Logică și structuri discrete

Mașini Turing. Calculabilitate

Casandra Holotescu

casandra@cs.upt.ro

https://tinyurl.com/lecturesLSD

Ce se poate și nu se poate calcula?

Dată fiind o problemă, se poate scrie un program care o rezolvă?

Teorema lui Cantor: *Nu există bijecție* de la X la $\mathcal{P}(X)$

$$|X| < |\mathcal{P}(X)|$$

Care e legătura?

Programe și probleme

Un *program* (în cod sursă) e un șir de caractere.
deși nu orice șir de caractere e un program valid

Notând cu *Progs* mulțimea programelor și Σ alfabetul caracterelor

$$Progs \subseteq \Sigma^*$$

O clasă de *probleme* (sunt multe altele):

Fiind dată o mulțime de șiruri $S = \{s_1, s_2, ..., s_n, ...\}$, și un șir w, aparține el mulțimii date, $w \in S$?

Orice mulțime de șiruri definește (cel puțin) o problemă

$$\mathcal{P}(\Sigma^*) \subseteq Probs$$

Teorema lui Cantor ne spune atunci:

$$|Progs| \leq |\Sigma^*| < |\mathcal{P}(\Sigma^*)| \leq |Probs|$$

Nu putem asocia deci fiecărei probleme un program!

Ce putem calcula?

DFA/NFA recunosc doar limbaje regulate

Într-un automat (DFA sau NFA) comportamentul e determinat complet de *stare* și *intrare*

Automatul "știe" doar starea în care se află:
are *memorie finită*

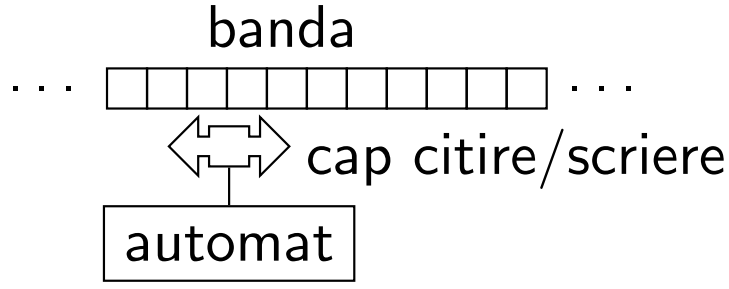
L=$\{a^n b^n | n \in \mathbb{N}\}$ nu e un limbaj regulat
ar trebui să *numărăm* câți a apar, verificăm să fie la fel de mulți b.
fără nicio limitare

$\Rightarrow$ pt. a recunoaște limbajul
avem nevoie de o structură cu *memorie nelimitată*

Conceptual, un calculator nu are nici el limită de memorie
(desi în realitate aceasta este, desigur, finită).

Mașina Turing = automat cu stări finite + memorie nelimitată

Mașina Turing

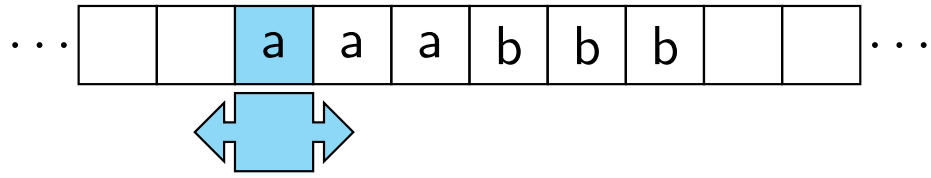


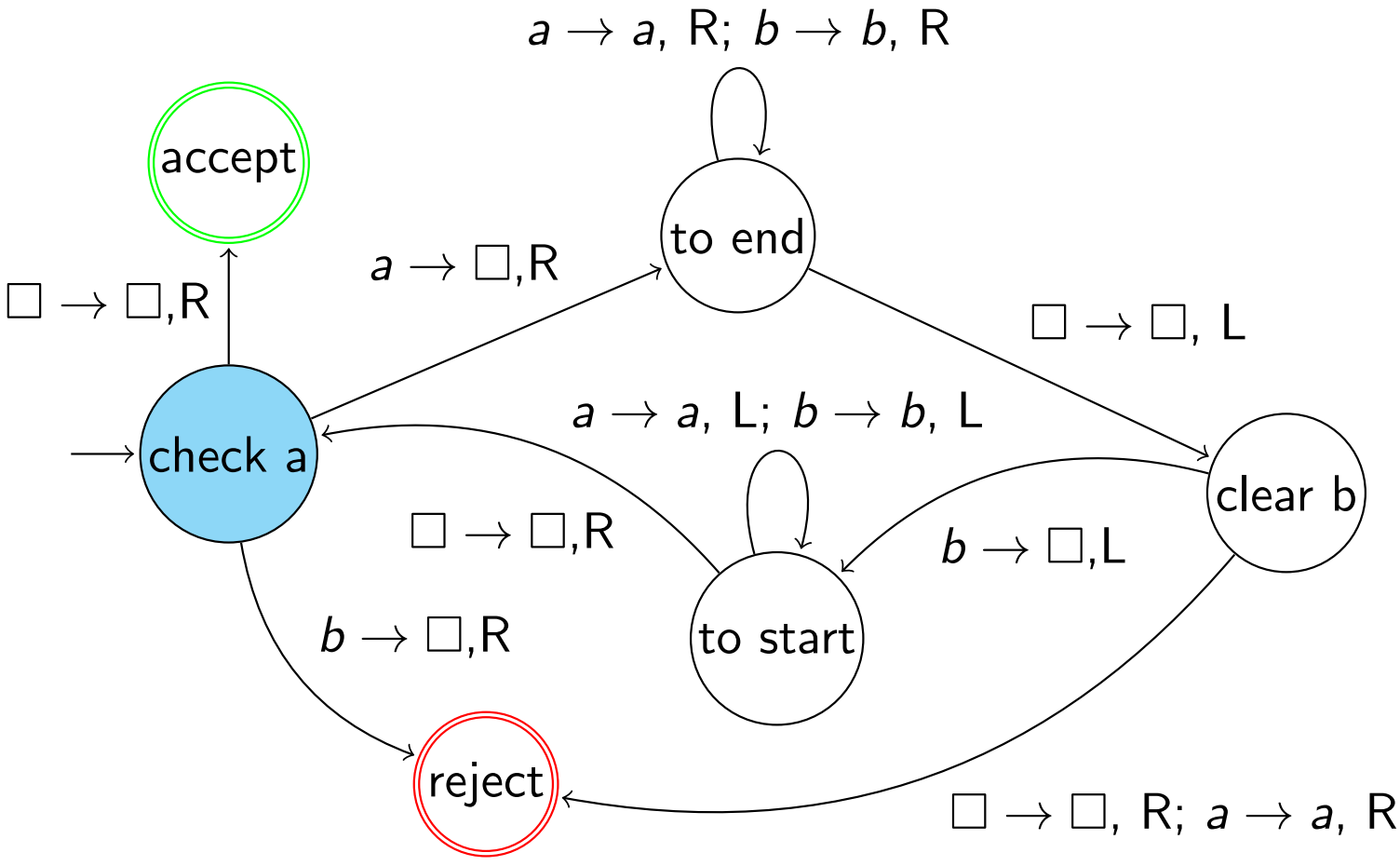
Mașina Turing e compusă din:

- un *automat cu stări finite*
- o *bandă* cu un număr infinit de *celule*
  fiecare celulă a benzii conține un *simbol*
  (banda poate fi infinită la unul/ambele capete, e echivalent)
- un *cap* de citire/scriere al simbolurilor de pe bandă
  (*controlat de automat*)

Automatul și conținutul benzii determină *împreună* comportamentul mașinii Turing.

Exemplu: mașina Turing pt. $L=\{a^n b^n | n \in \mathbb{N}\}$





Mașina Turing

Automatul și conținutul benzii determină comportamentul.

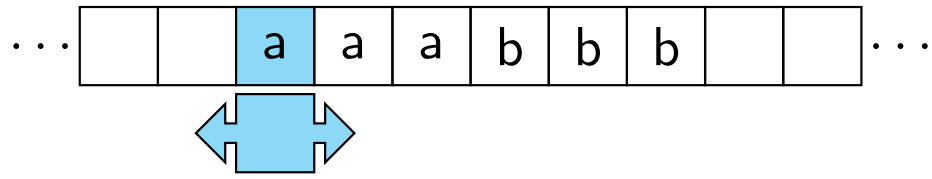
În funcție de

- *starea curentă* a automatului
- *simbolul citit* de pe bandă

Mașina efectuează următoarele acțiuni:

- trece în *starea următoare* (a automatului)
- *scrie* un (nou) *simbol* sub capul de citire/scriere
- *mută capul* de citire/scriere la stânga sau la dreapta

Mașina Turing



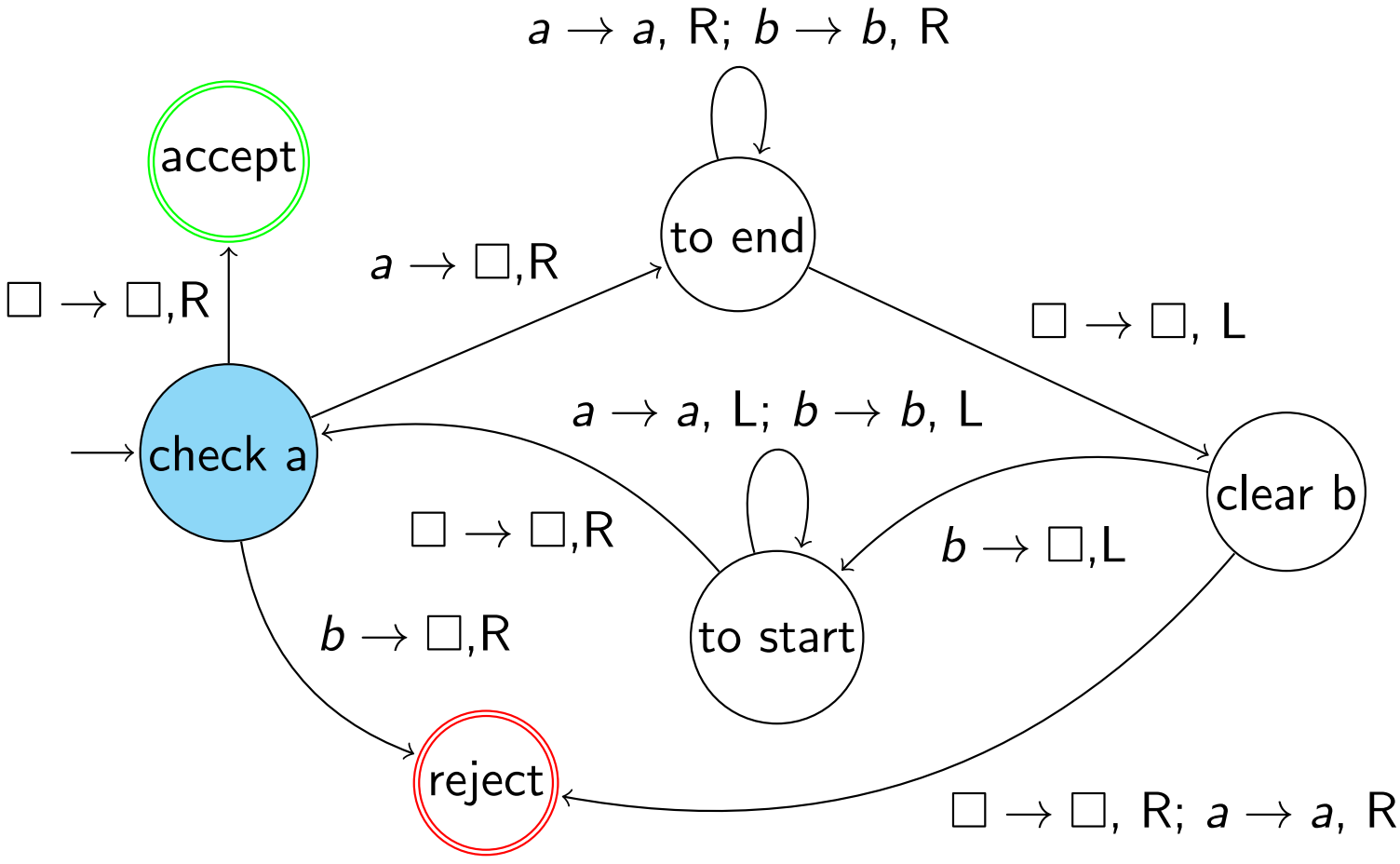
Inițial, banda conține un șir finit de *simboluri de intrare*,

capul de citire/scriere e pe *primul* simbol (cel mai din stânga);

restul celulelor conțin un *simbol special* ($\square$ = *vid* sau *blank*)

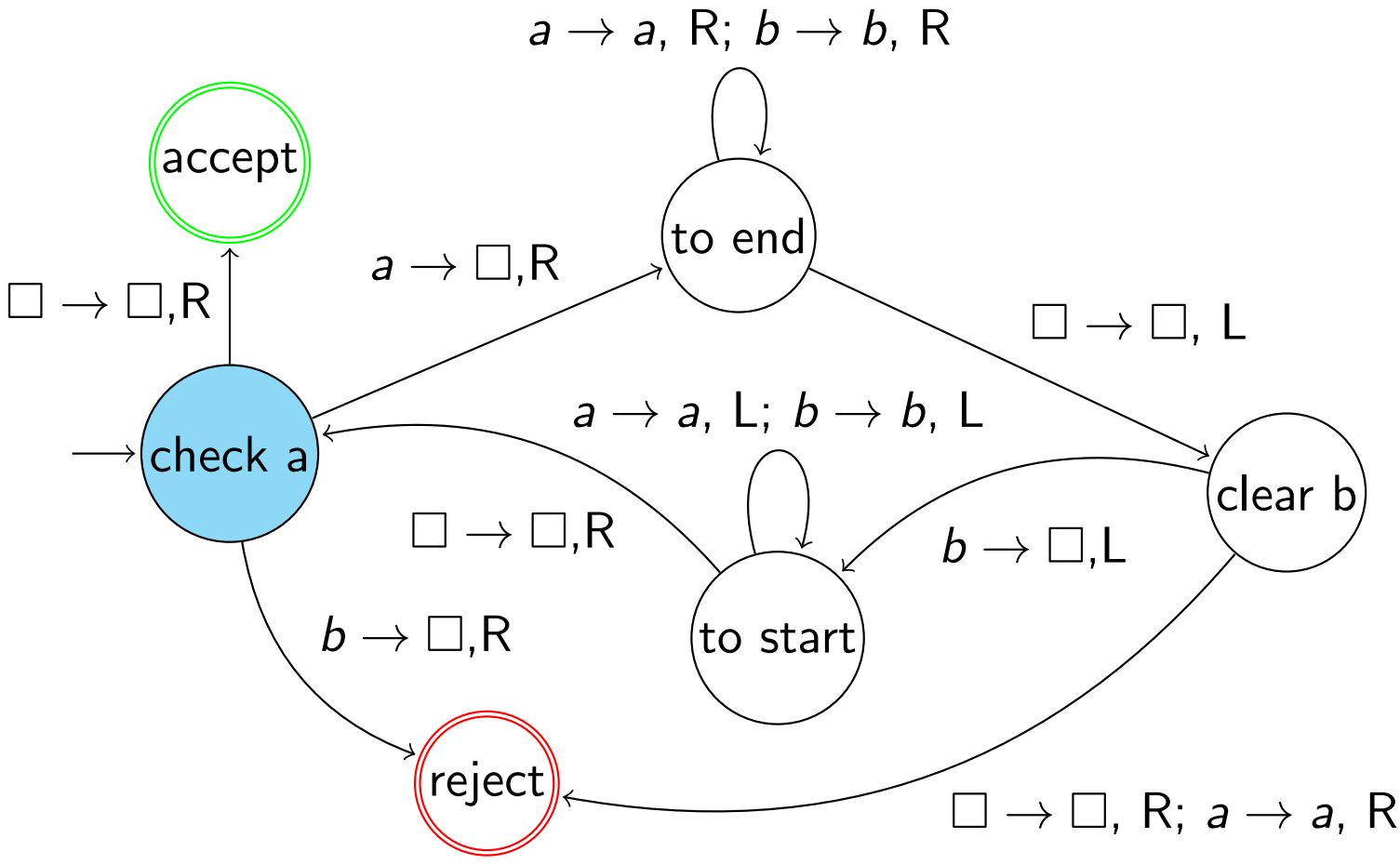
La fiecare pas, este citit/scris *doar* simbolul aflat *imediat sub* capul de citire/scriere!

Ca orice automat, mașina Turing începe execuția din *starea inițială*.

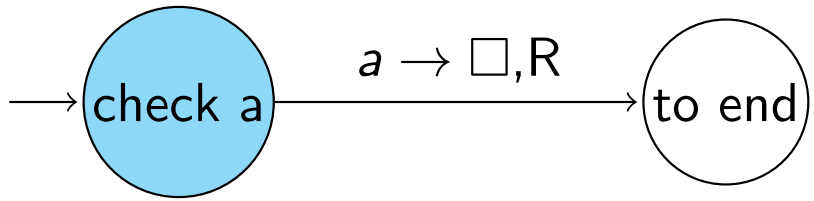


Într-o mașină Turing, tranzițiile dintre stări au forma de mai jos

simbol citit $\rightarrow$ *simbol scris*, *direcție* mutare cap (L/R)



Într-o mașină Turing, tranzițiile dintre stări au forma de mai jos
simbol citit $\rightarrow$ *simbol scris*, *direcție* mutare cap (L/R)

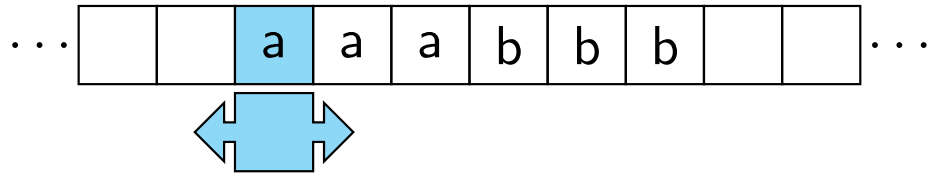


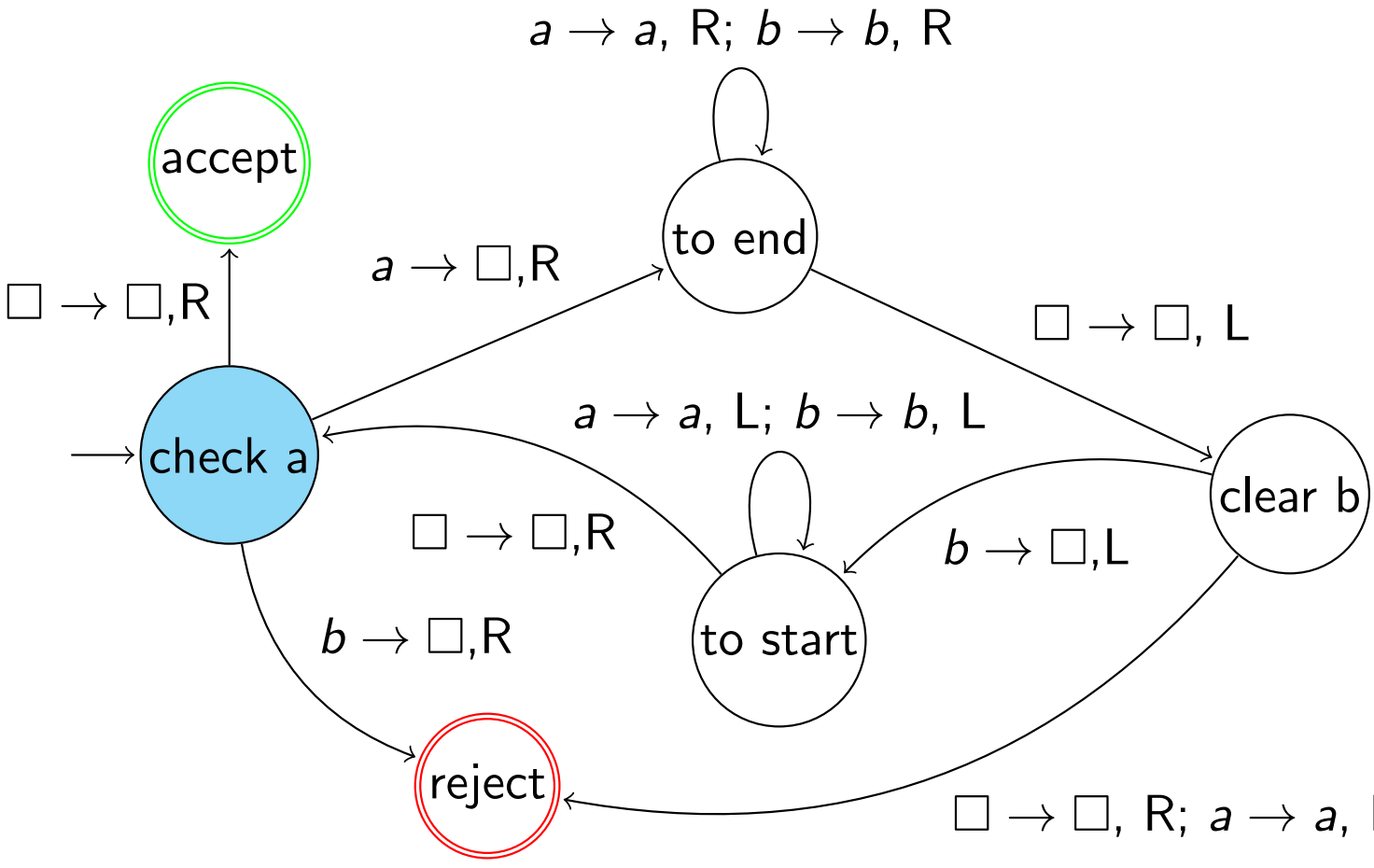
Tranziția $a \rightarrow \square$, R: *dacă* se citește simbolul a de pe bandă, atunci se scrie $\square$ și se mută capul de citire/scriere cu o celulă la *dreapta*

R: mută capul de citire/scriere cu o celulă la dreapta
L: mută capul de citire/scriere cu o celulă la stânga

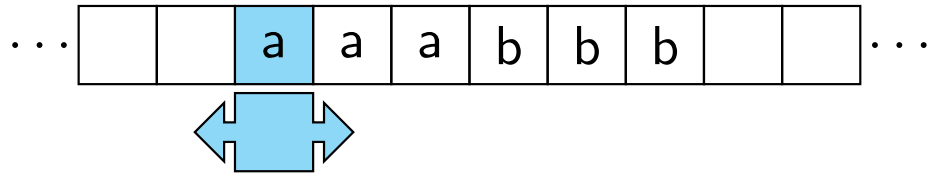
$\square$: simbolul vid

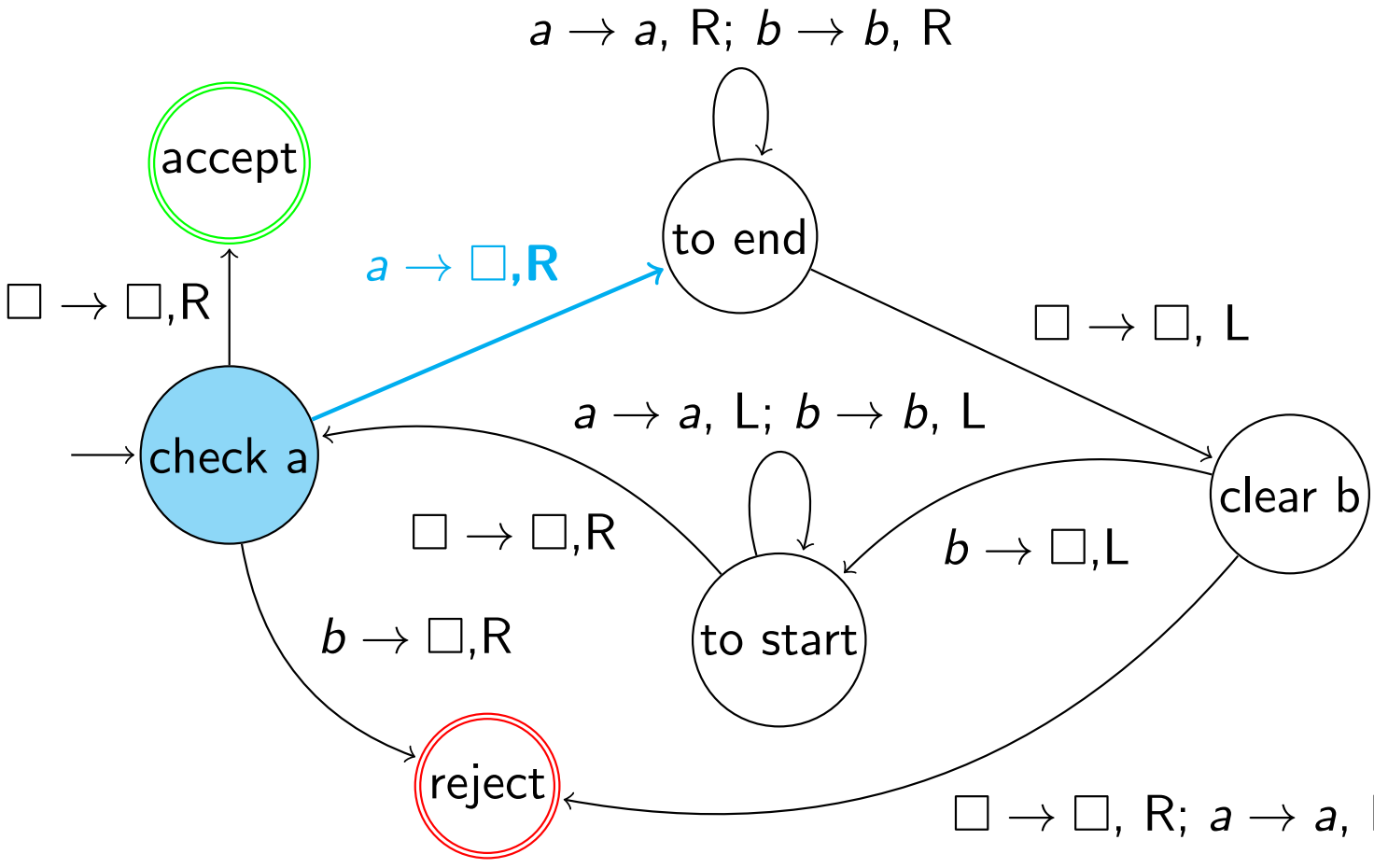
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



