M and some input w and says: M accepts w!

- Nobody can tell!   …can't you just run M on w?
  NO. M on w may never halt. But a decider must halt!

- Theorem: ATM= {(M,w) : M is a TM and M accepts w} is undecidable
- Idea: Proof by contradiction

- Theorem: ATM= {(M,w) : M is a TM and M accepts w} is undecidable
- Idea: Proof by contradiction

(1) We assume we have a decider D for ATM

(2) Using D, we derive a logical contradiction.

How ?

(3) We conclude that assumption (1) is false,

D cannot exist, and so ATM is undecidable

- Theorem: ATM= {(M,w) : M is a TM and M accepts w} is undecidable
- Idea: Proof by contradiction

(1) We assume we have a decider D for ATM

(2) Using D, we derive a logical contradiction.

Construct another decider D'

Show an input Y that D' neither accepts nor rejects

So D' cannot be a decider. Contradiction

What is Y?

(3) We conclude that assumption (1) is false,

D cannot exist, and so ATM is undecidable

- Theorem: ATM= {(M,w) : M is a TM and M accepts w} is undecidable
- Idea: Proof by contradiction

(1) We assume we have a decider D for ATM

(2) Using D, we derive a logical contradiction.

Construct another decider D'

Show an input Y that D' neither accepts nor rejects

So D' cannot be a decider. Contradiction

Y is D' itself! We run D' on its source code

(3) We conclude that assumption (1) is false,

D cannot exist, and so ATM is undecidable

- Theorem: ATM= {(M,w) : M is a TM and M accepts w}

  is undecidable

- Proof

Assume D decides ATM

Build D' := "On input M: Run D(M,M).

If it accepts, REJECT

If if rejects, ACCEPT."

D' is a decider because ????

- Theorem: ATM= {(M,w) : M is a TM and M accepts w}
  is undecidable
- Proof

Assume D decides ATM

Build D' := "On input M: Run D(M,M).
If it accepts, REJECT
If if rejects, ACCEPT."

D' is a decider because D is. However:

D' accepts D' ⇨ ???

- Theorem: ATM= {(M,w) : M is a TM and M accepts w}
  is undecidable
- Proof

Assume D decides ATM

Build D' := “On input M: Run D(M,M).
If it accepts, REJECT
If if rejects, ACCEPT.”

D' is a decider because D is. However:

D' accepts D' ⇨ D(D',D') rejects ⇨ ???

- Theorem: ATM= {(M,w) : M is a TM and M accepts w}
  is undecidable
- Proof

Assume D decides ATM

Build D' := "On input M: Run D(M,M).
If it accepts, REJECT
If if rejects, ACCEPT."

D' is a decider because D is. However:

D' accepts D' ⇨ D(D',D') rejects ⇨ D' rejects D'

D' rejects D' ⇨ ???

- Theorem: ATM= {(M,w) : M is a TM and M accepts w} is undecidable
- Proof

Assume D decides ATM

Build D' := "On input M: Run D(M,M).
If it accepts, REJECT
If if rejects, ACCEPT."

D' is a decider because D is. However:

D' accepts D' ⇨ D(D',D') rejects ⇨ D' rejects D'

D' rejects D' ⇨ D(D',D') accepts ⇨ ???

- Theorem: ATM= {(M,w) : M is a TM and M accepts w} is undecidable
- Proof

Assume D decides ATM

Build D' := "On input M: Run D(M,M).
If it accepts, REJECT
If if rejects, ACCEPT."

D' is a decider because D is. However:

D' accepts D' ⇨ D(D',D') rejects ⇨ D' rejects D'

D' rejects D' ⇨ D(D',D') accepts ⇨ D' accepts D'

A contradiction either way. So D cannot exist.

- Theorem: ATM= {(M,w) : M is a TM and M accepts w} is undecidable

- To prove some other language L undecidable, show

  L decidable ⇨ATM decidable

  This is sufficient by theorem above and contrapositive

- Such an implication is called a reduction of ATM to L

- Theorem: H = {(M,w) : M is a TM and M halts on w}
  is undecidable

- Interpretation: You have a piece of JAVA code that you are not sure if it works or if it is going to crash (crash = loop forever, freeze, get stuck, etc.)

- Nobody, by looking at the code, can tell!
  H is undecidable

- Theorem: H = {(M,w) : M is a TM and M halts on w} is undecidable
- Proof:

Suppose D decides H. We build D' that decides ATM

D' := "On input (M,w): Run D(M,w)

If it rejects, REJECT

Otherwise, run M on w until it halts

If M accepts, ACCEPT

If M rejects, REJECT."

- D' accepts (M,w) ⇔ M does not reject nor freeze on w

Done

- Theorem: L = {M : M is a TM and M accepts 001}
  is undecidable

- Proof:

First, for a TM M and an input w, we define
M'_w as := "On input x,

If $x \neq 001$, ACCEPT

Otherwise, run M on w,

if it accepts, ACCEPT

if it rejects, REJECT."