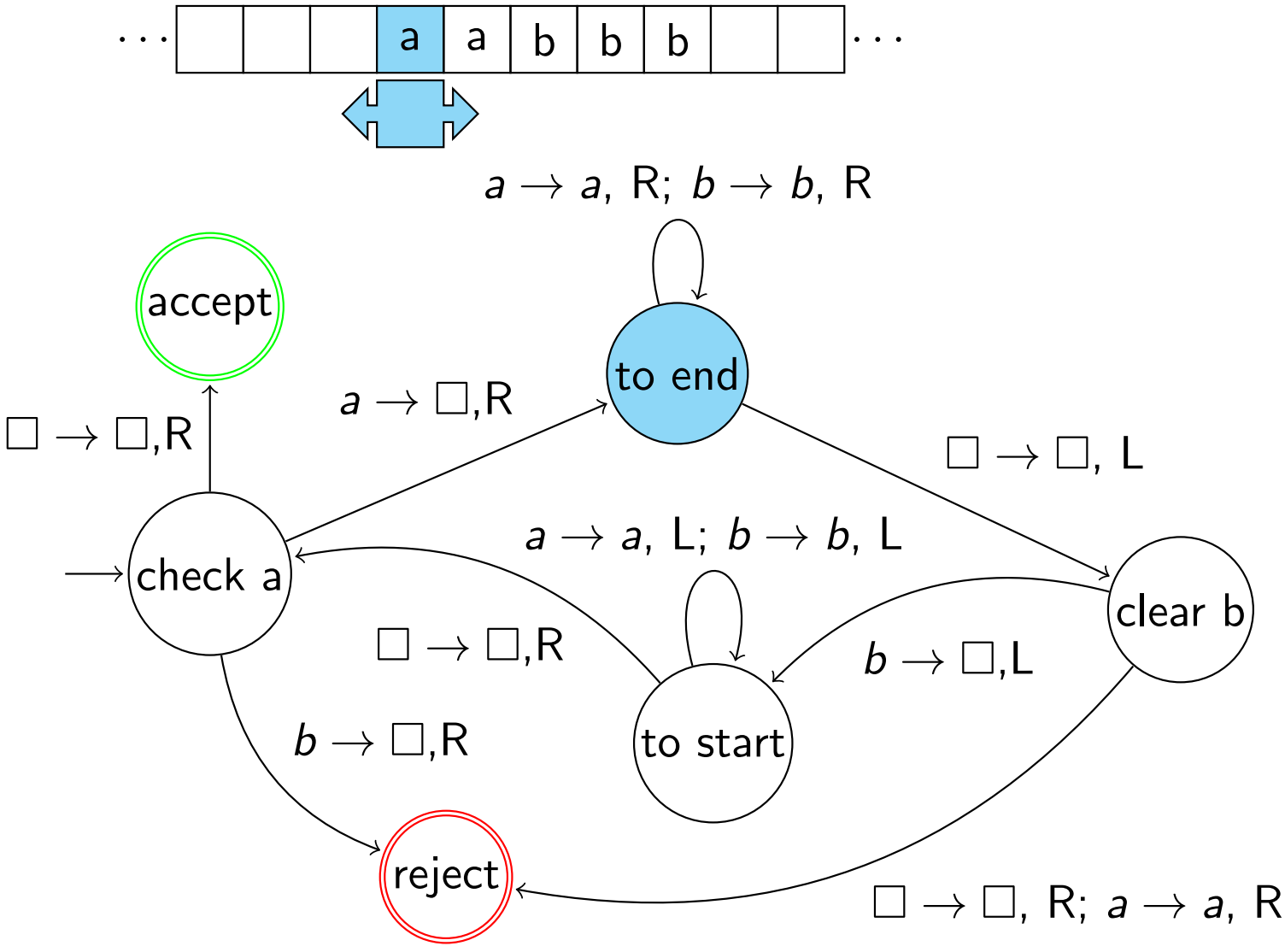


Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$

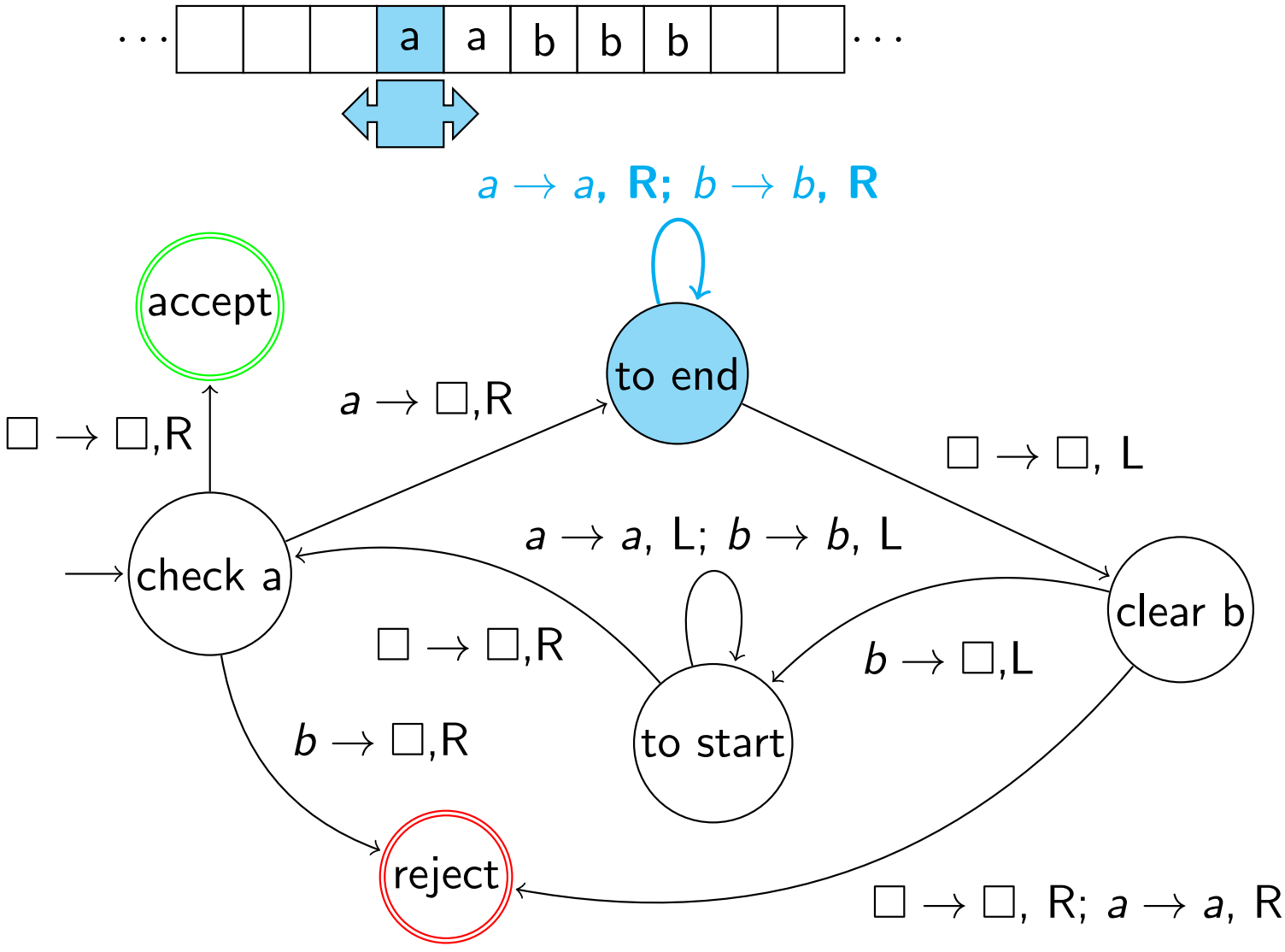




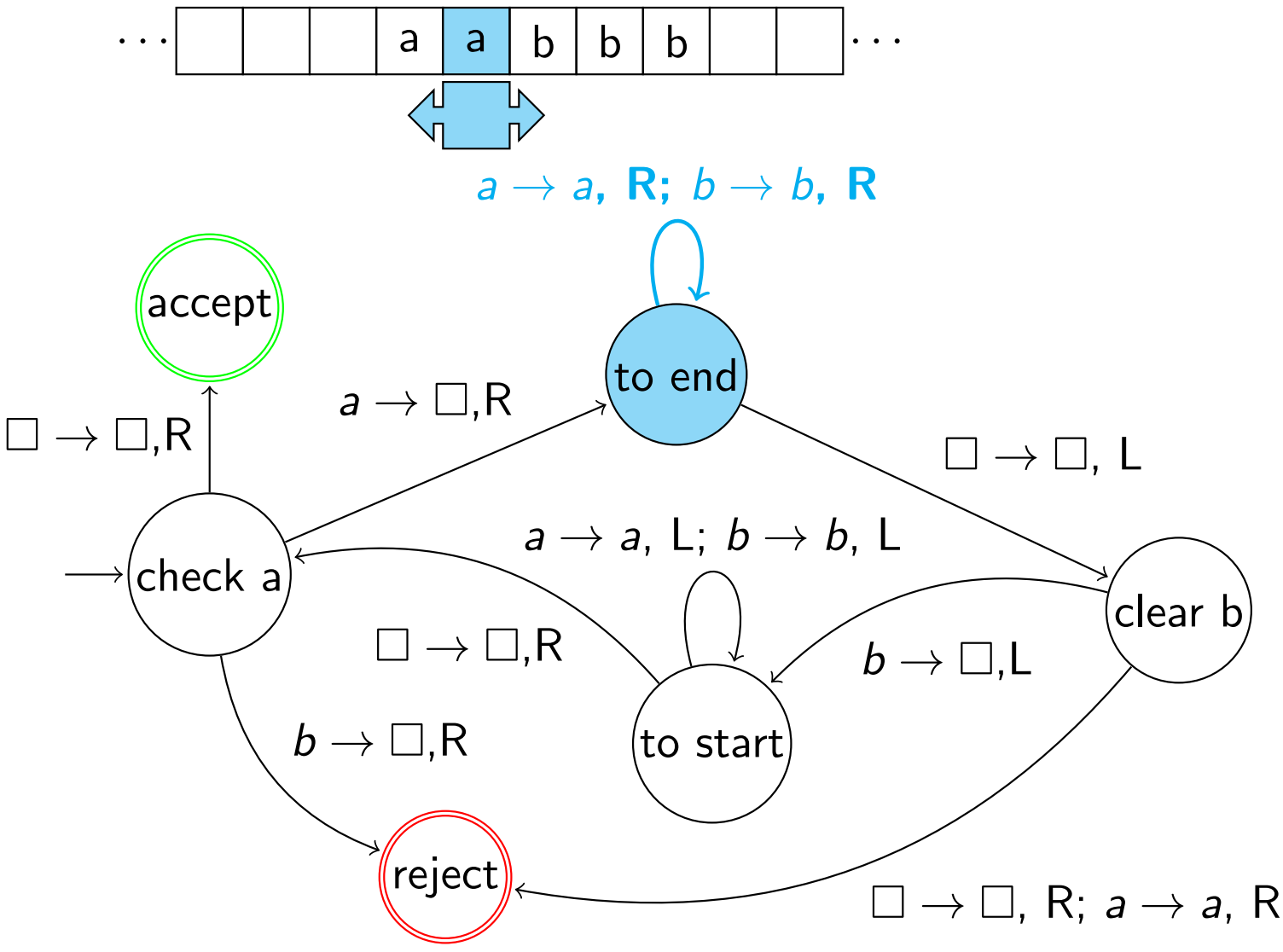
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



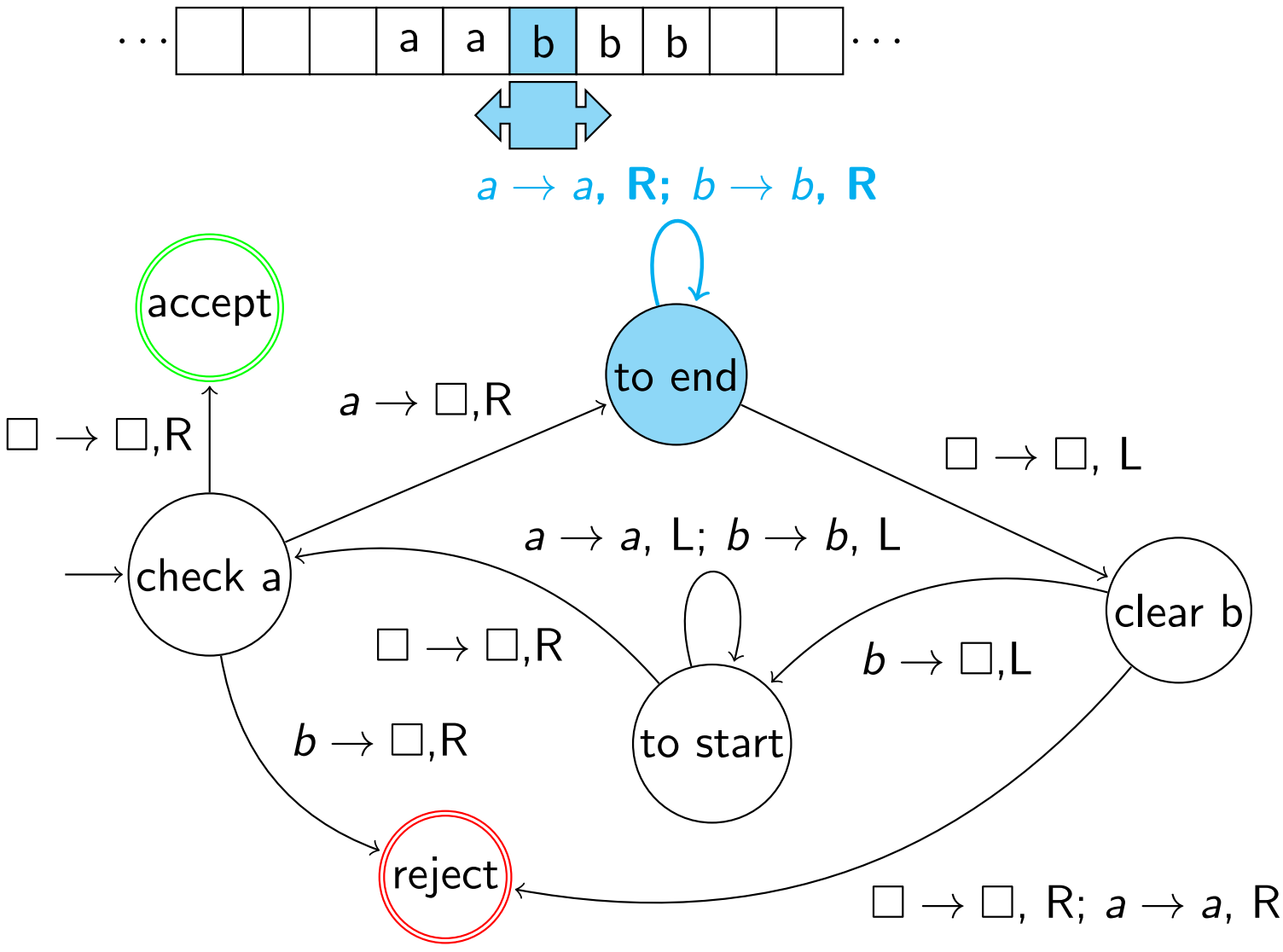
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



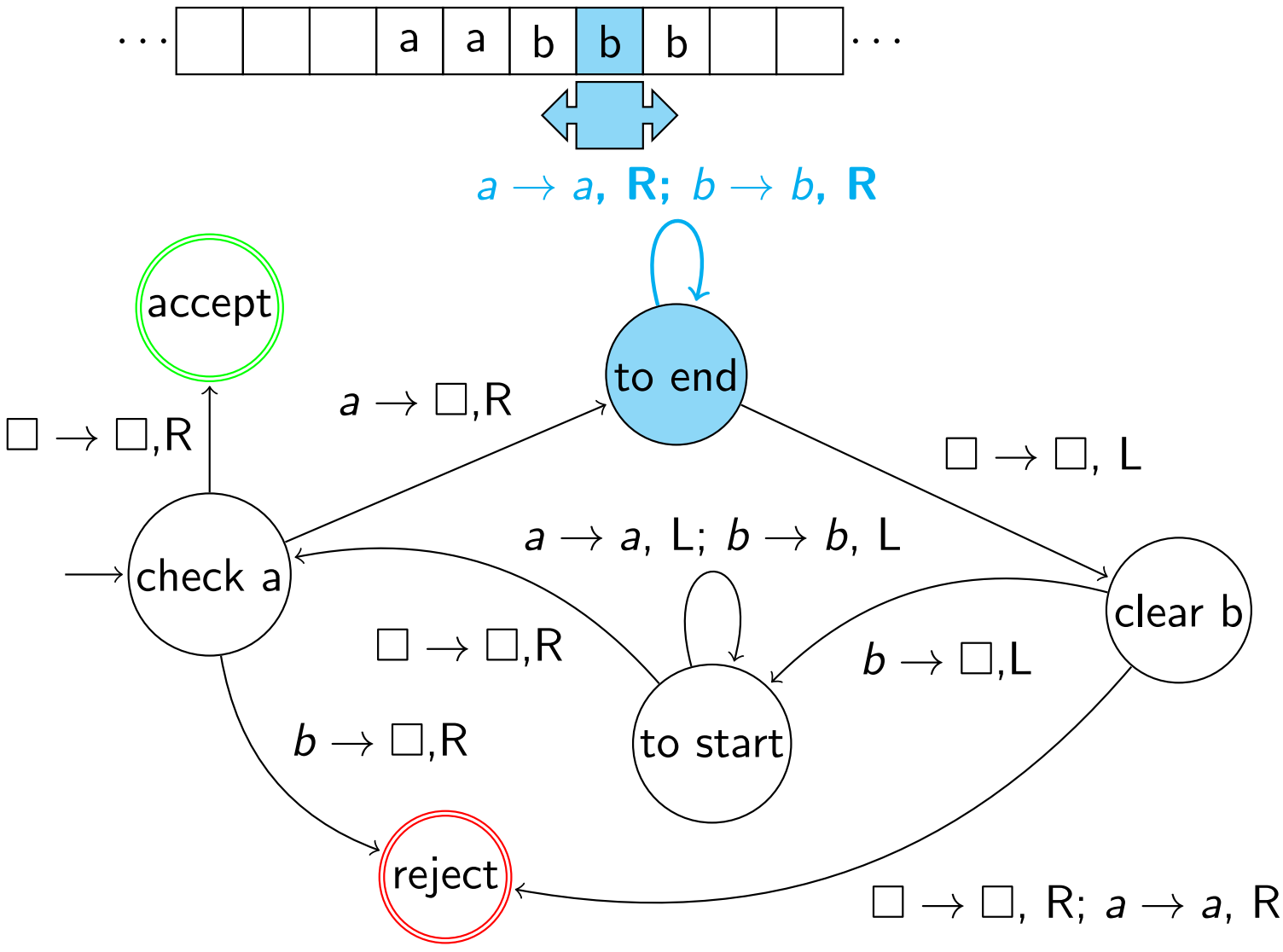
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



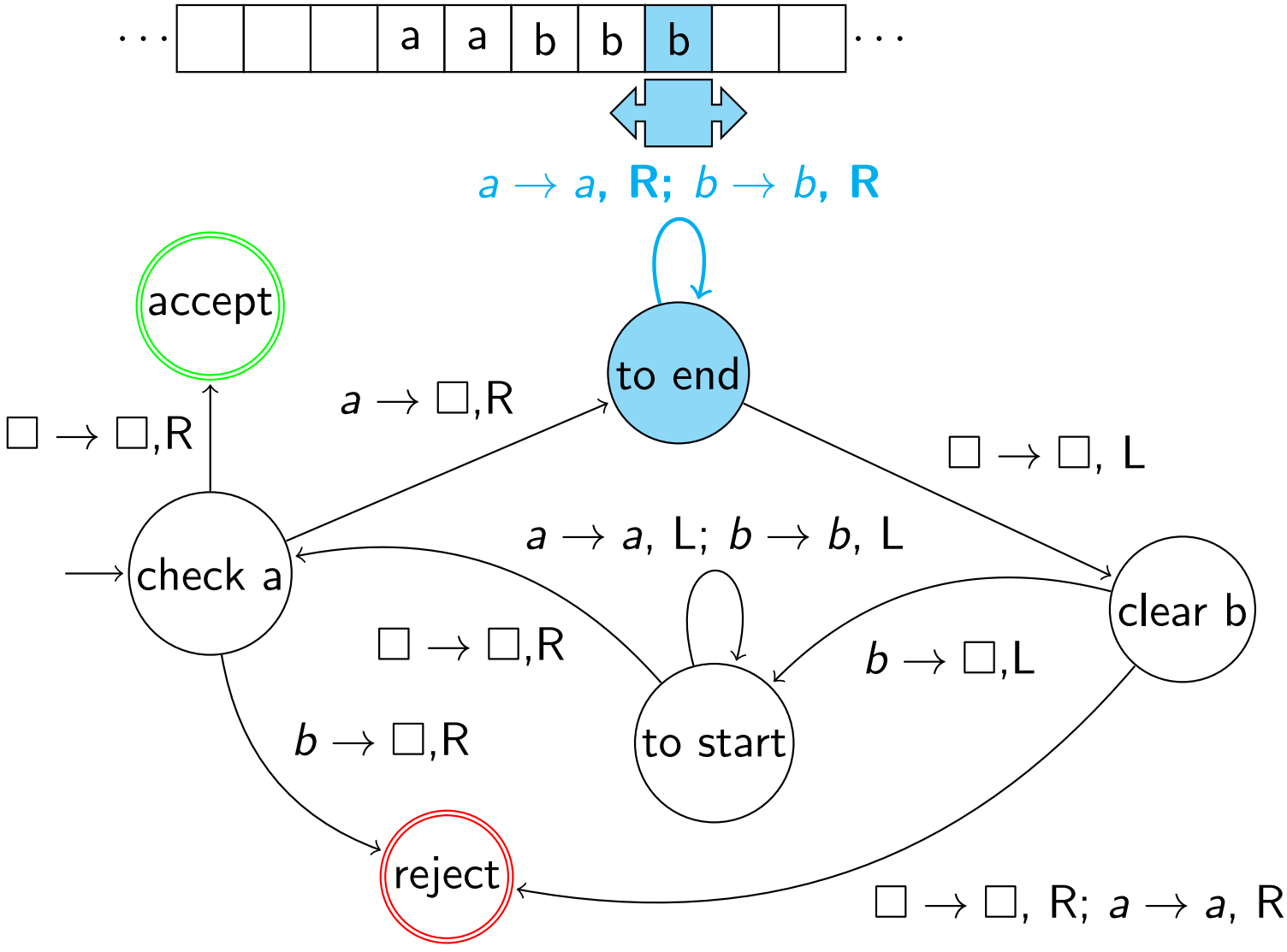
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



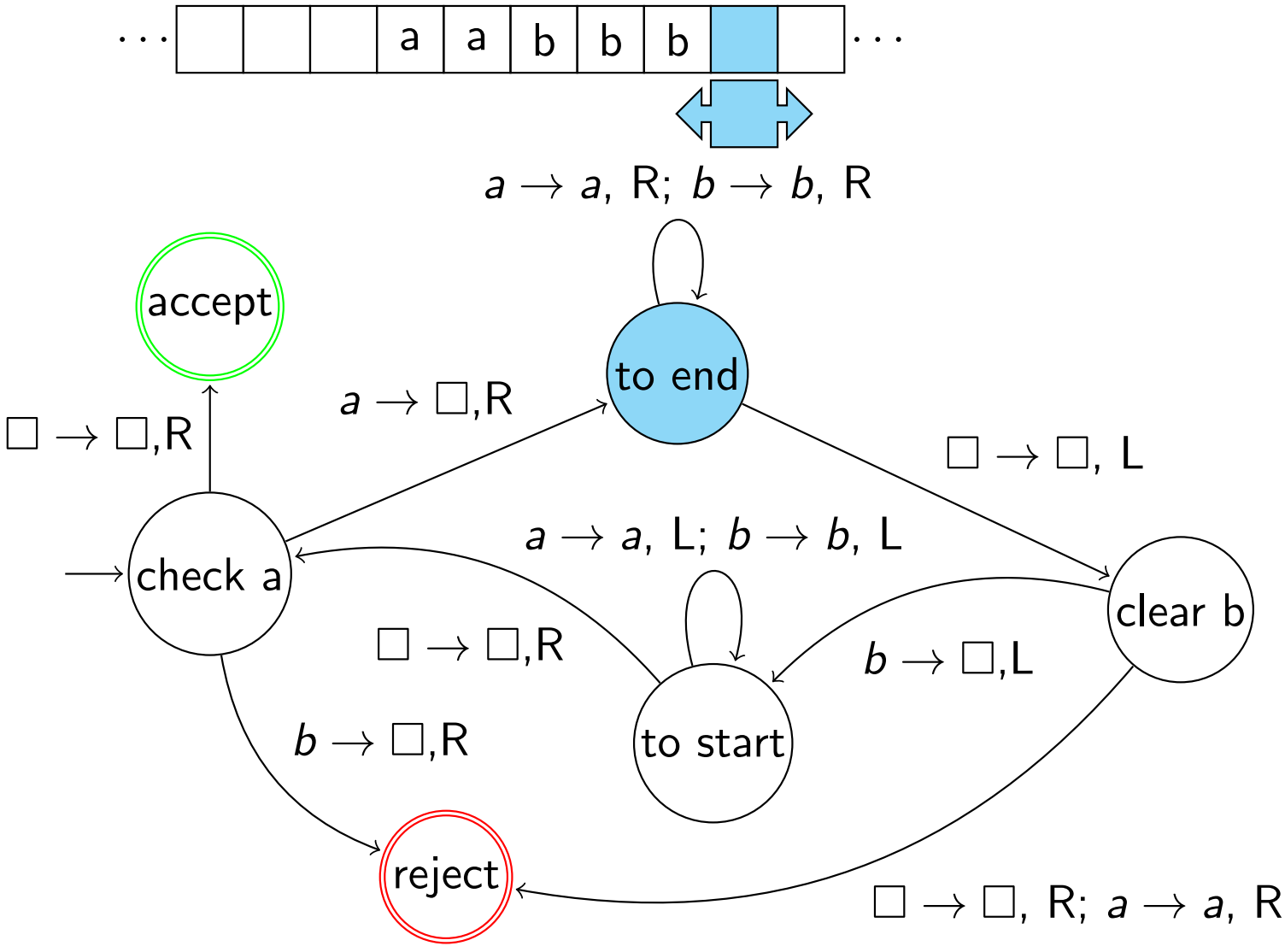
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



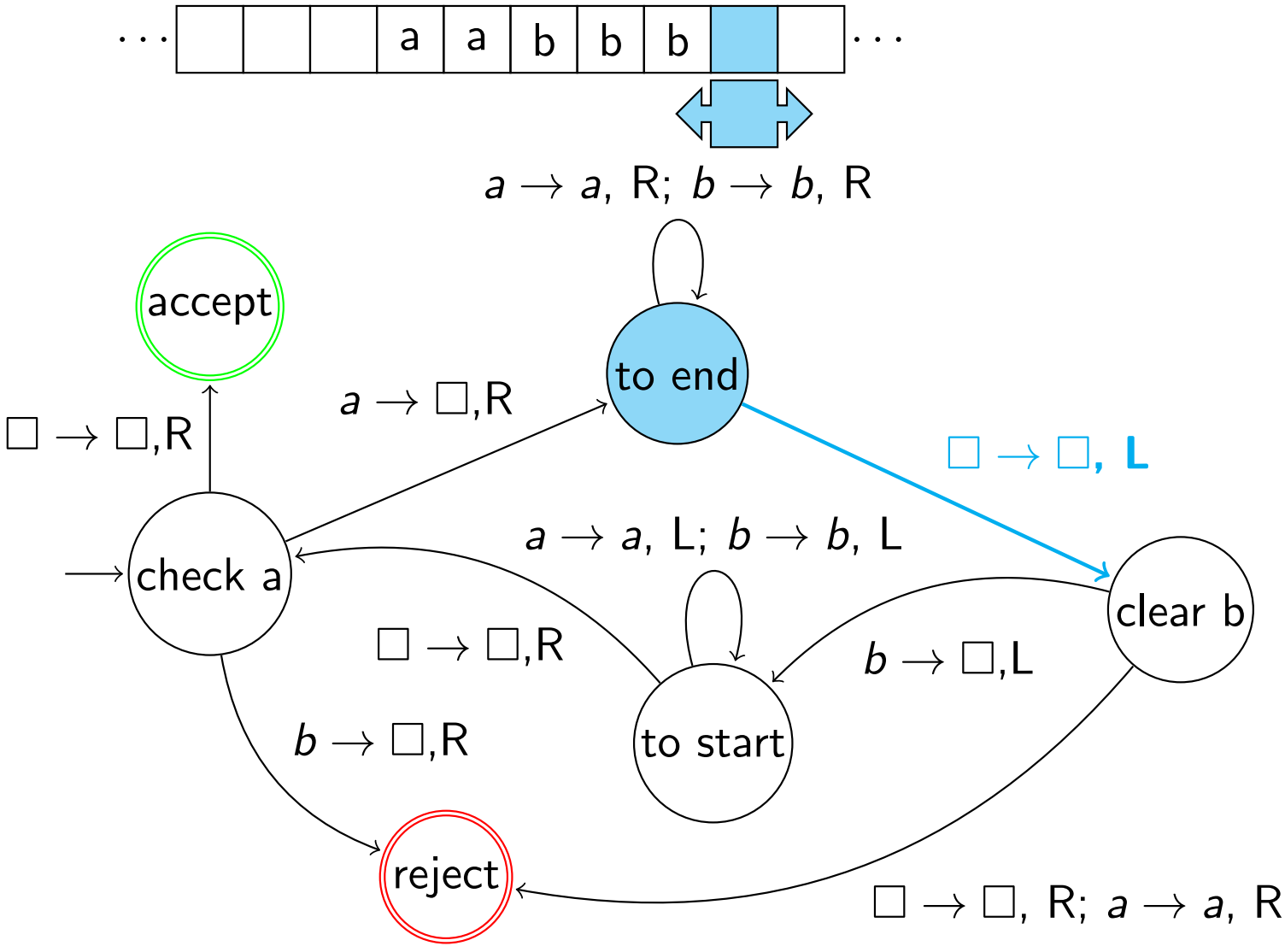
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



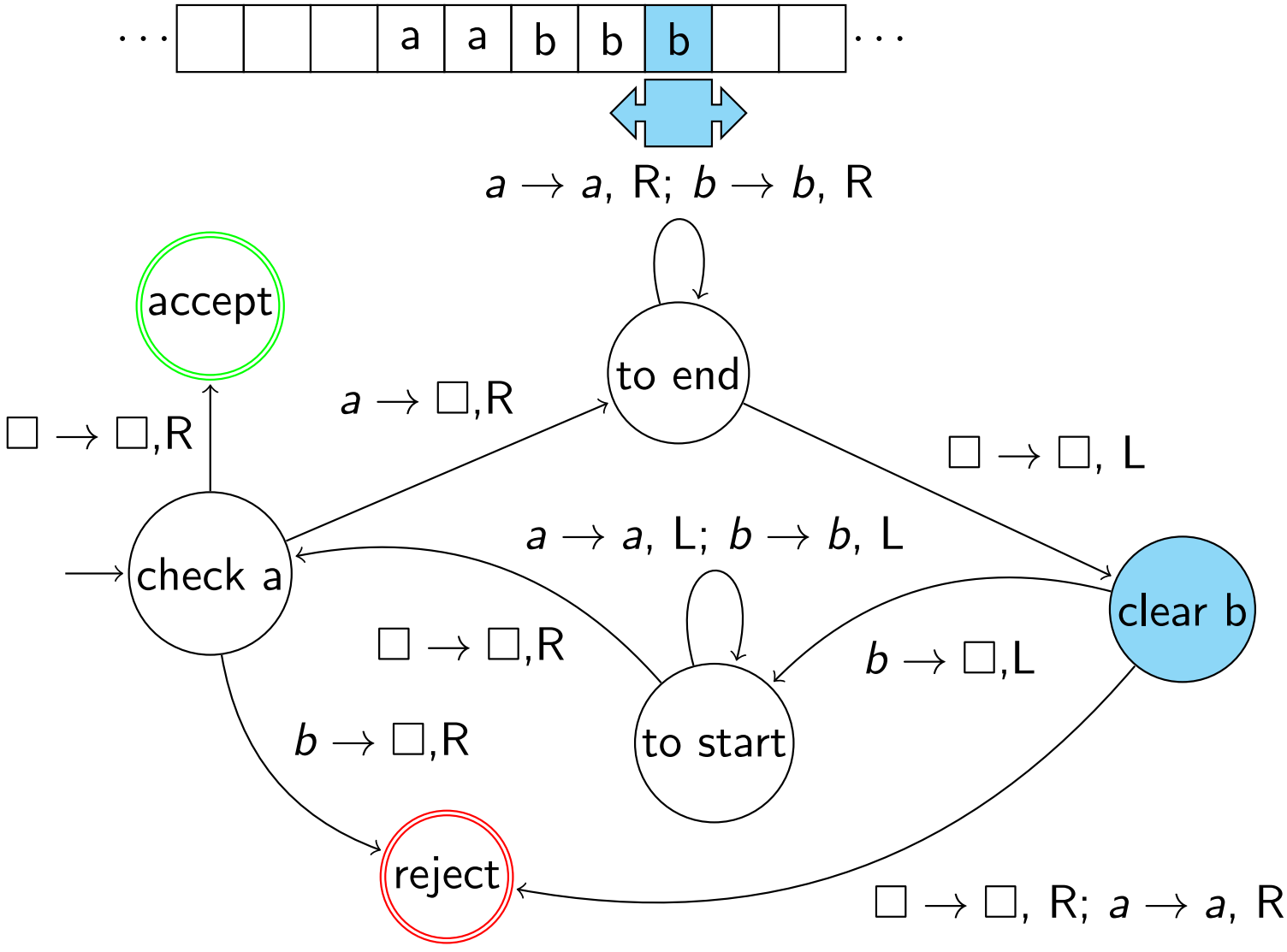
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



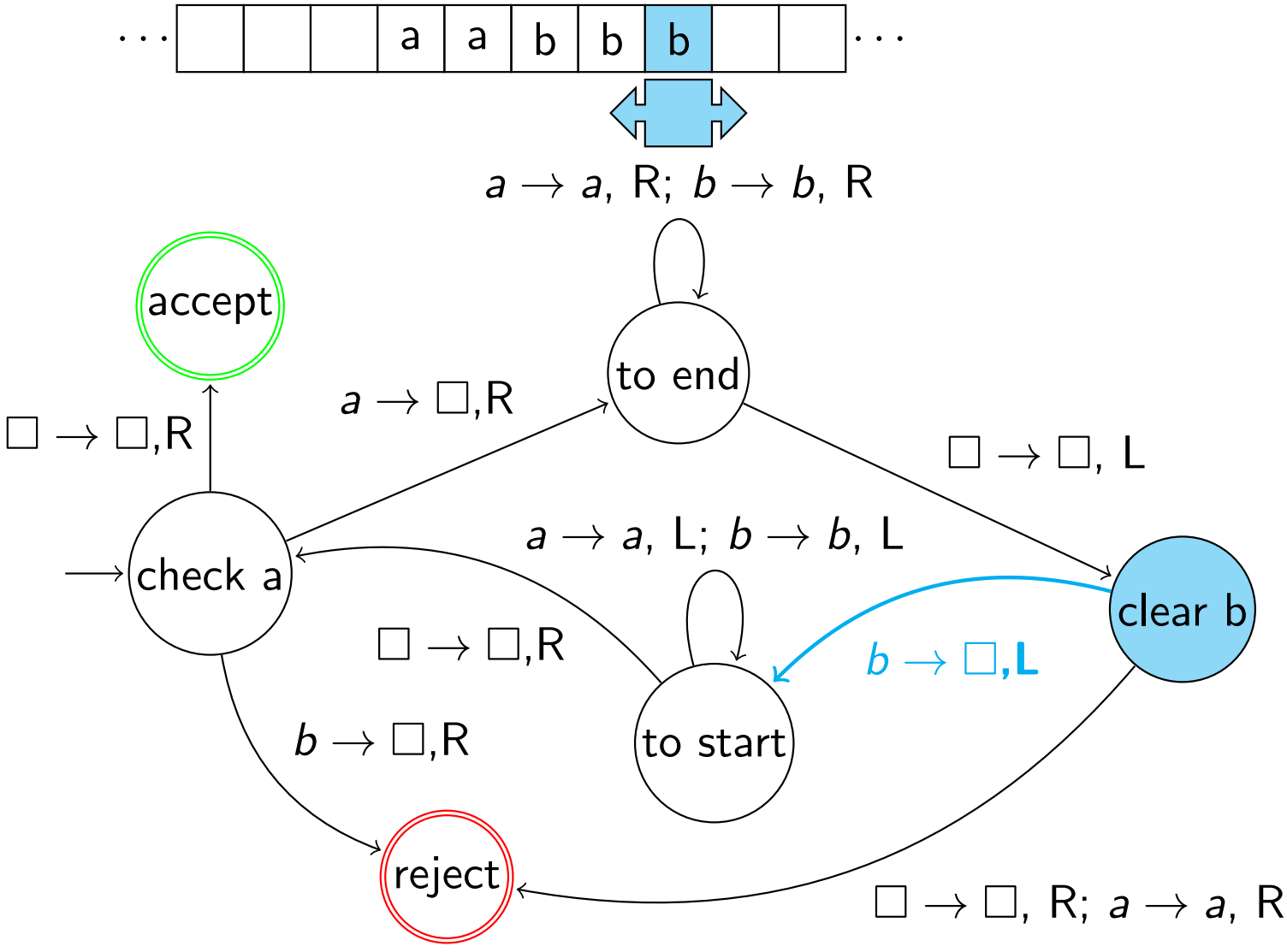
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



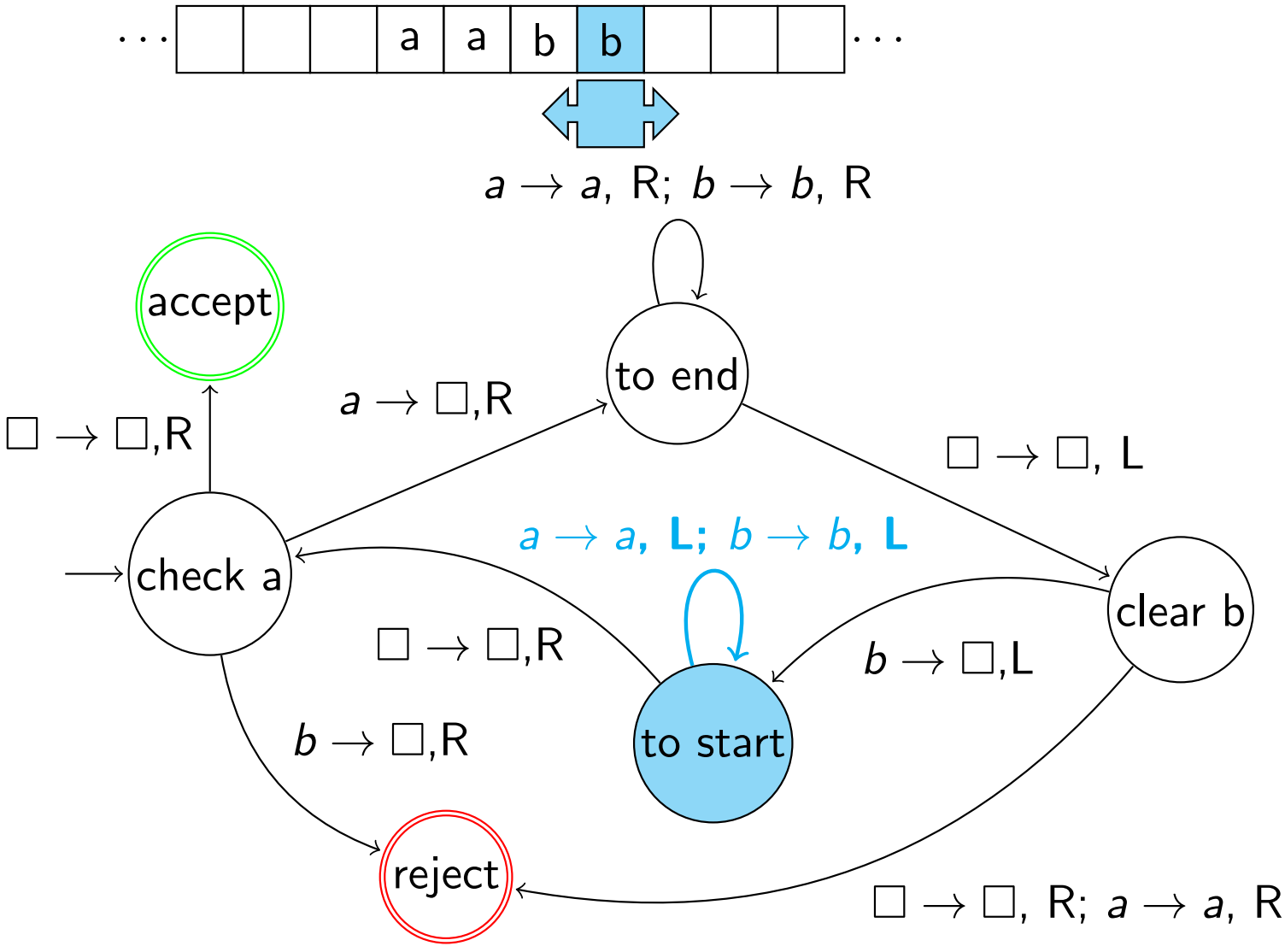
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



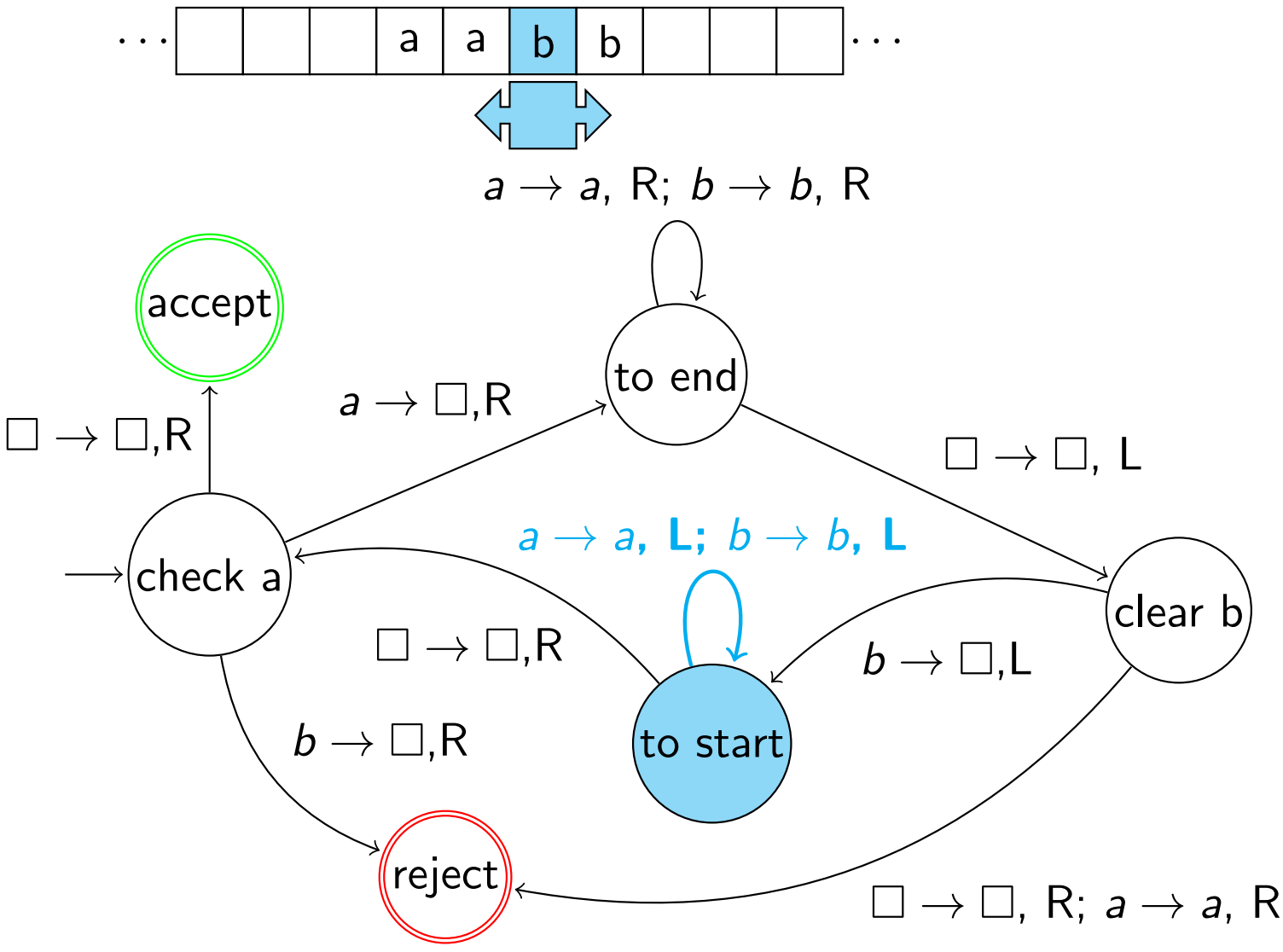
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



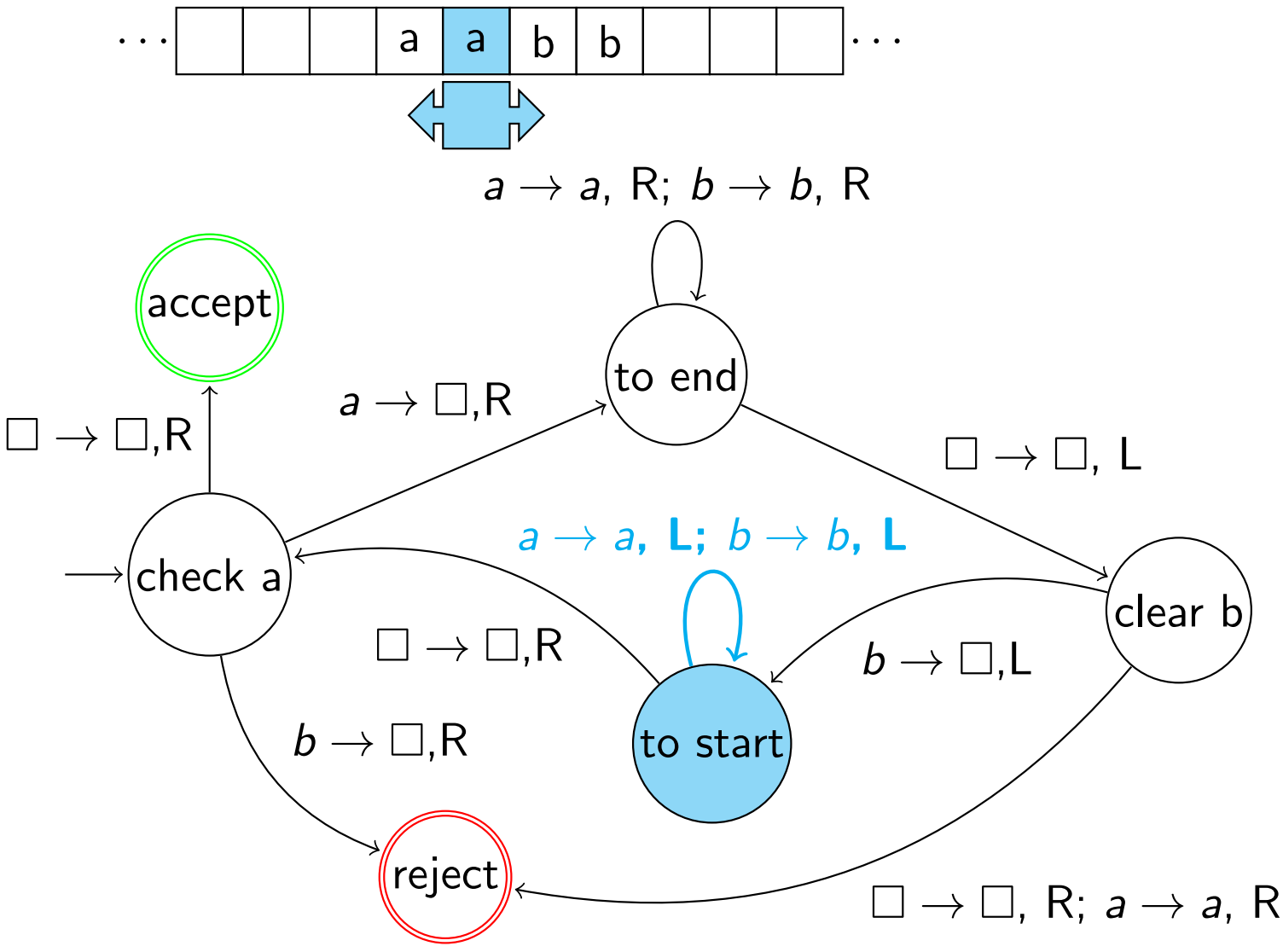
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