- Note: M'_w accepts 001 ⇔ ?

- Theorem: L = {M : M is a TM and M accepts 001}
  is undecidable
- Proof:

First, for a TM M and an input w, we define
M'_w as := "On input x,

If $x \neq 001$, ACCEPT

Otherwise, run M on w,

if it accepts, ACCEPT

if it rejects, REJECT."

- Note: M'_w accepts 001 ⇔ M accepts w

- Theorem: L = {M : M is a TM and M accepts 001}

  is undecidable

- Proof:

  Suppose D decides L. We build D' that decides ATM

  D' := "On input (M,w):

  Build M'_w and Run D(M'_w)

  If it accepts, ACCEPT

  If it rejects, REJECT."

- D' accepts (M,w) ⇔ ?

- Theorem: L = {M : M is a TM and M accepts 001}
  is undecidable
- Proof:

Suppose D decides L. We build D' that decides ATM

D' := "On input (M,w):

Build M'_w and Run D(M'_w)

If it accepts, ACCEPT

If it rejects, REJECT."

- D' accepts (M,w) ⇔ D accepts M'_w ⇔

?

- Theorem: L = {M : M is a TM and M accepts 001}
  is undecidable
- Proof:

  Suppose D decides L. We build D' that decides ATM

  D' := "On input (M,w):

  Build M'_w and Run D(M'_w)

  If it accepts, ACCEPT

  If it rejects, REJECT."

- D' accepts (M,w) ⇔ D accepts M'_w ⇔
  M'_w accepts 001 ⇔ ?

- Theorem: L = {M : M is a TM and M accepts 001}

  is undecidable

- Proof:

  Suppose D decides L. We build D' that decides ATM

  D' := "On input (M,w):

  Build M'_w and Run D(M'_w)

  If it accepts, ACCEPT

  If it rejects, REJECT."

- D' accepts (M,w) ⇔ D accepts M'_w ⇔

  M'_w accepts 001 ⇔ M accepts w Done

- What about

  {(M,w) : M is a TM and L(M) is finite

  L(M) = Ø

  |L(M)| = 56 or 127

  ....

- All undecidable

- TM are so powerful that you
  cannot decide anything about what they do!

- The undecidability proofs seen so far have not used anything specific about TM. Same proofs work with JAVA instead of TM

- TM are more useful than JAVA in pinpointing the simplest languages that are undecidable

- We now give a few examples

- $\{ D : D \text{ is a DFA and } L(D) = \Sigma^* \}$

  Decidable?

- $\{ G : G \text{ is CFG and } L(G) = \Sigma^* \}$

- { D : D is a DFA and L(D) = ∑ * }

  Decidable? Yes

  How?

- { G : G is CFG and L(G) = ∑ * }

- { D : D is a DFA and L(D) = Σ^* }

  Decidable? Yes

  How? Check if every state reachable from start

  state is accept

- { G : G is CFG and L(G) = Σ^* }

  Decidable?

- { D : D is a DFA and L(D) = ∑ * }

  Decidable? Yes

  How? Check if every state reachable from start

  state is accept

- { G : G is CFG and L(G) = ∑ * }

  Decidable? No

  Why?

- { D : D is a DFA and L(D) = ∑ * }

  Decidable? Yes

  How? Check if every state reachable from start
  state is accept

- { G : G is CFG and L(G) = ∑ * }

  Decidable? No

  Why? Can use it to decide ATM, which is undecidable
  Idea: simulate TM via CFG
  Would be much more complicated with JAVA

- Theorem: L:={G: G is CFG and L(G)=∑*} undecidable

- Proof: Suppose D decides L
- We construct D' that decides ATM:
- D' := "On input (M,w):

    construct CFG G : L(G) ≠ ∑* ⇔ M accepts w

    run D on G

    if it accepts, REJECT

    if it rejects, ACCEPT"

- Key of proof is construction of G

- Given (M,w) want G: $L(G) \neq \Sigma^*$ ⇔ M accepts w

- We construct G : L(G) = all strings that are NOT accepting computations of M on w

- Represent computation by sequence of configurations separated by #: $C_1\#C_2\#C_3$ ...