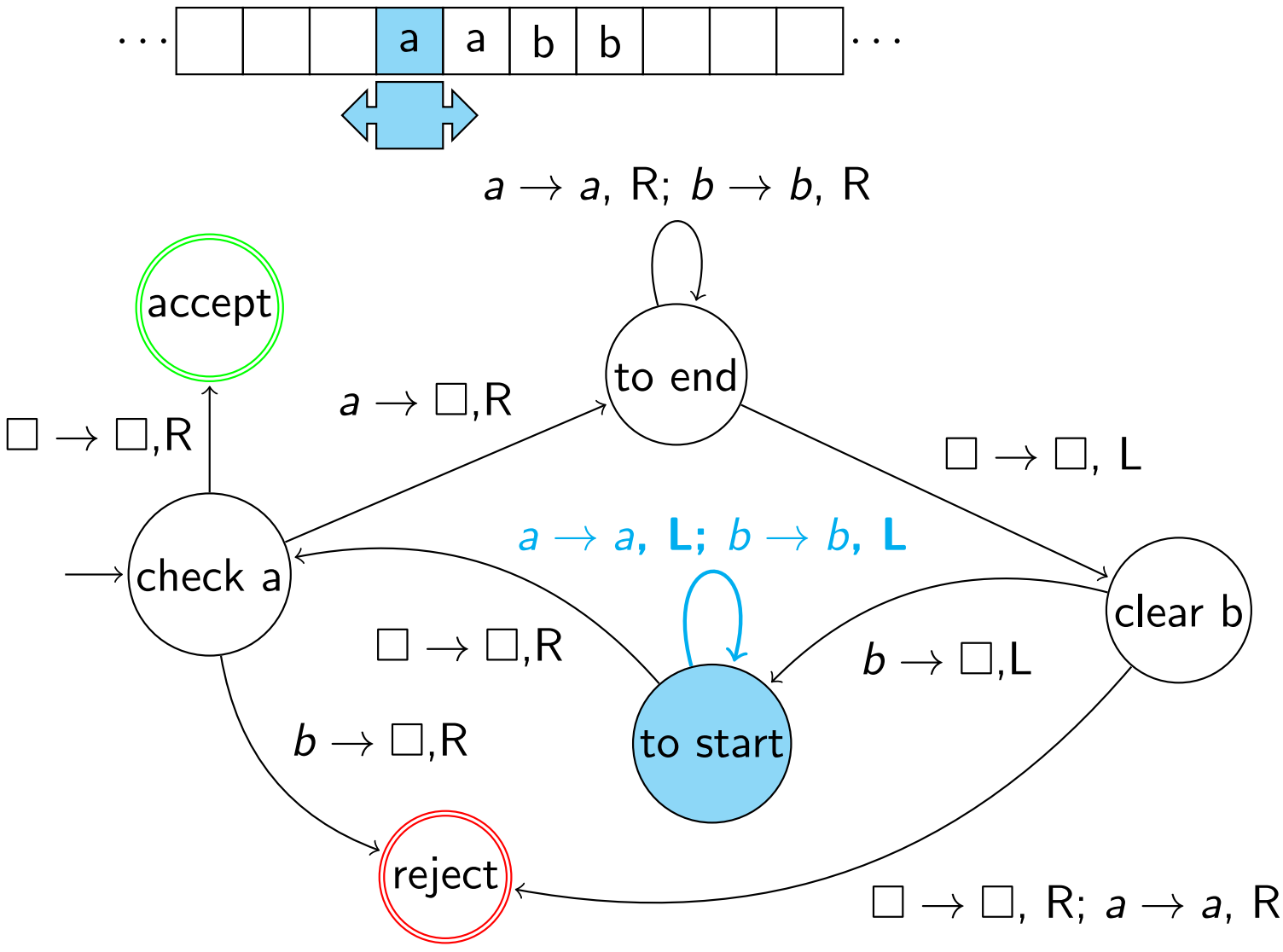
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



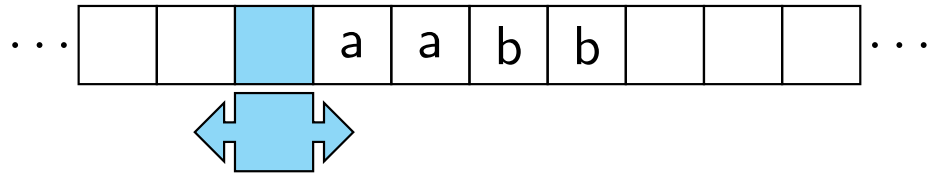
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$

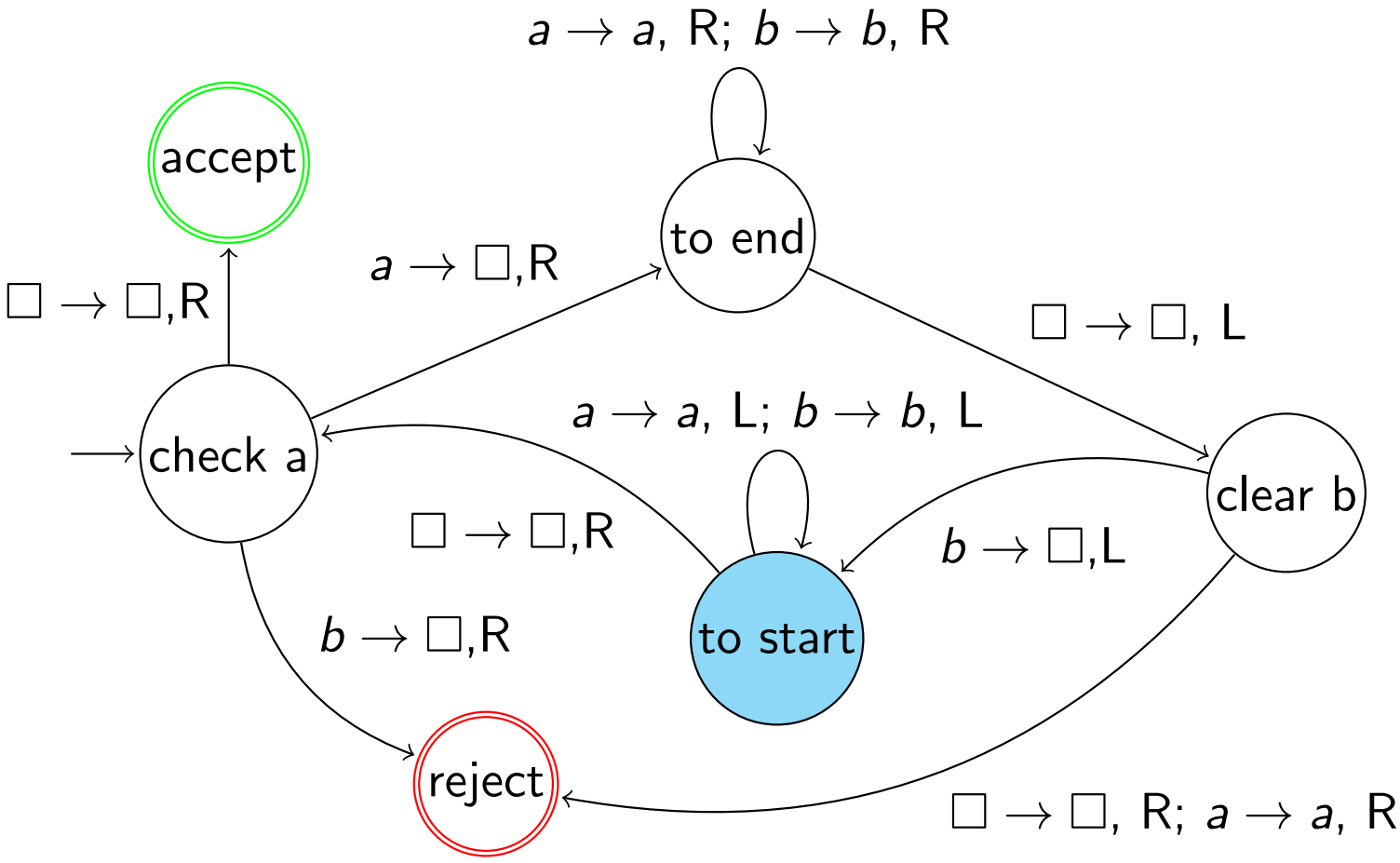


Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$

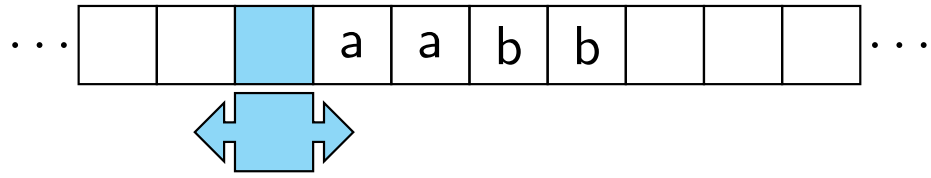


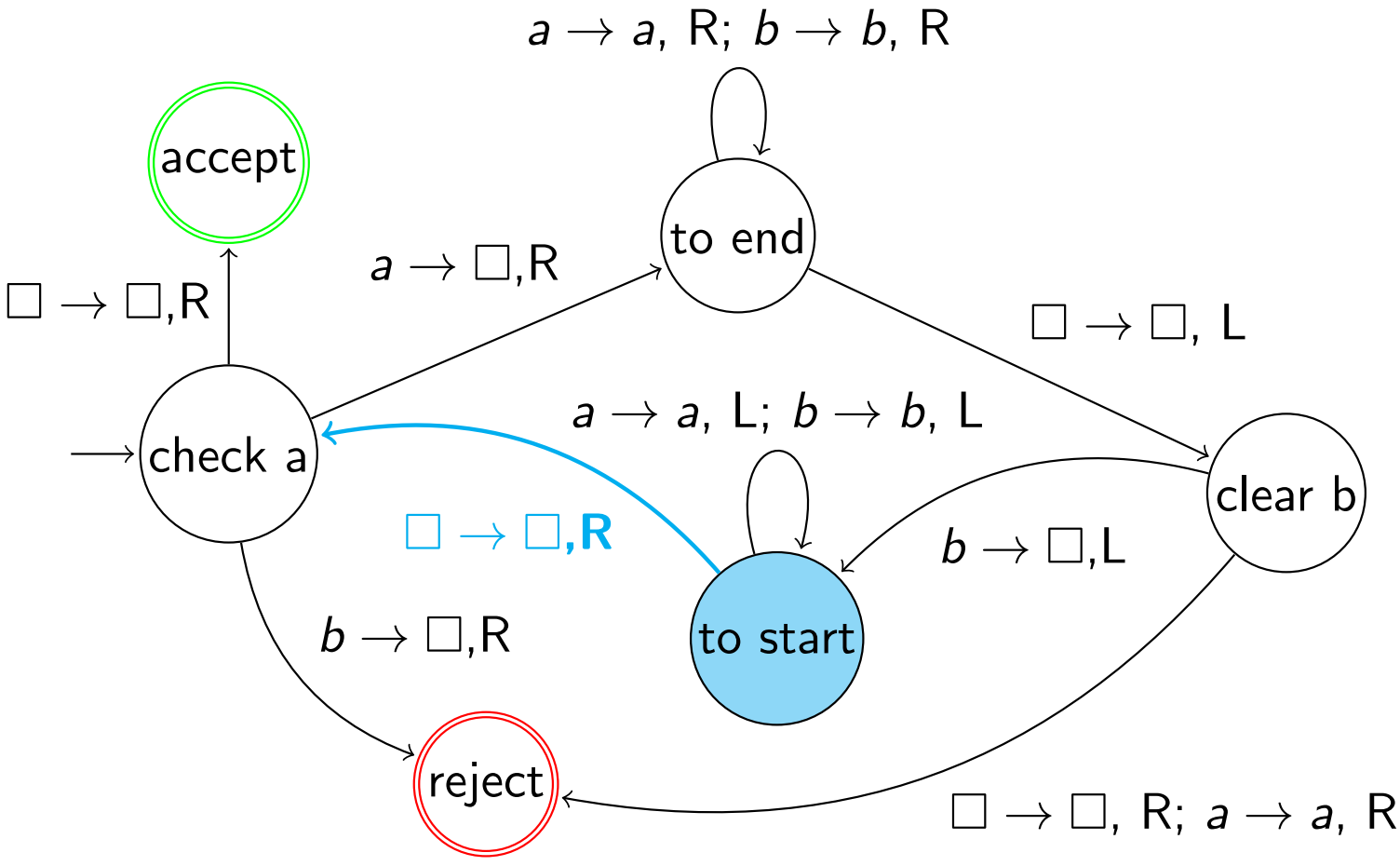
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



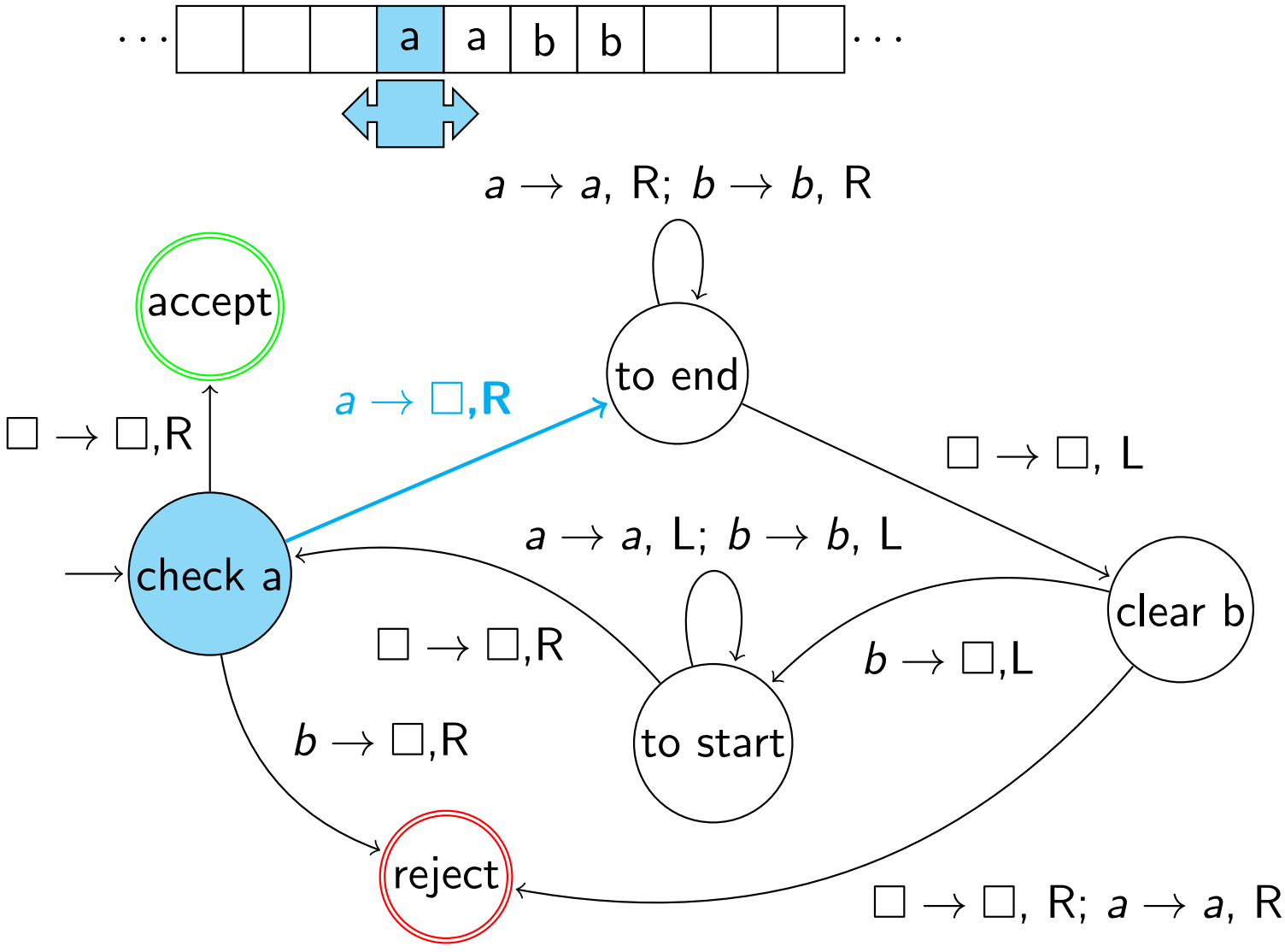


Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$

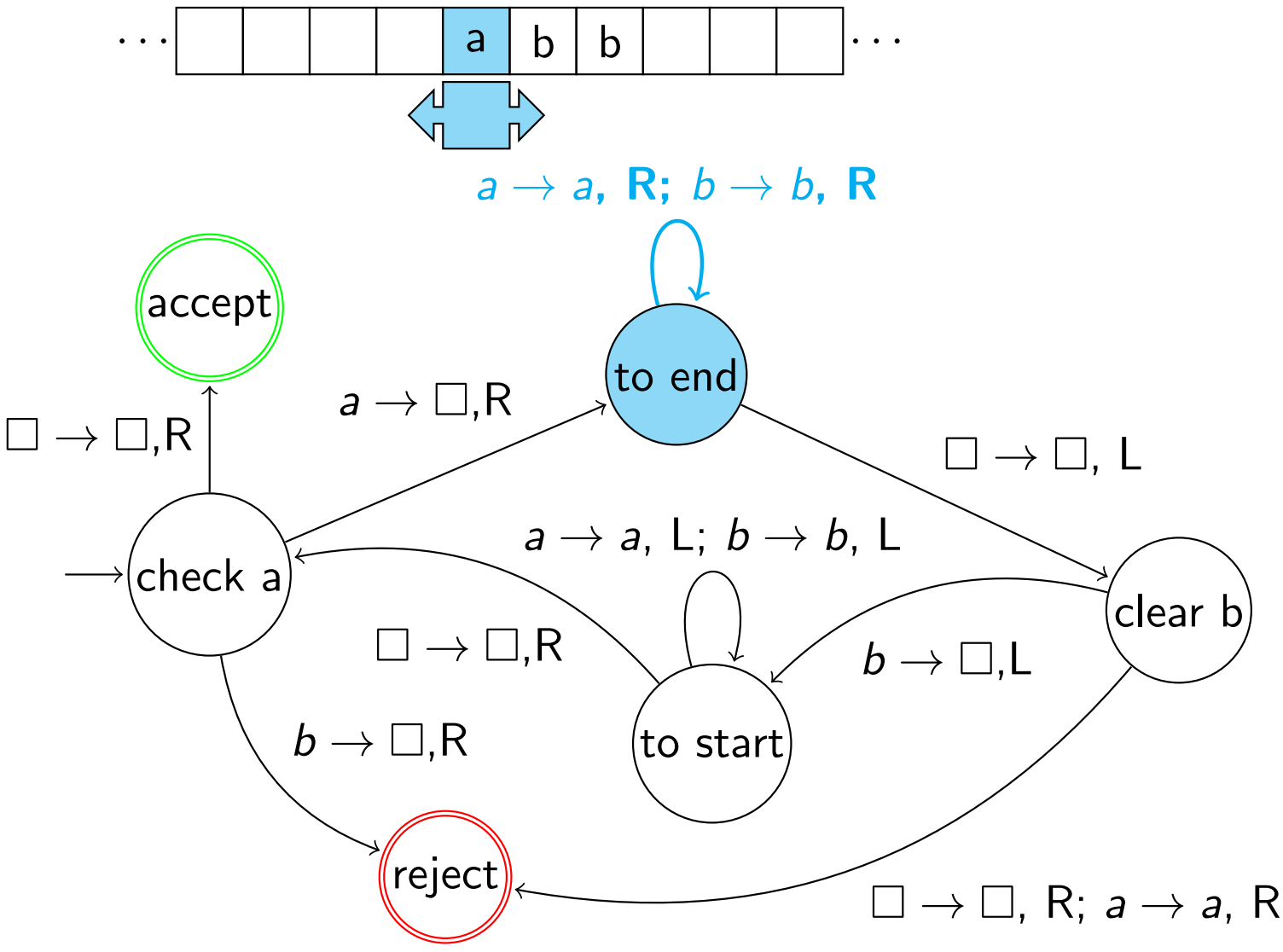




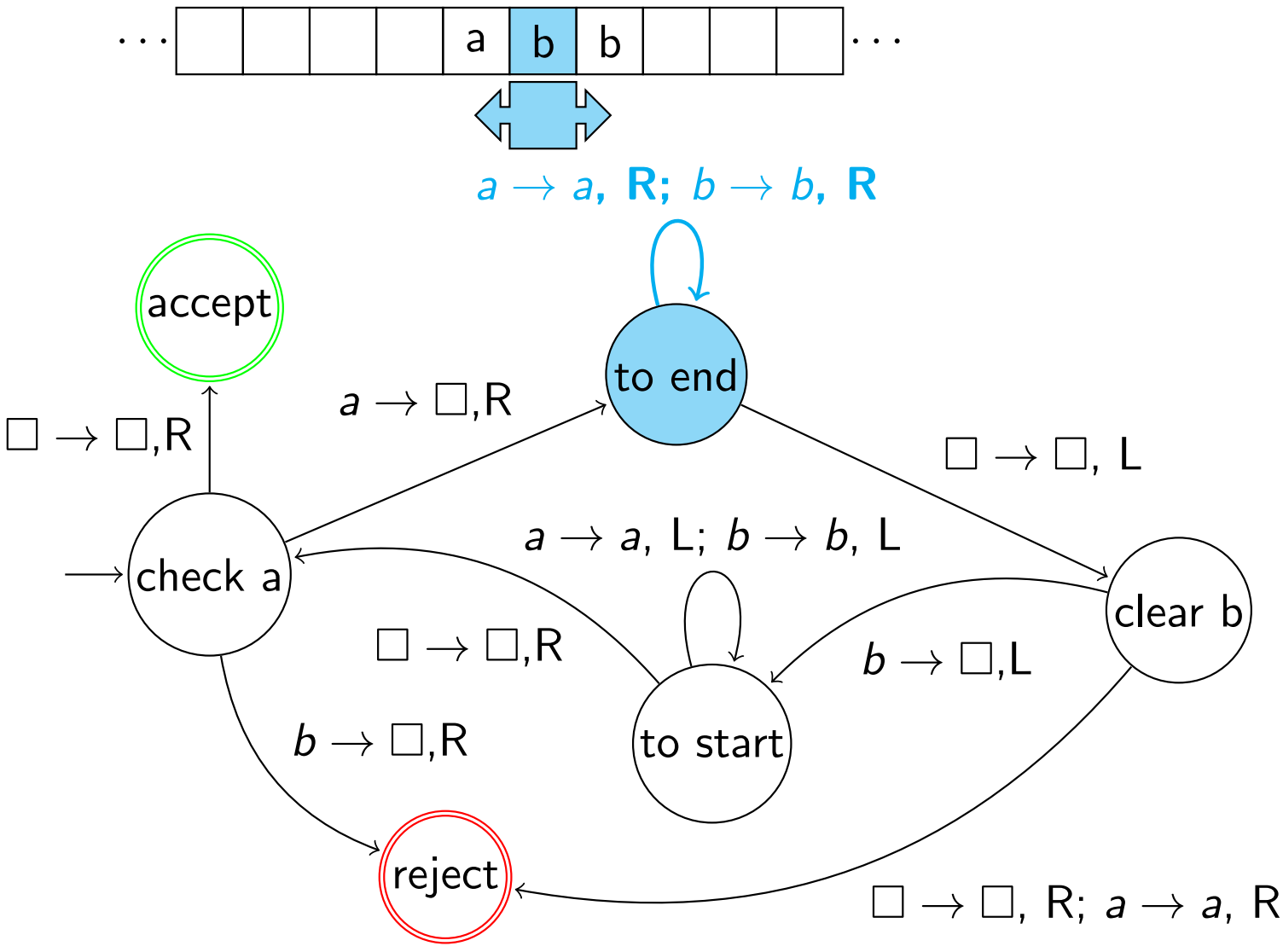
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