- Example: $q_0000101\#1q_300101\#10q_20101$

- Construct G: L(G) = all strings over $\Delta = \{\#\} \cup \Gamma \cup Q$ that are NOT accepting computations of M on w

- A string $C_1\#C_2\#C_3\#\ldots\#C_k$ is in L(G) $\Leftrightarrow$
  (a) C_1 is not the start configuration, or
  (b) C_k is not an accept configuration, or
  (c) $\exists i : C_i$ does not yield C_{i+1}

- We construct CFG for (a), (b), and (c) separately then use closure under U

- (a) CFG G_a : $L(G_a)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that C_1 is not the start configuration
- Recall start configuration is q_0w
- Consider Regular Expression R = $q_0 w \# \Delta^*$

  L(R) = strings starting with (start configuration)#
- not L(R) is regular, hence context-free
- All these transformations can be performed by TM

- (b) CFG G_b : $L(G_b)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that C_k is not an accept configuration

- Consider RE R = $\Delta^* \, q_{accept} \, (\Delta\text{-}\{\#\})^*$

  L(R) = strings where C_k contains accept state

- not L(R) is regular, hence context-free

- All these transformations can be performed by TM

- (c) CFG G_c : $L(G_c)$ = strings $C_1\#C_2\#C_3\#\ldots\#C_k$ such that $\exists\, i$: C_i does not yield C_{i+1}

- Here we: use power of CFG, and

  exploit locality of TM computation

- Technical detail: we show $\exists\, i$: C_i does not yield C_{i+1}^R

- Write TM computation as: $C_1\#C_2^R\#C_3\#C_4^R\#\ldots$

- (c) CFG G_c : $L(G_c)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that $\exists i$: C_i does not yield C_{i+1}^R

- Next is the idea; there are a few details to be filled in

- Construct G_c : $L(G_c) = \Delta^* \, abc \, (\Delta-\{\#\})^t \, \# \, (\Delta-\{\#\})^t \, fed \, \Delta^*$

  for any $t \geq 0$, any 6 symbols

a b c	
d e f	

  that are inconsistent with TM transition function δ

- Note: Essentially this is CFG for $w\#w^R$ seen before

- Recap:
- Theorem: L:={G: G is CFG and L(G)=$\sum^*$} undecidable
- Key of proof is, on input (M,w), construct CFG G : L(G) = all strings that are NOT accepting computations of M on w
- Use locality of TM computation (easier than JAVA)
- Conceptually simple, but a few details

- Theorem: ECF={(G,G'): G, G' CFG and L(G) = L(G') } undecidable

- Meaning: You think you have a 5-line grammar that is equivalent to another 5000-page grammar

- Nobody can tell if they are indeed equivalent

- Theorem: ECF={(G,G'): G, G' CFG and L(G) = L(G') }
  undecidable
- Proof: Suppose D decides ECF

We construct D' that decides {G: G CFG, L(G)=$\sum^*$ }

D' := "On input G:

Build CFG G' = $S \to \varepsilon \mid Sa \quad \forall a \in \sum$

Run D(G,G')

If it accepts, ACCEPT

If it rejects, REJECT."

- L(G') = $\sum^*$, so D' accepts G $\Leftrightarrow$ L(G) = L(G') = $\sum^*$

- Undecidability in logic

- Consider sentences over $\mathbb{N} = \{1,2,3,...\}$
  using variables x, y, z
  operations +, multiplication,
  equality =
  connectives $\wedge$ $\vee$
  quantifiers $\exists$, $\forall$

- Example: $\exists x > 1\ \exists y > 1 : 5039 = xy$
  Meaning: ?

- Undecidability in logic

- Consider sentences over $\mathbb{N} = \{1,2,3,\ldots\}$
  using variables x, y, z
  operations +, multiplication,
  equality =
  connectives $\wedge$ $\vee$
  quantifiers $\exists$, $\forall$

- Example: $\exists x > 1 \ \exists y > 1 : 5039 = xy$
  Meaning: 5039 is not prime (a false sentence)

- $\forall q\ \exists p > q$ not $(\exists x > 1\ \exists y > 1 : p = xy)$

  There are infinitely many primes

  Proved by Euclid ~ 2300 years ago

- $\forall a\ \forall b\ \forall c\ \forall n > 2,\ a^n + b^n \neq c^n$

  Fermat's last theorem, stated in 1637

  Proved by Andrew Wiles in 1995 (358 years later)

- $\forall q\ \exists p > q$ not $(\exists x > 1\ \exists y > 1 : p = xy \lor p+2 = xy)$

  Twin prime conjecture

- Theorem [Godel, Church]

  TRUTH = { S : S is a true sentence over $\mathbb{N}$}

  is undecidable

- Proof sketch:

  Given TM M and input w,
  build a formula $S_{M,w}$ such that:

  $S_{M,w}$ true ⇔ M accepts w

  use integers to encode configurations of TM

- Note: without multiplication, TRUTH is decidable

- Undecidability in mathematics

Polynomials: $p(x,y,z) = x^2 + 56\, y + 13xy^3z$

- $H10 = \{\, p(x_1, \ldots, x_n) : p(x_1, \ldots, x_n)$ is a polynomial and
  and $\exists\, a_1, \ldots, a_n \in \mathbb{N}$ such that $p(a_1, \ldots, a_n) = 0 \}$

- Hilbert asked for a “decider” for H10 in 1900

- Theorem [Matiyasevich, 1970] H10 is undecidable