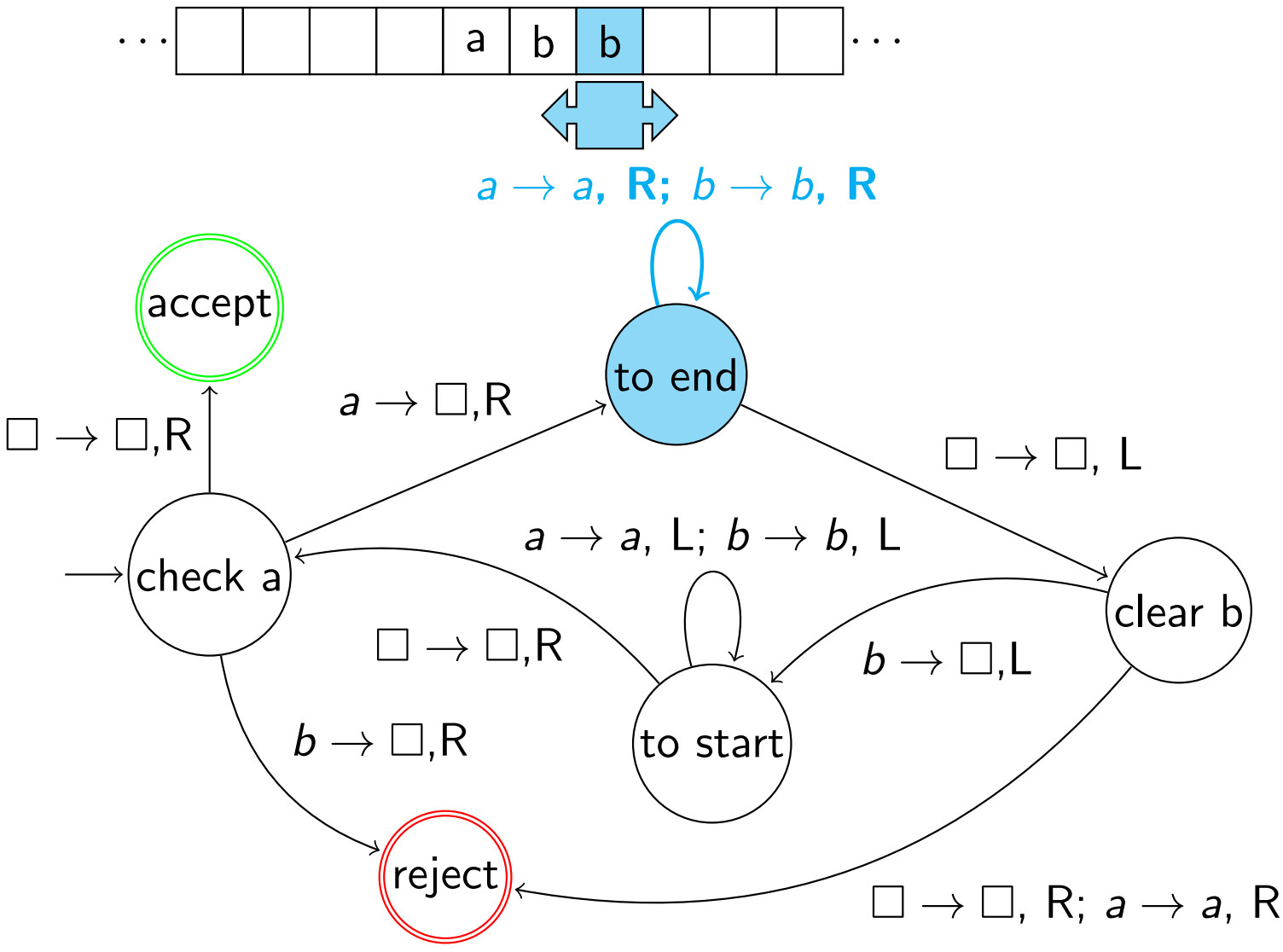
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



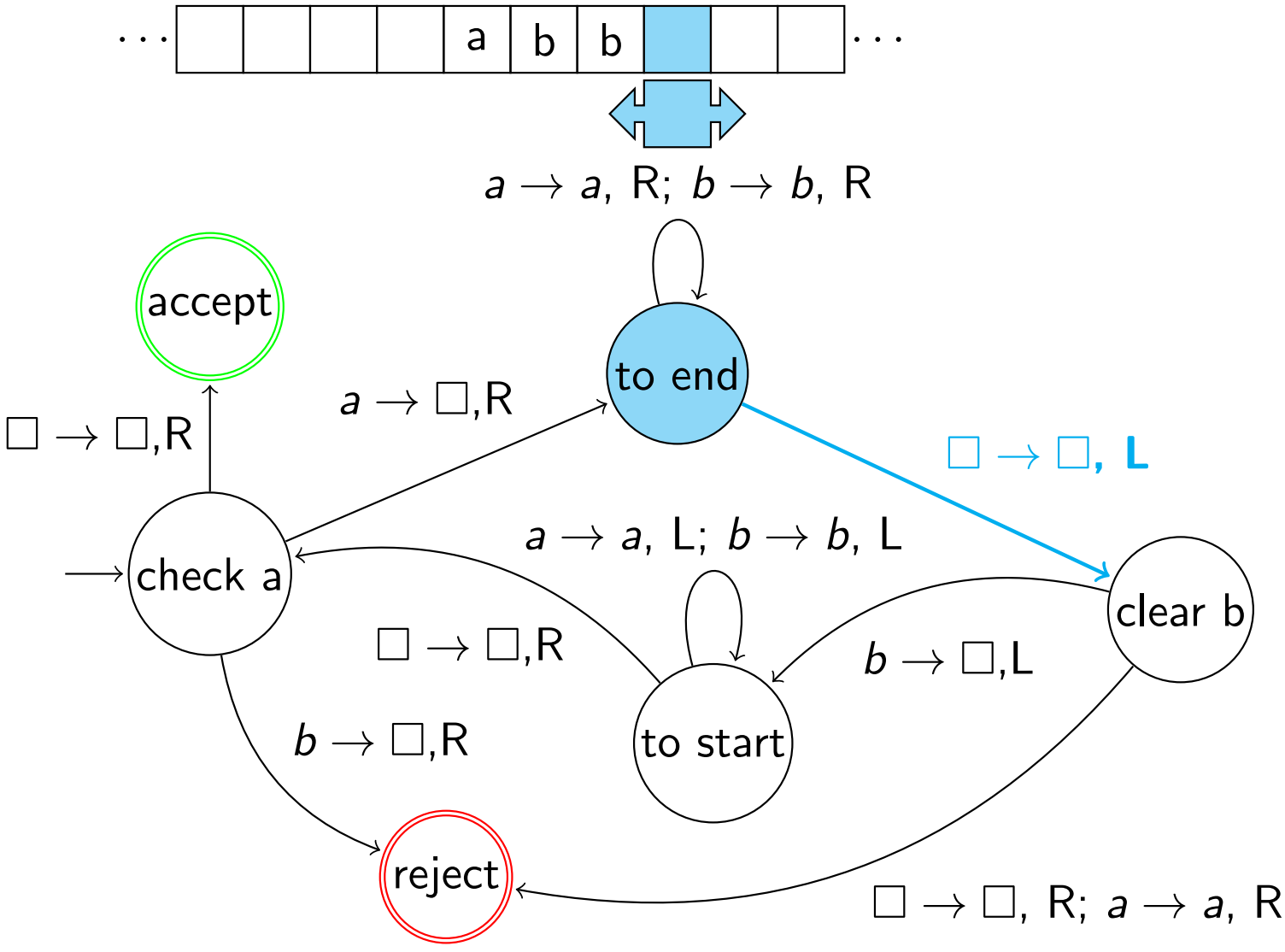
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



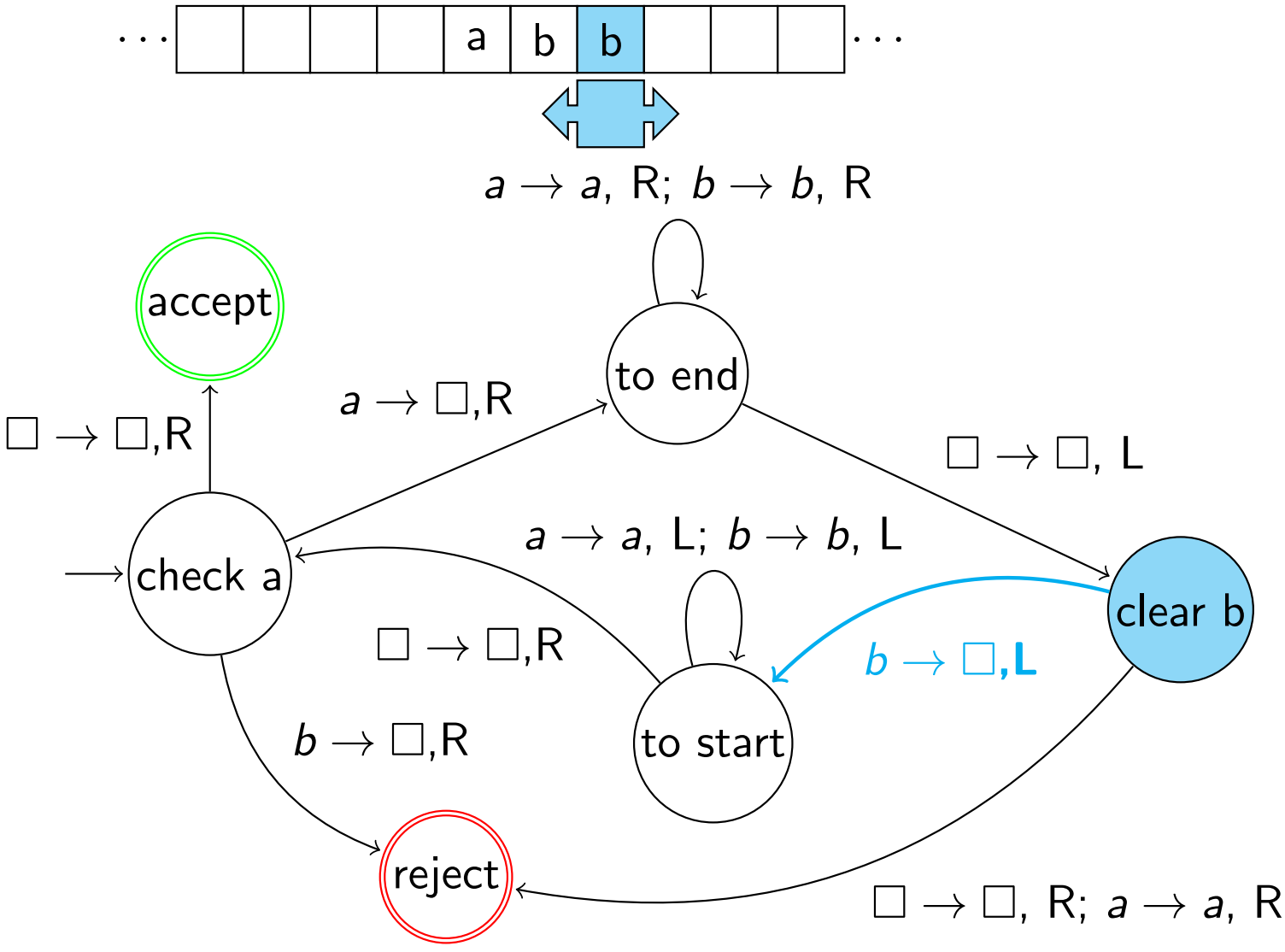
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



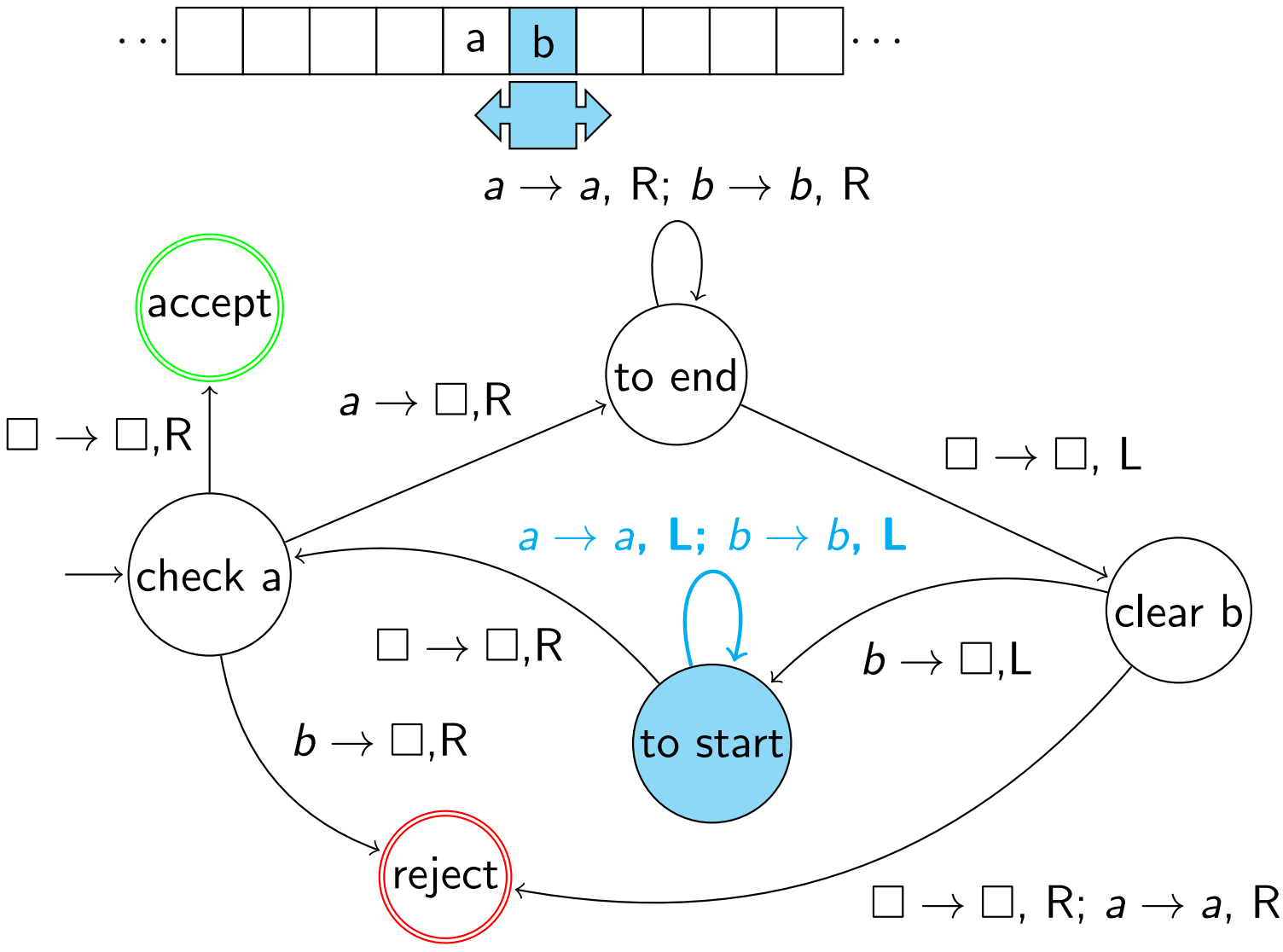
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



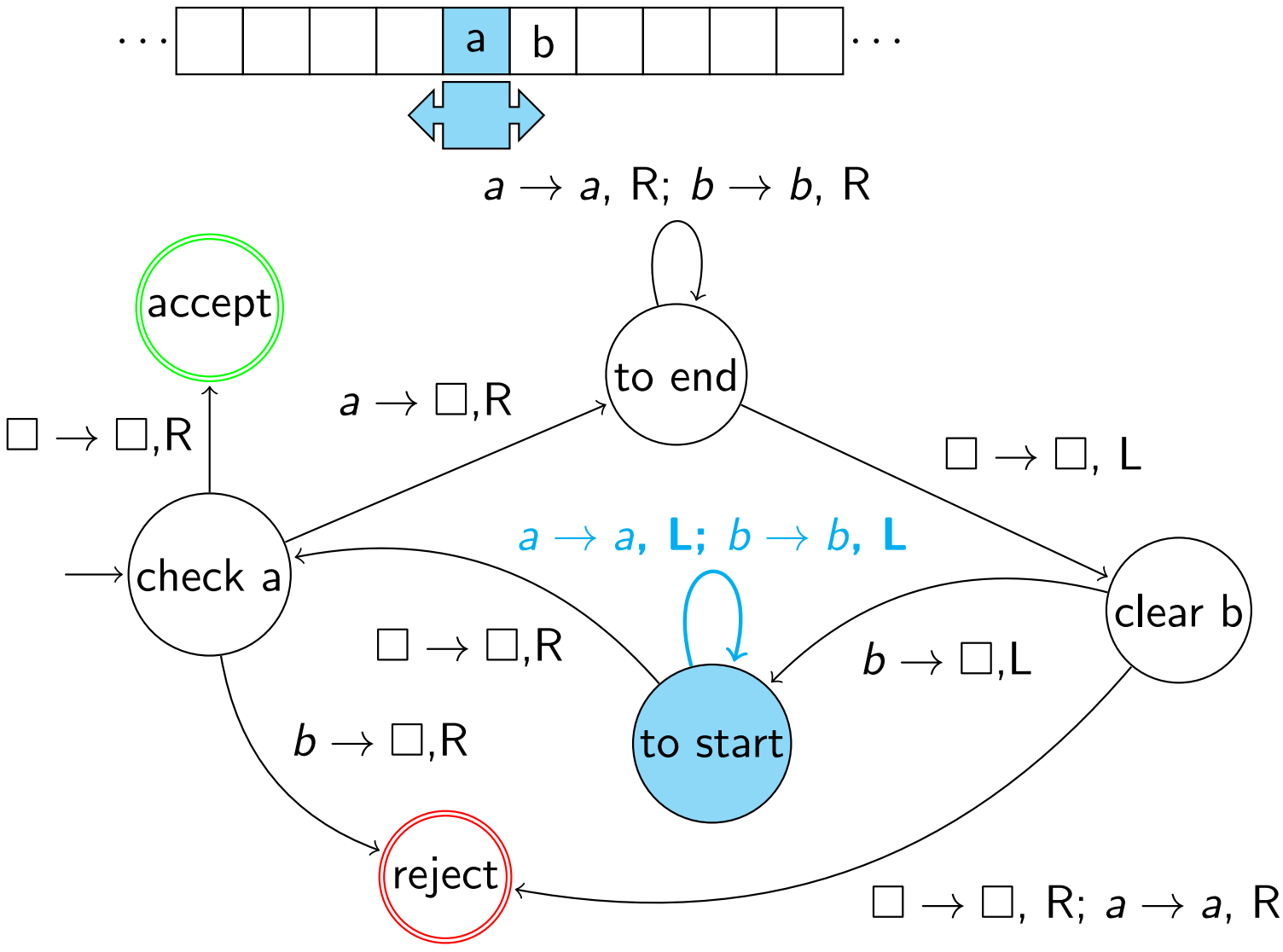
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



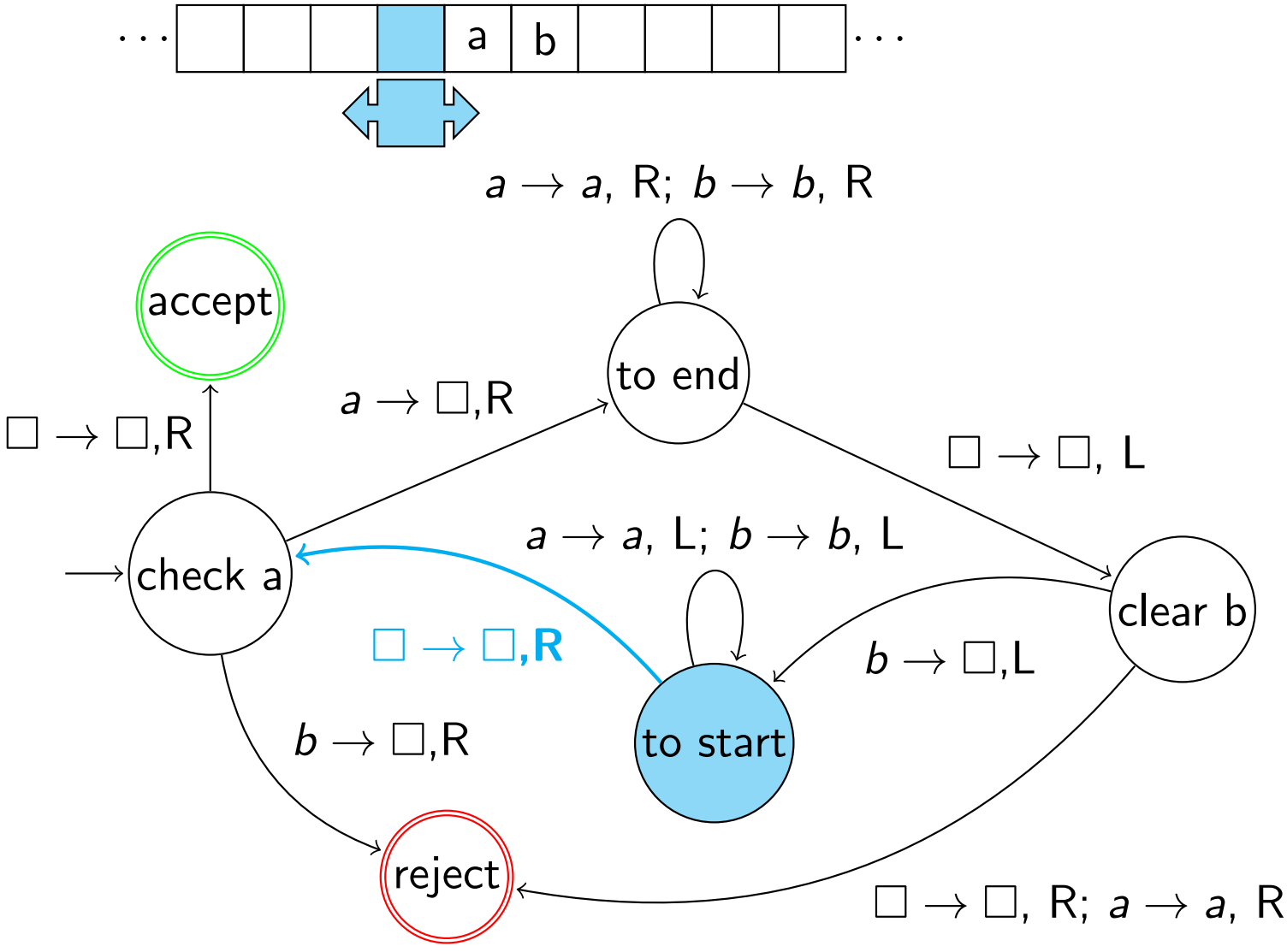
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



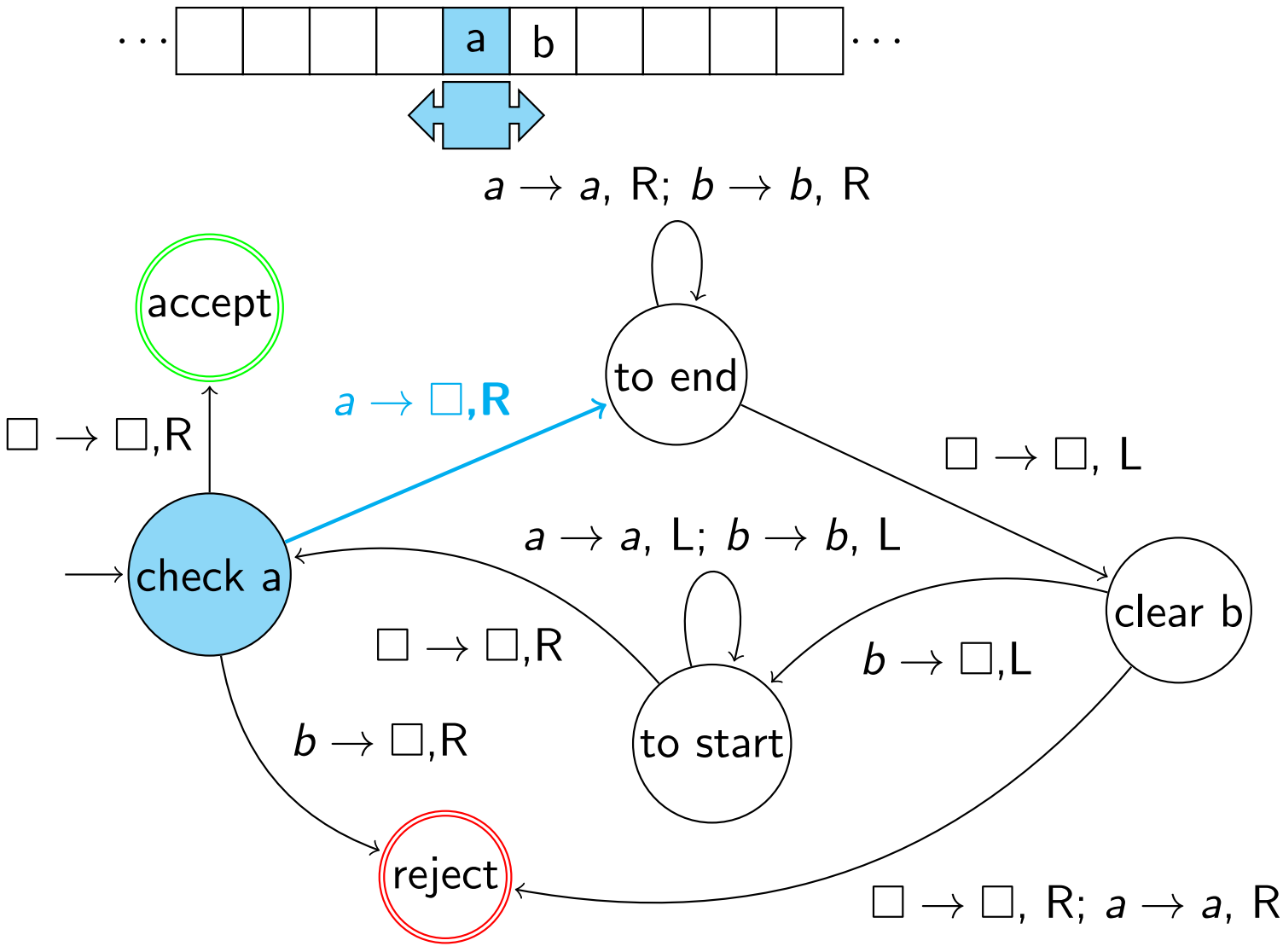
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



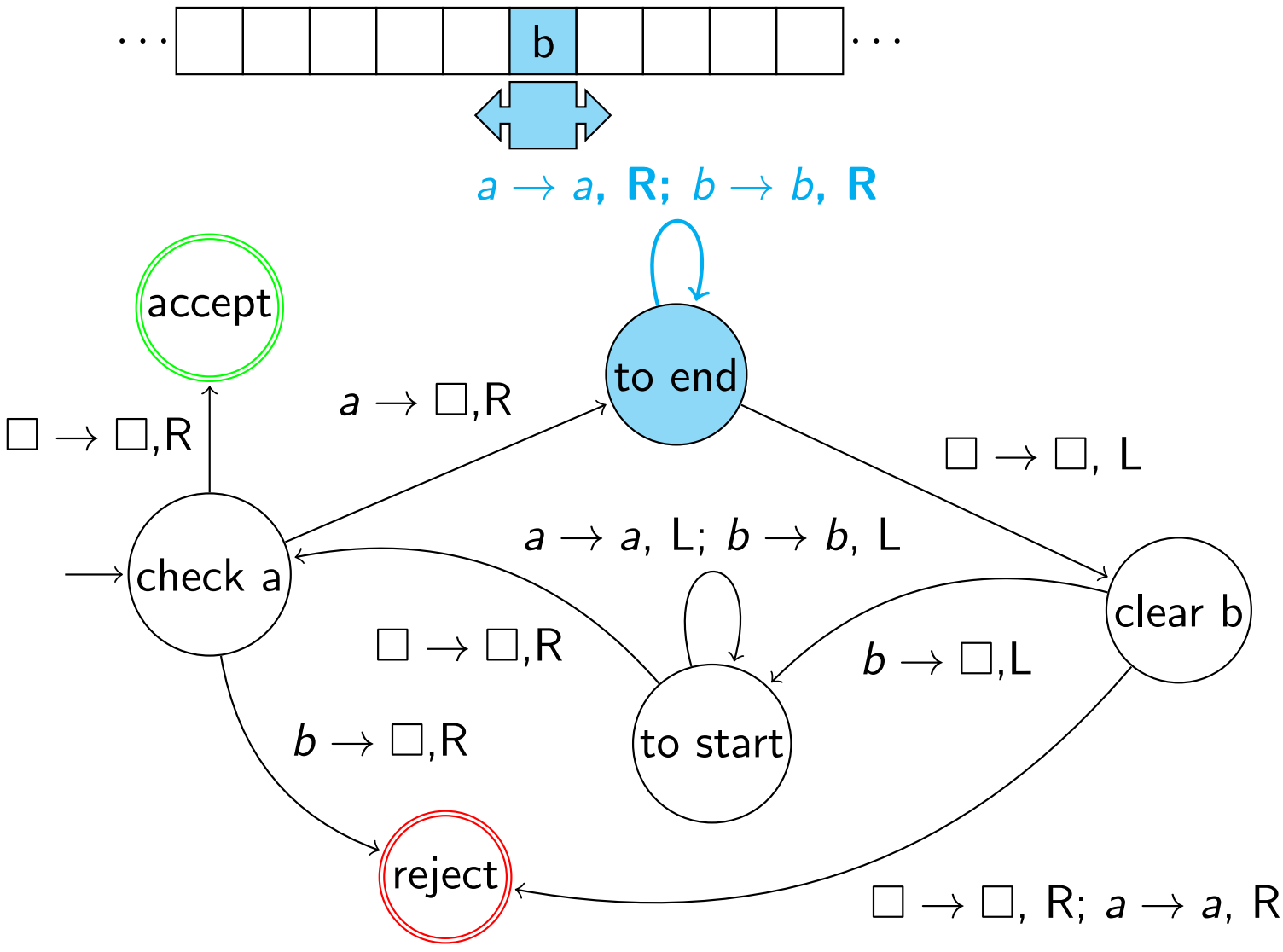
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



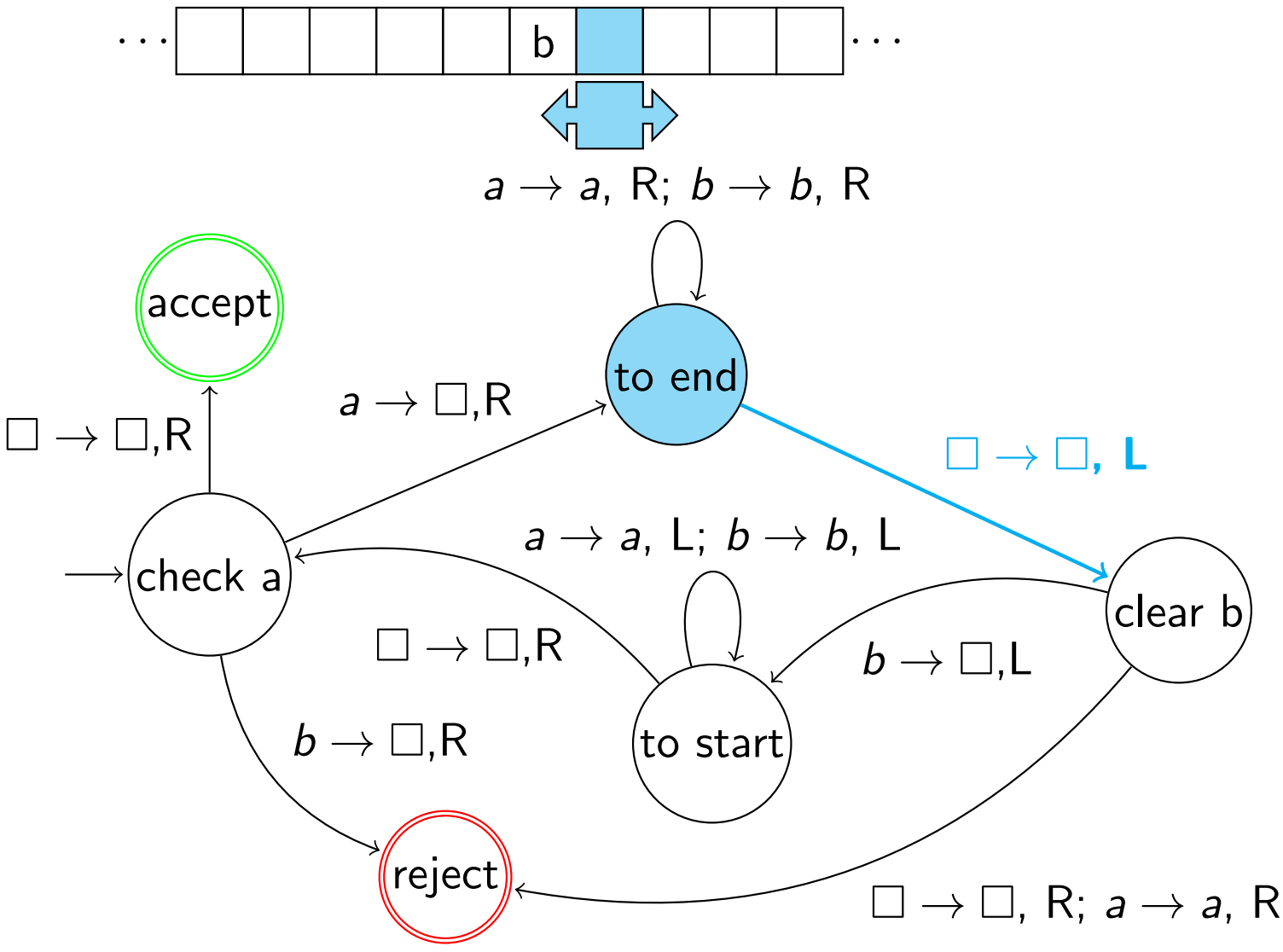
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



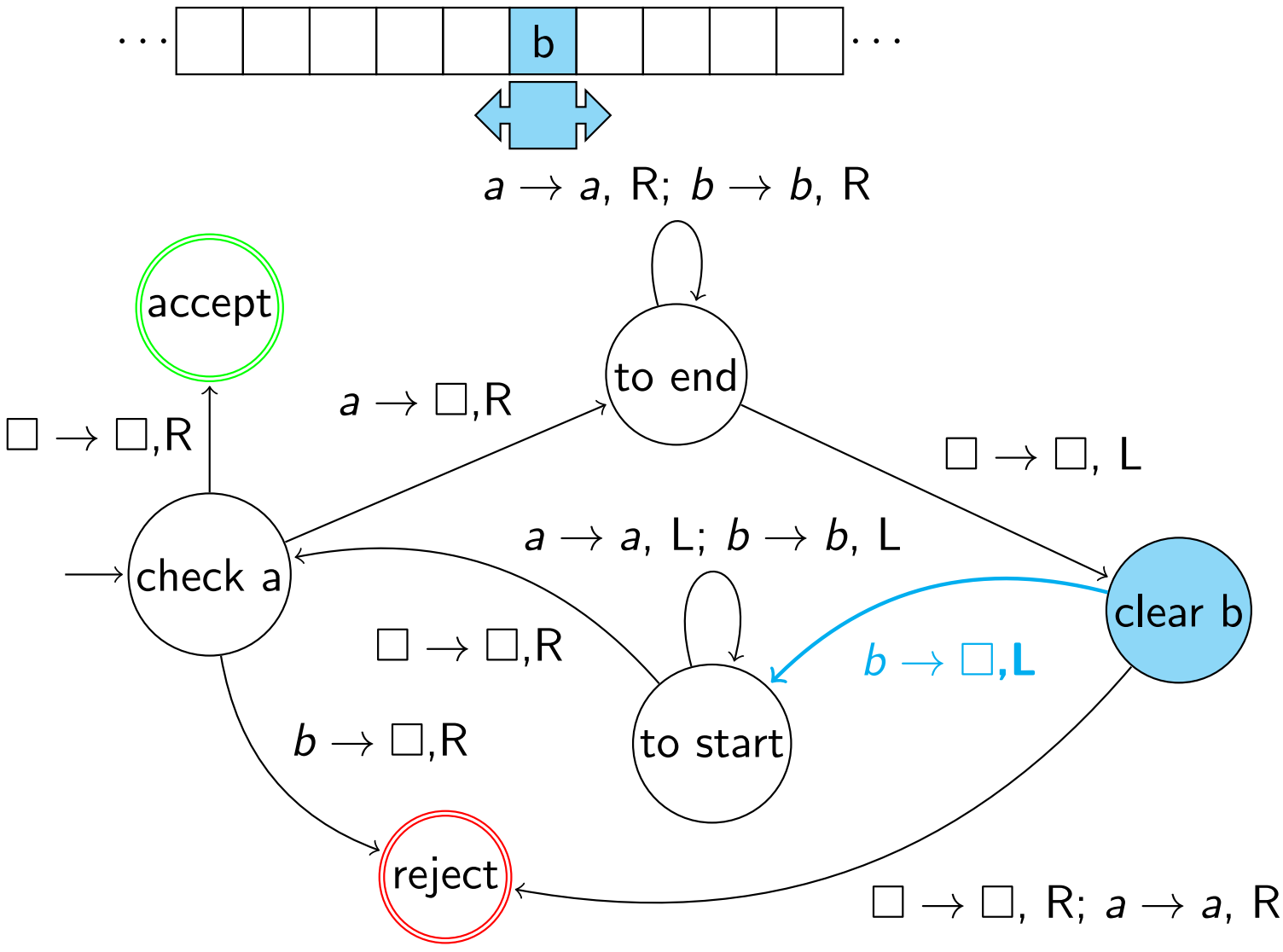
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



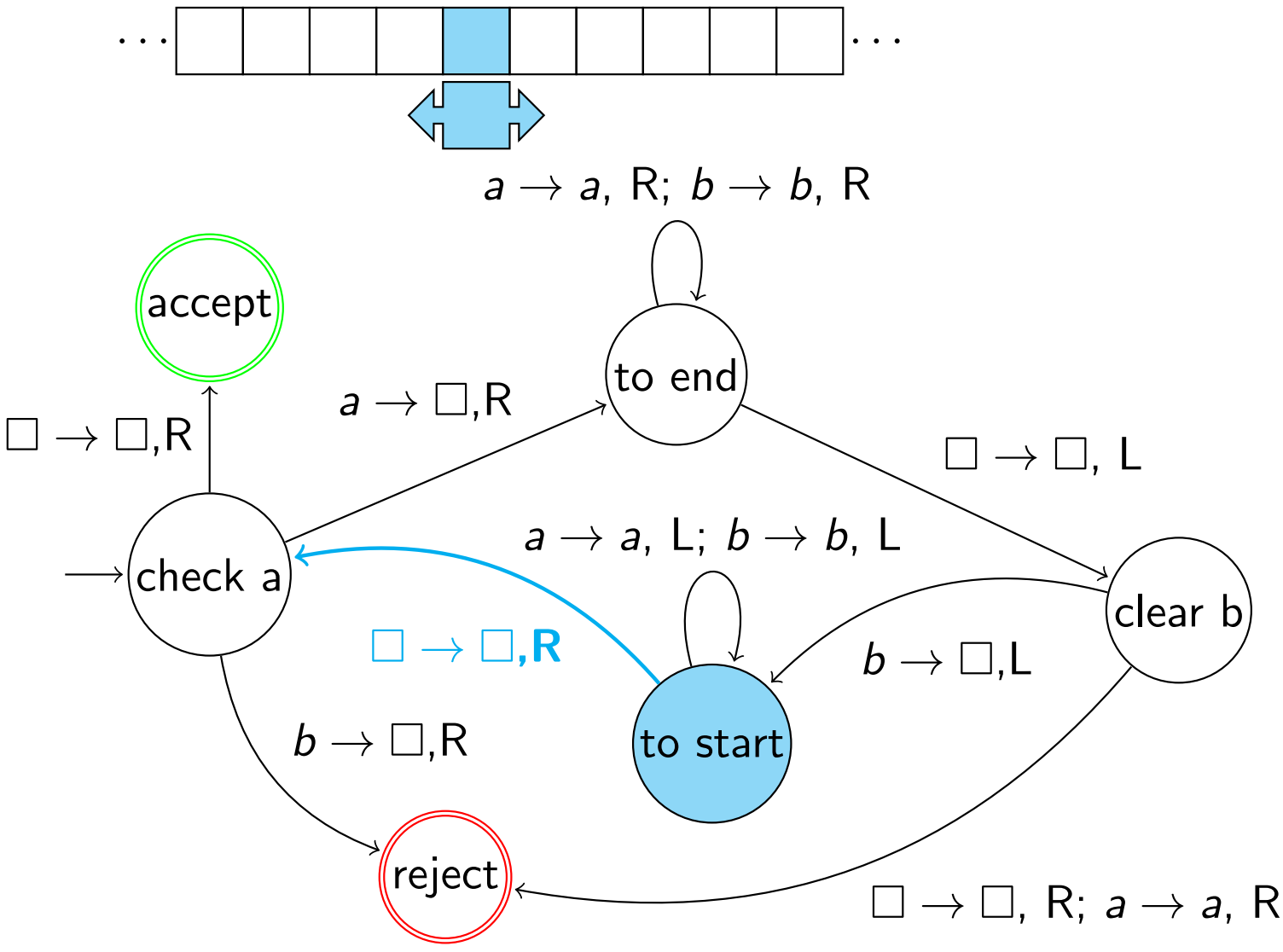
Exemplu: $aaabbb \in L=\{a^n b^n | n \in \mathbb{N}\}$



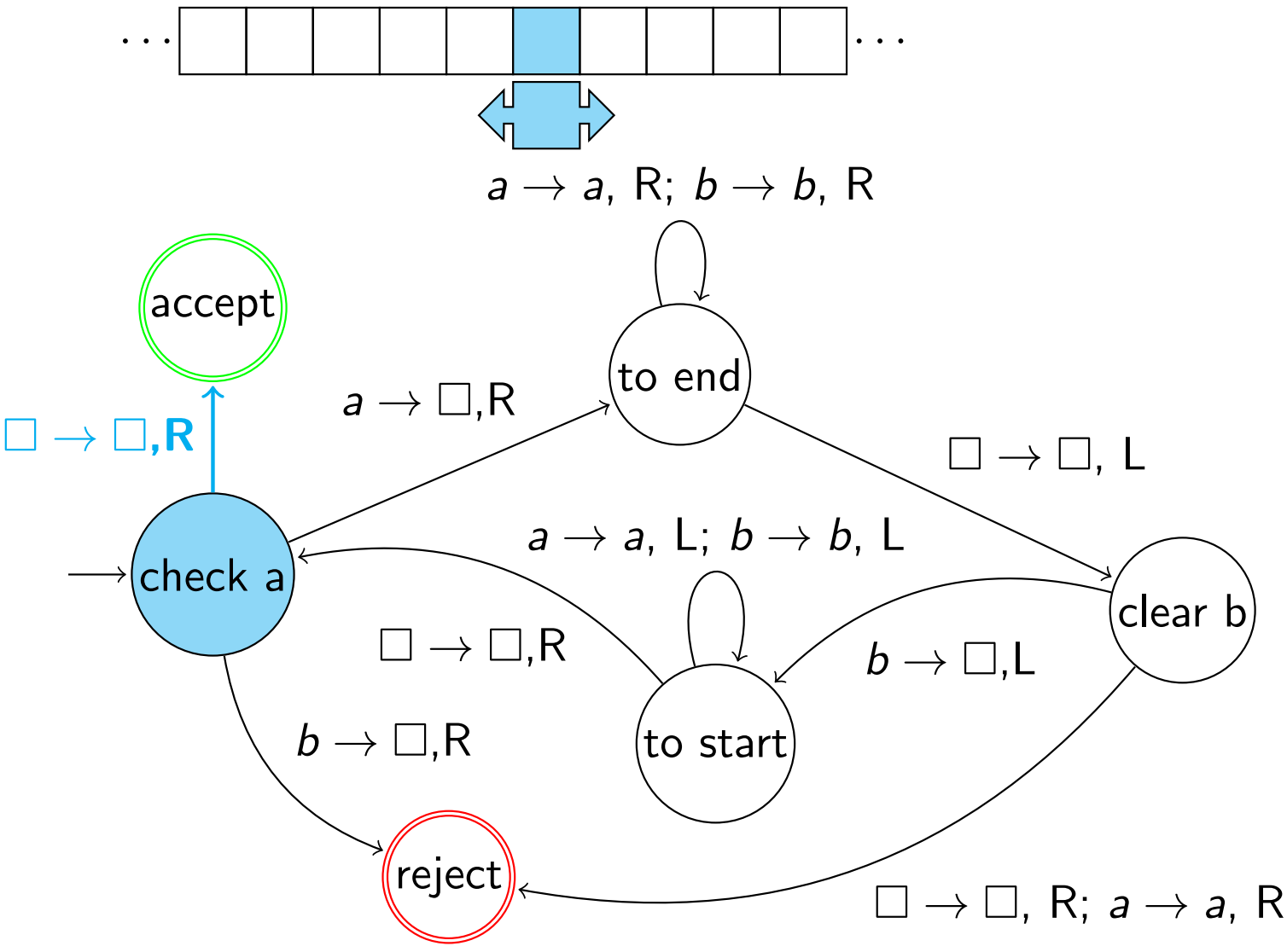
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



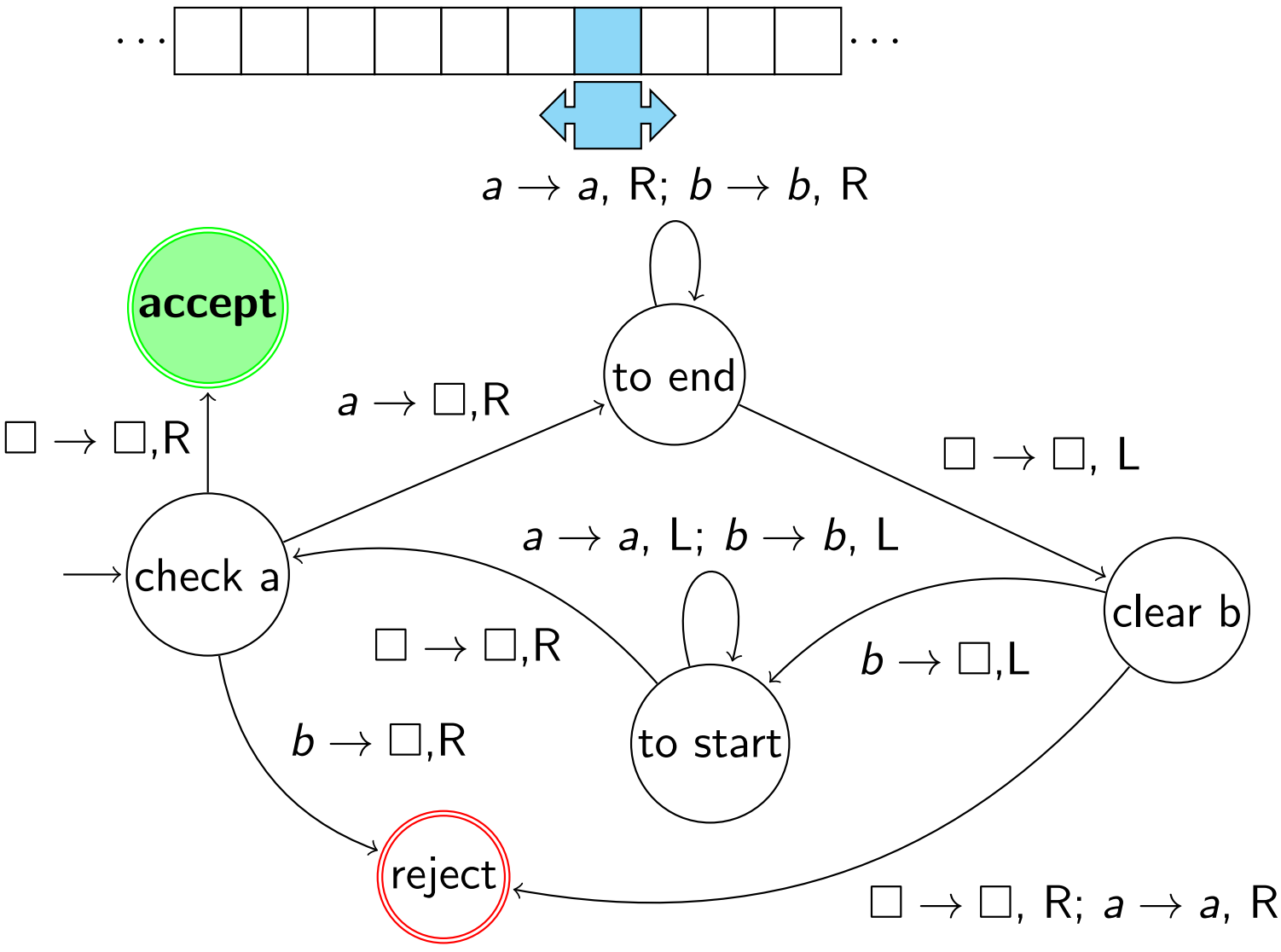
Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



Exemplu: $aaabbb \in \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



Important:

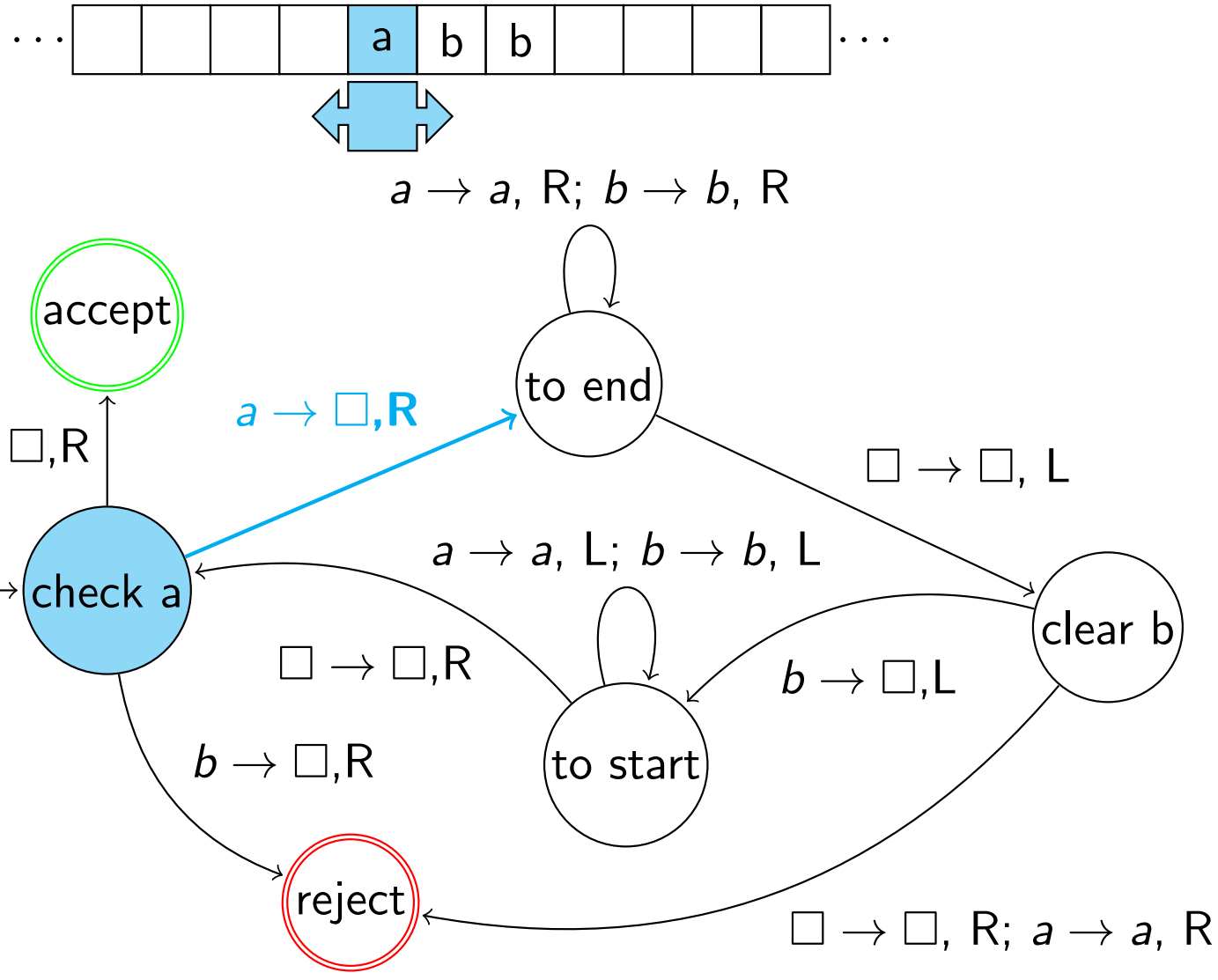
Spre deosebire de un automat DFA/NFA, o mașină Turing **nu** *se oprește* la terminarea șirului de intrare!

Execuția continuă până când se ajunge într-una din *stările finale*:

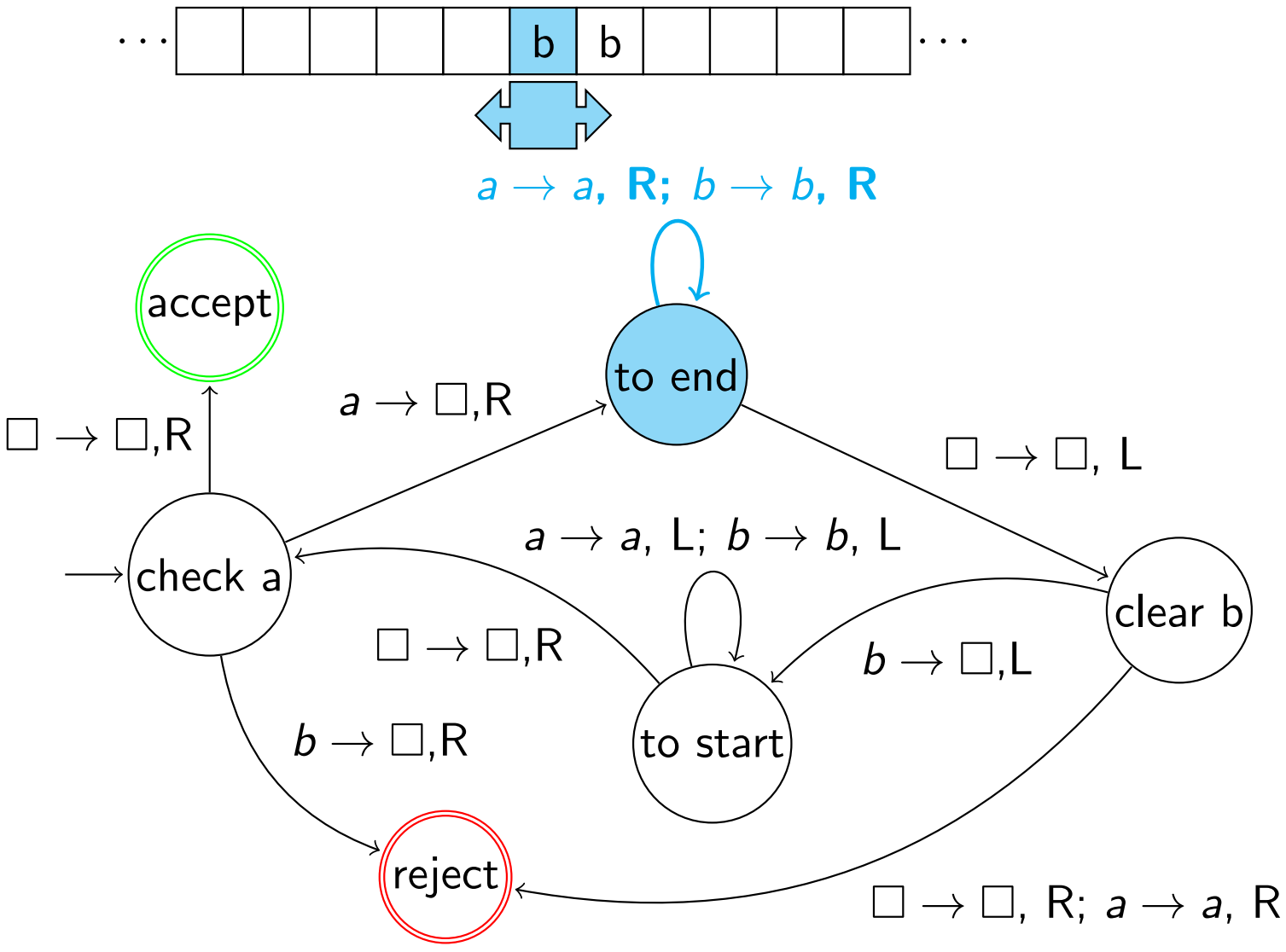
de **acceptare**: șirul este acceptat, face parte din limbaj

de **rejectare**: șirul este respins, nu face parte din limbaj

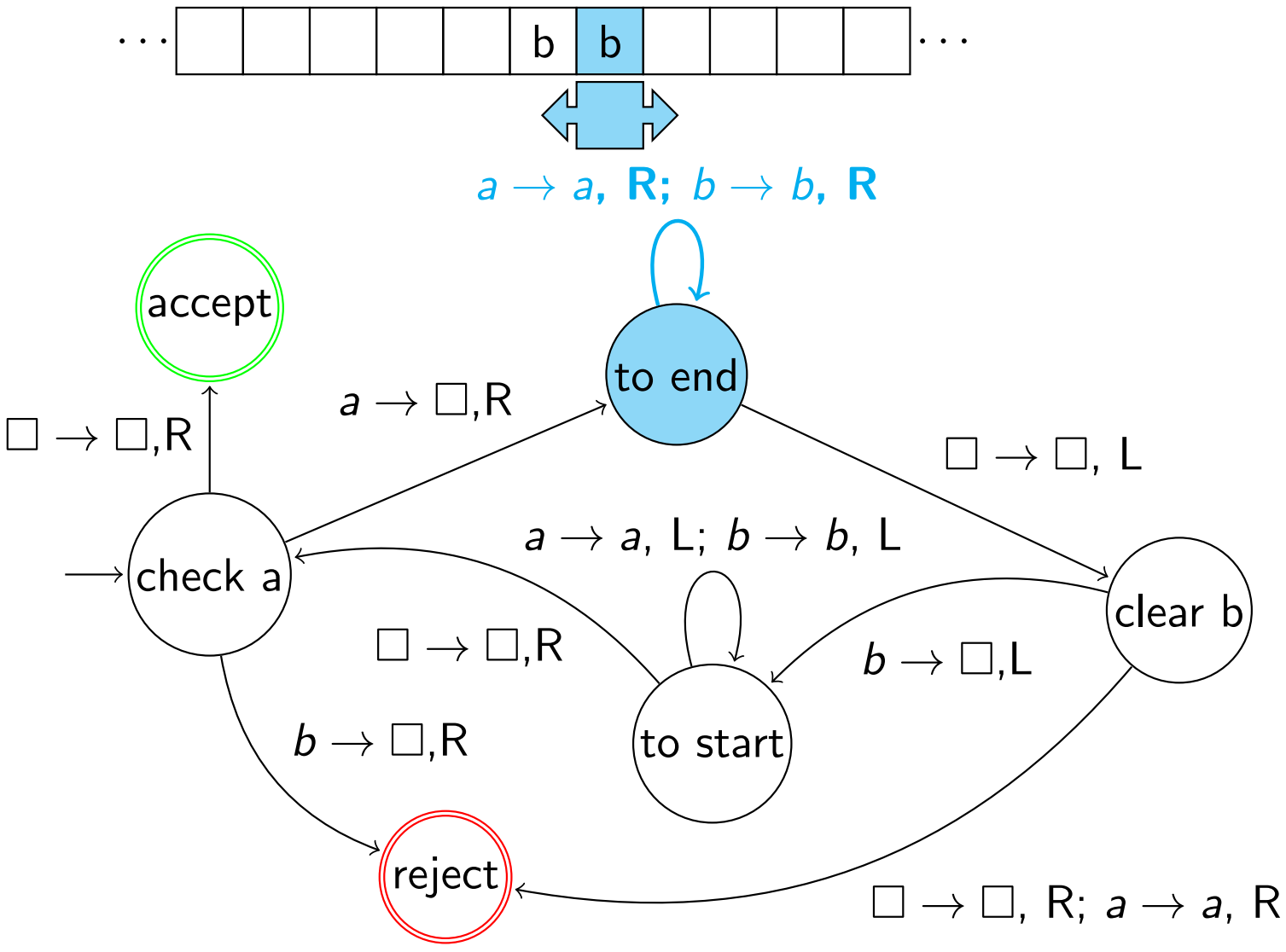
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



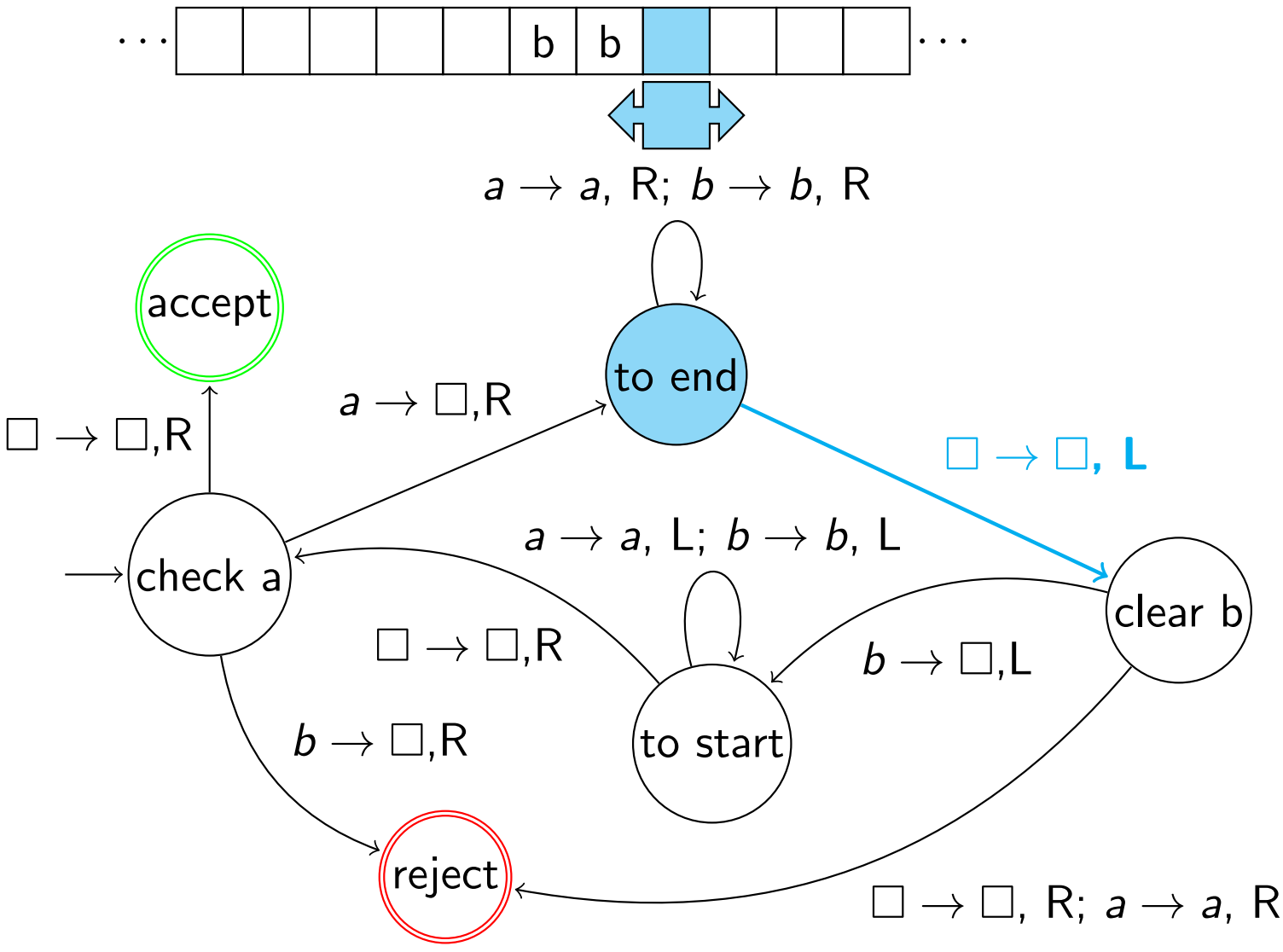
Exemplu: $abb \notin L=\{a^n b^n | n \in \mathbb{N}\}$



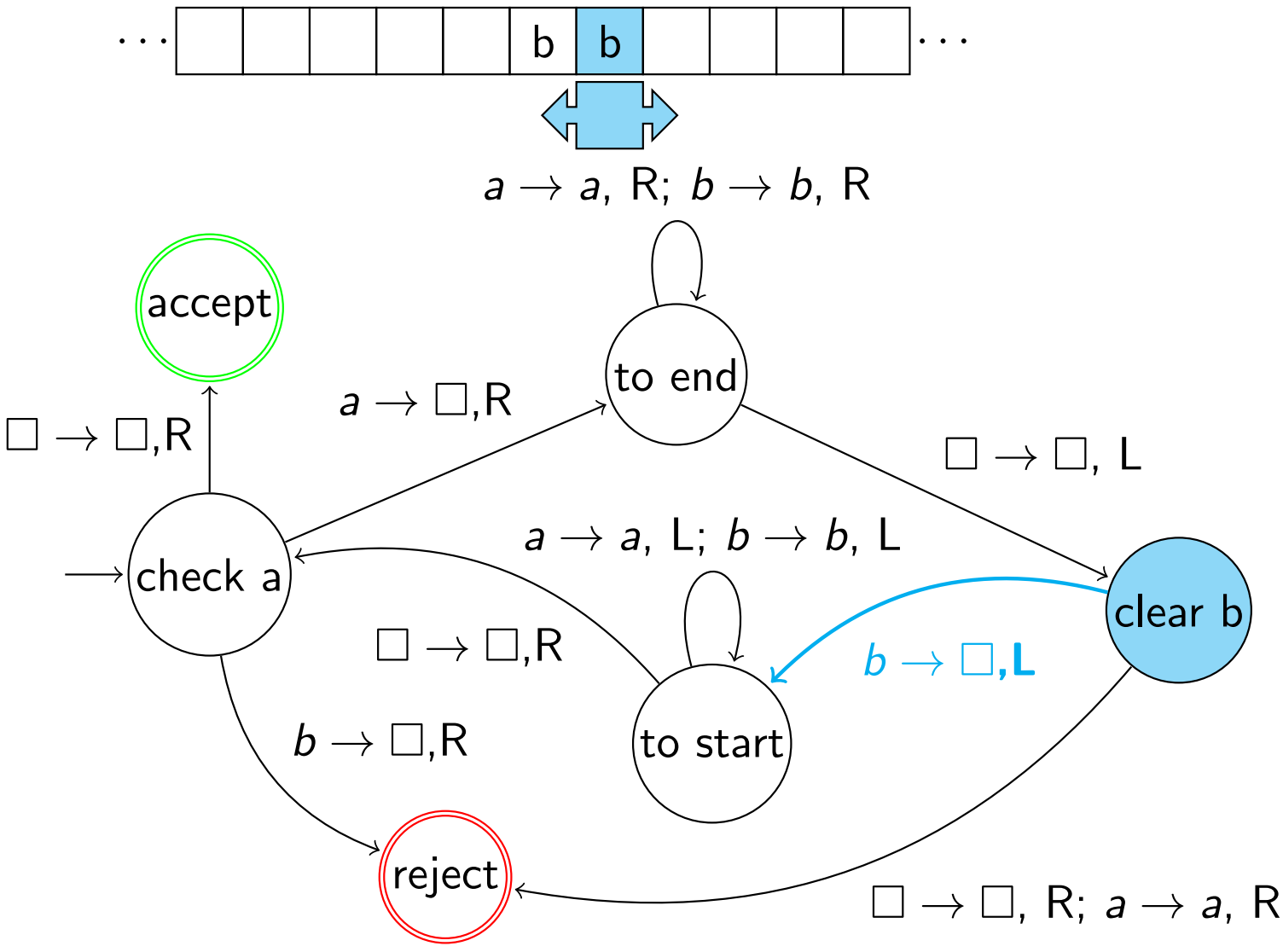
Exemplu: $abb \notin L=\{a^n b^n | n \in \mathbb{N}\}$



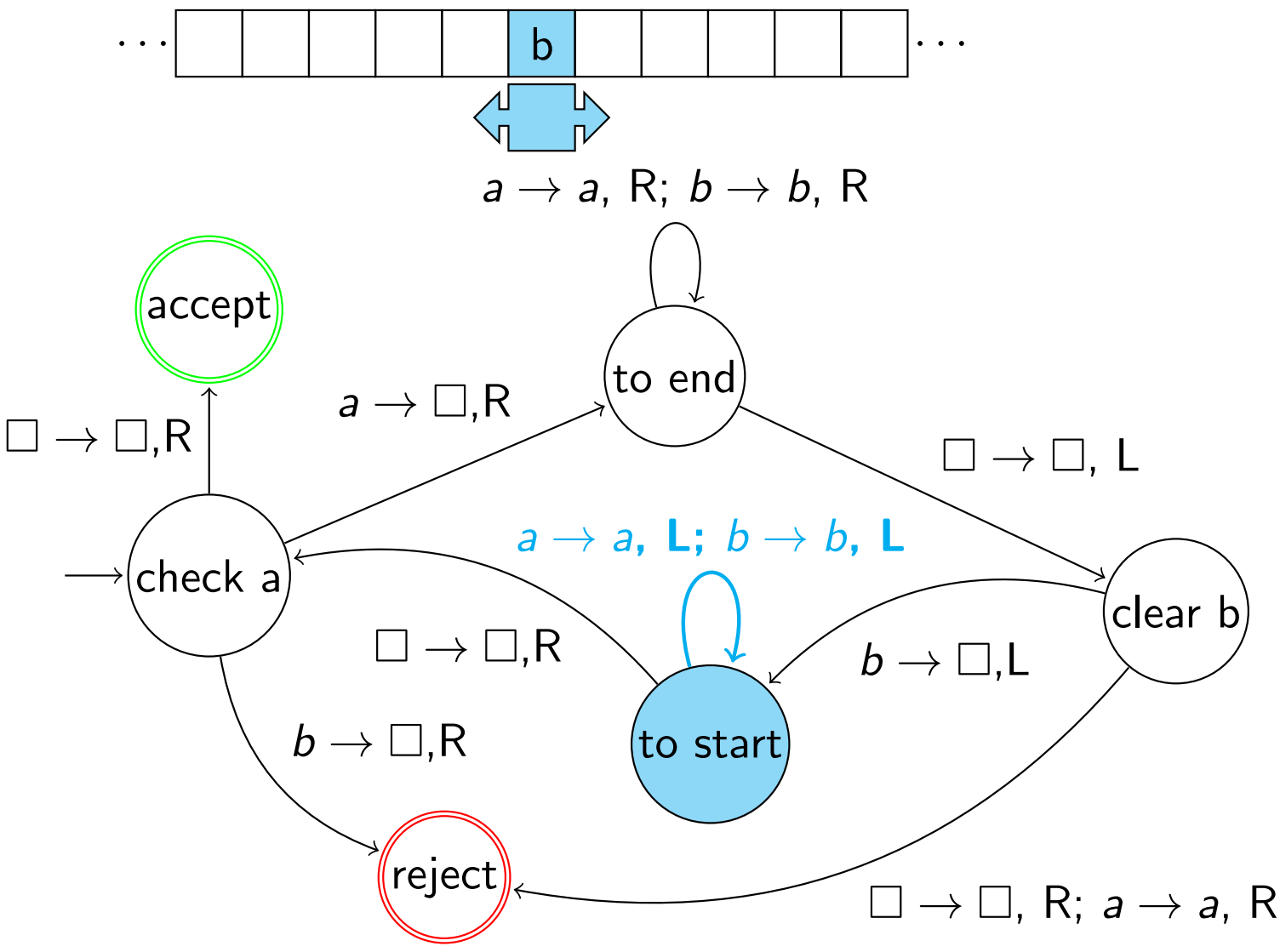
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



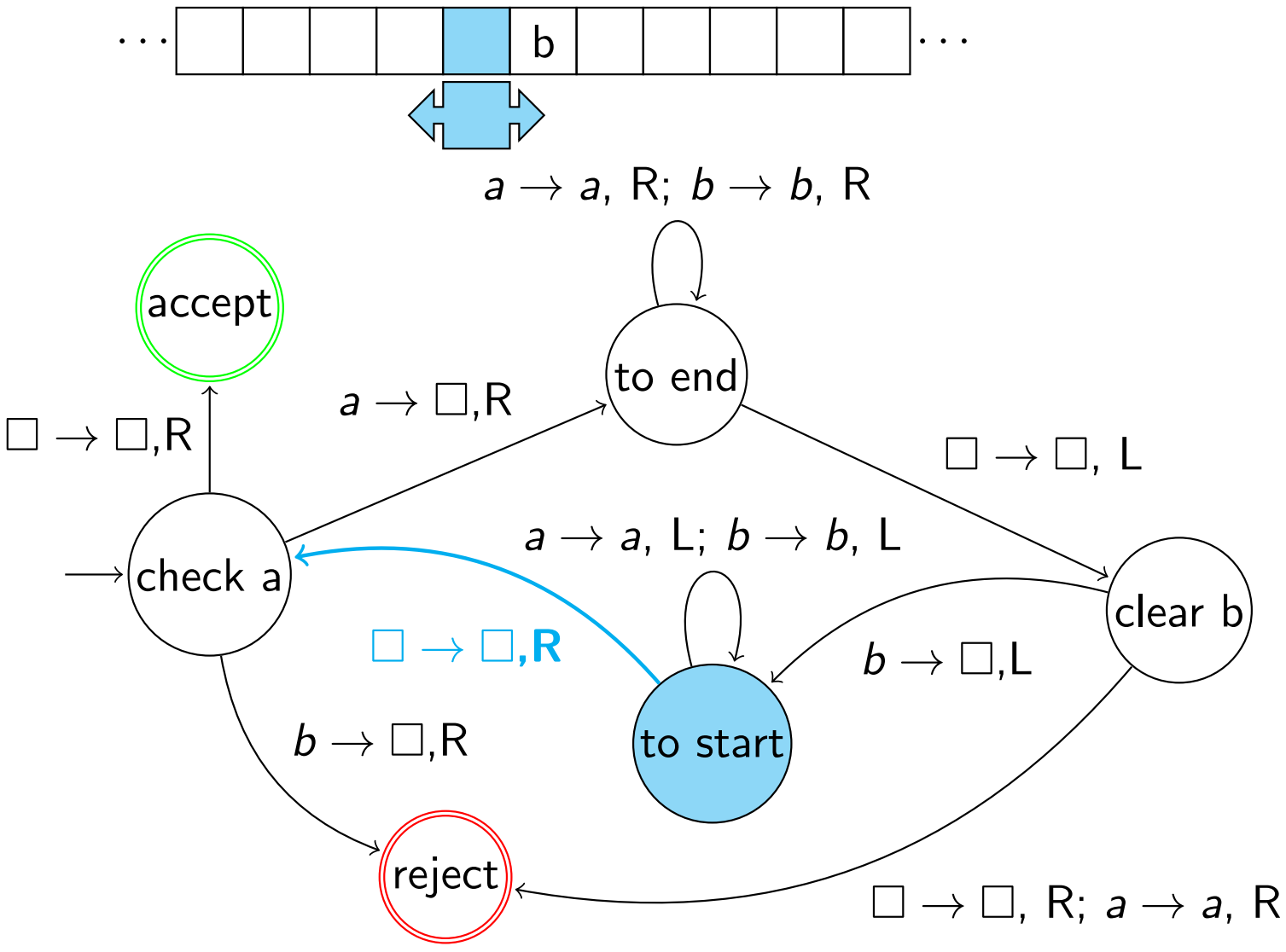
Exemplu: $abb \notin \mathrm{L}=\{a^n b^n | n \in \mathbb{N}\}$



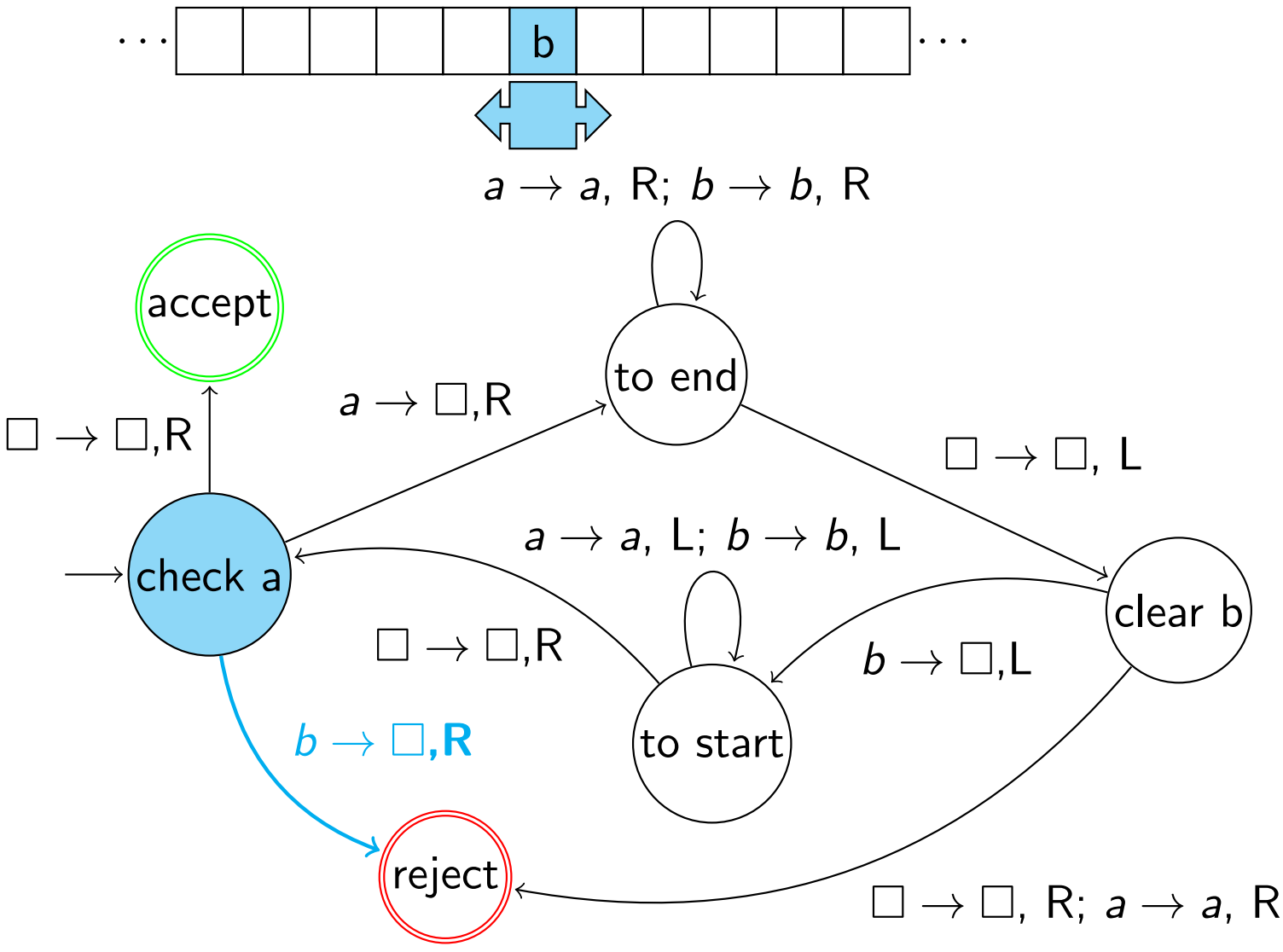
Exemplu: $abb \notin L=\{a^n b^n | n \in \mathbb{N}\}$



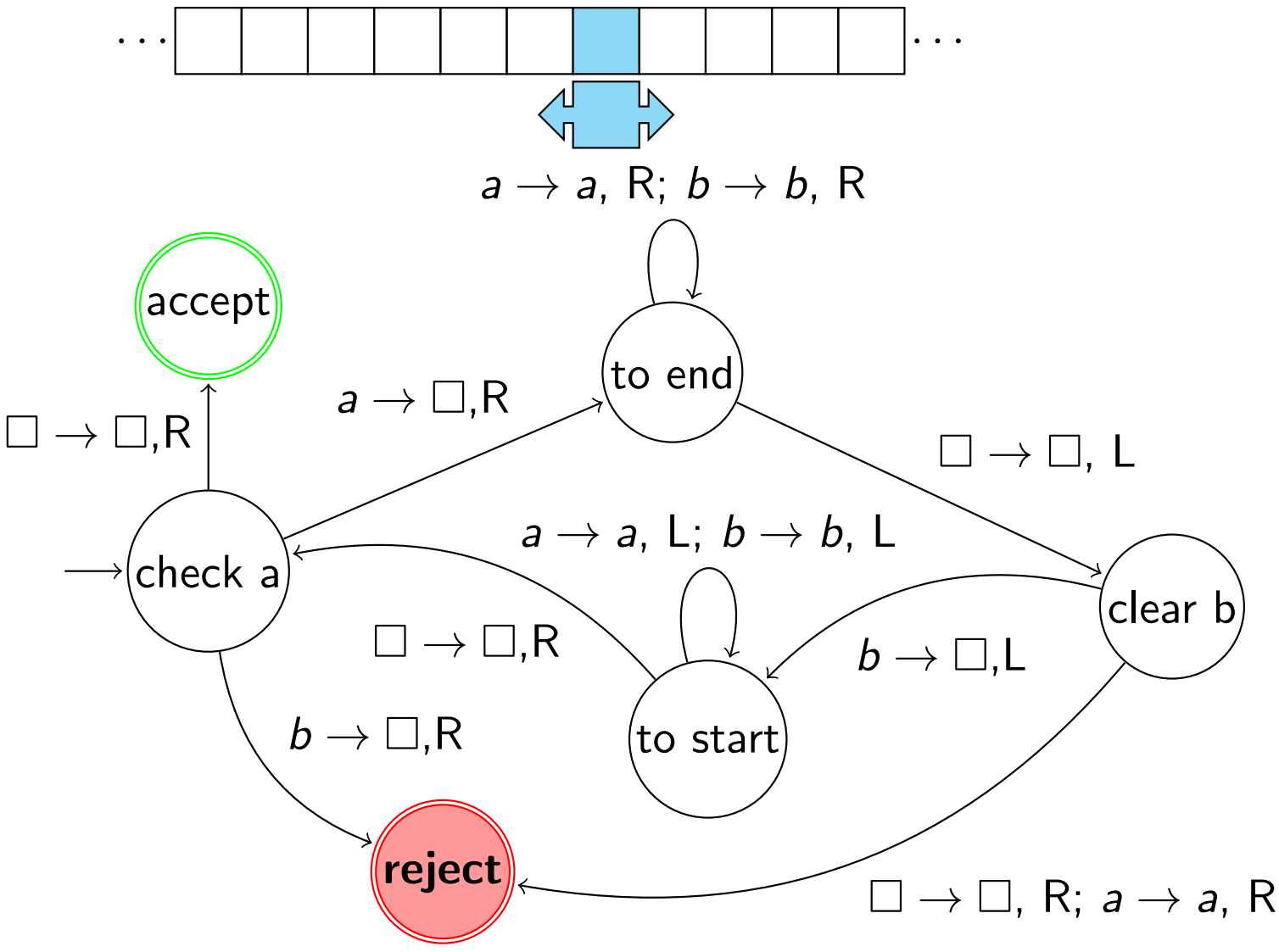
Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



Exemplu: $abb \notin L = \{a^n b^n | n \in \mathbb{N}\}$



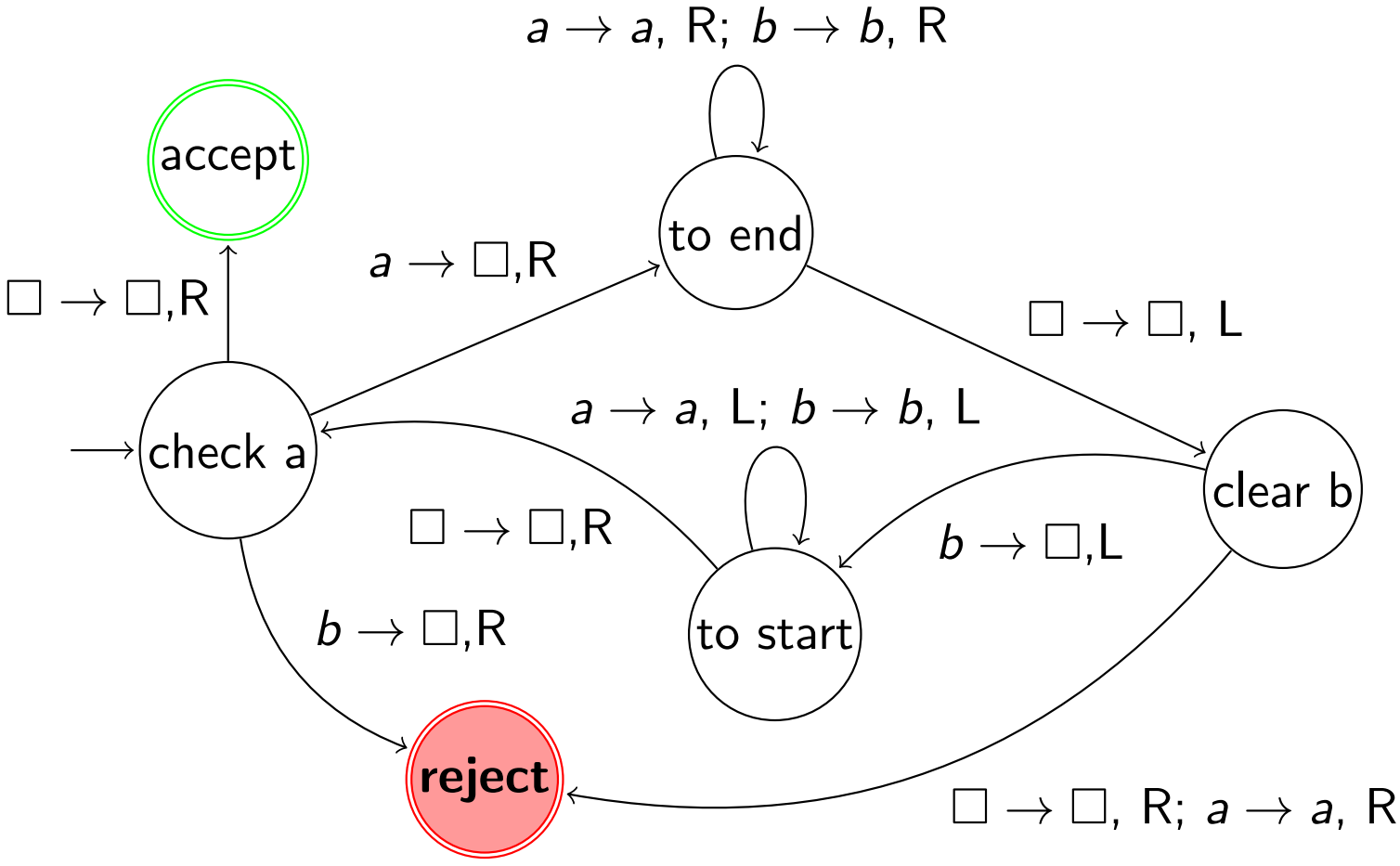
Exemplu: $abb \notin L=\{a^n b^n | n \in \mathbb{N}\}$



Obs.: mașinile Turing sunt deterministe!

Pentru fiecare combinație de stare non-finală q și simbol de bandă γ, există o *unică* tranziție $\delta(q, \gamma)$

(tranzițiile lipsă, dacă există, duc implicit în *starea de rejectare*)



Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)

Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)

Γ: mulțimea simbolurilor de pe bandă (care pot fi scrise);
important: $\Sigma \subset \Gamma$ (Γ are cel puțin un simbol în plus, $\square$)

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
funcția de tranziție:
dă starea următoare,
simbolul cu care e înlocuit cel curent,
și mutarea la stânga sau dreapta
(în unele versiuni, echivalente, capul poate și rămâne pe loc)

$q_0 \in Q$: starea inițială a automatului de control

$\square \in \Gamma \setminus \Sigma$: simbolul vid (blanc) ($\square \notin \Sigma$)
toate celulele cu excepția unui număr finit sunt inițial vide

$F \subseteq Q$: mulțimea stărilor finale, automatul se oprește (halt)

Important:

Spre deosebire de un automat DFA/NFA, o mașină Turing **nu se oprește** la terminarea șirului de intrare!

Execuția continuă până când se ajunge într-una din **stările finale**:

de **acceptare**: șirul este acceptat, face parte din limbaj

de **rejectare**: șirul este respins, nu face parte din limbaj

Există situații în care nu se ajunge **niciodată** într-o stare finală!!

Pentru anumite limbaje și anumite șiruri de intrare, mașina Turing poate intra într-un **ciclu infinit**, *fără să accepte sau să respingă vreodată* șirul de intrare primit!

Mașini Turing de tip subrutină

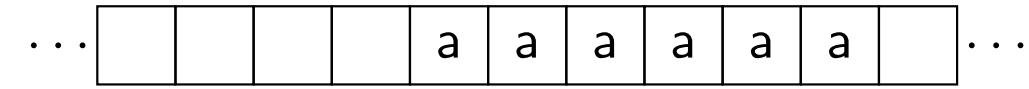
O mașină Turing este *de tip subrutină* dacă, în loc să accepte sau să respingă un șir de intrare, *efectuează anumite transformări* asupra acestuia

după care intră într-o stare finală (marcată *done* sau *halt*).

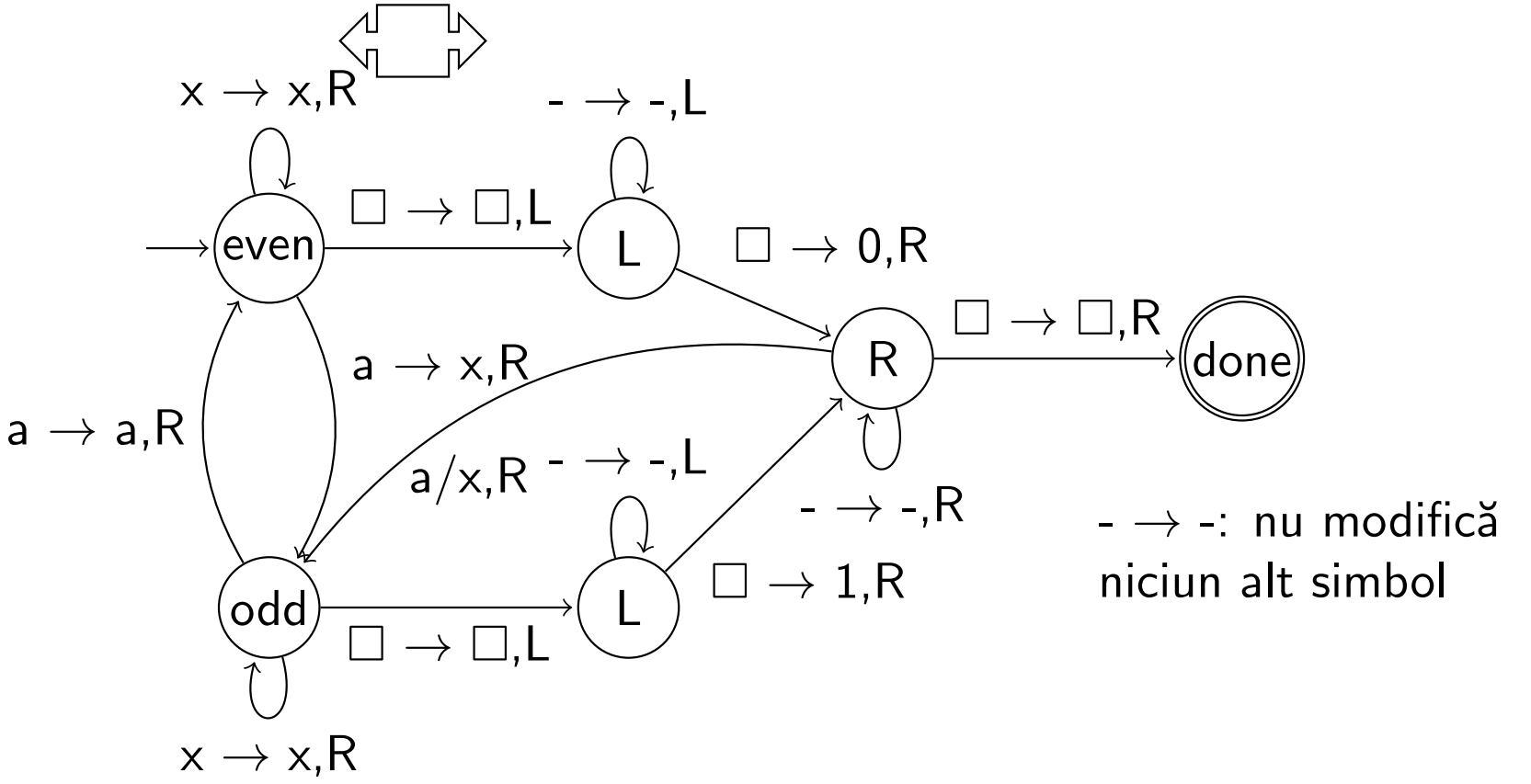
Astfel, pe bandă rezultă un nou șir, care poate fi prelucrat mai departe sau acceptat/respins de o altă mașină Turing.

Mașinile Turing subrutină pot fi folosite pentru a *compune* mașini Turing mai *complexe* din mașini Turing mai *simple*.

Exemplu: numără simboluri și scrie numărul în binar



câți a sunt pe bandă?



- → -: nu modifică niciun alt simbol

obține fiecare bit din nr. de a

--→: schimbă a cu x din 2 în 2

←--: scrie 0 sau 1 după paritate

repetă până ∄ a: *done*

□□□□aaaaaa□ --→ □□□□xaxaxa□

←-- □□□0xaxaxa□ --→ □□□0xxxaxx□

←-- □□10xxxaxx□ --→ □□10xxxxxx□

←-- □110xxxxxx□ --→ □110xxxxxx□

done

Cât de puternică e o mașină Turing?

Ce putem calcula cu ajutorul ei?

Conceptual, un calculator ideal e un calculator cu memorie nelimitată (RAM, HDD/SSD, etc.).

Un calculator ideal *poate simula* o mașină Turing (poate face tot ce face și aceasta)

O mașină Turing *poate simula* un calculator ideal (poate face aceleași lucruri: aritmetica, cicluri, decizii, variabile, etc.) !!

Practic poate descrie *orice calcul* (implementabil prin program)

Metodă de calcul efectivă

O metodă de calcul efectivă este un *sistem computațional* cu următoarele proprietăți:

- calculul consistă dintr-o *serie de pași*
- există *reguli fixe* conform cărora un pas e urmat de un altul
- orice calcul care produce un rezultat va ajunge la acesta într-un *număr finit de pași*
- orice calcul care ajunge la un rezultat ajunge la un rezultat *corect*

Calculabilitate – Teza Church-Turing

Ce se poate *calcula*, și cum putem defini această noțiune ?

Teza Church-Turing (o afirmație despre noțiunea de *calculabilitate*)

Orice metodă de calcul efectivă este *echivalentă cu*,
sau *mai slabă* decât o mașină Turing.

Următoarele modele de calcul sunt echivalente:

- lambda-calculul
- mașina Turing
- funcțiile recursive

Lambda-calcul

Definit de Alonzo Church (1932); poate fi privit ca fiind cel mai simplu limbaj de programare

O expresie în lambda-calcul e fie:

- o *variabilă* — x
- o *funcție* — $\lambda x\,.\,e$ (funcție de variabilă x)
  în ML: `fun x -> e`
- o *evaluare* de functie — $e_1\ e_2$ (funcția e_1 aplicată argumentului e_2)
  la fel în ML: `f x` fără paranteze

Toate noțiunile fundamentale (numere naturale, booleni, perechi, decizie, recursivitate, etc.) pot fi exprimate în lambda-calcul.

Decidabilitate

Terminologie

Fie M o mașină Turing. Atunci, spunem că:

- M **acceptă** un șir w dacă ajunge într-o stare acceptoare în urma execuției pe w
- M **respinge/rejectează** un șir w dacă ajunge într-o stare rejectoare în urma execuției pe w
- M **ciclează infinit** pe un șir w dacă, în urma execuției pe w, nu ajunge nici într-o stare de acceptare, nici într-una de rejectare
- M **nu acceptă** w dacă îl rejectează sau ciclează infinit pe w
- M **nu rejectează** w dacă îl acceptă sau ciclează infinit pe w
- M **se oprește** pe w dacă îl acceptă sau îl rejectează

Limbajul unui mașini Turing

Limbajul unui mașini Turing M, notat $\mathcal{L}(M)$ este mulțimea tuturor șirurilor *acceptate* de M.

$$\mathcal{L}(M) = \{w \in \Sigma^* \mid M \textit{ accepts } w\}$$

Pt. orice $w \notin \mathcal{L}(M)$, M nu acceptă w (respinge / ciclează infinit).

Limbaje recognoscibile

Un limbaj L este *recognoscibil* dacă există o mașină Turing M astfel încât $L = \mathcal{L}(M)$ (L este limbajul lui M).

O mașină Turing M astfel încât $L = \mathcal{L}(M)$ este un *recognizer* pentru limbajul L.

Clasa **RE** reprezintă mulțimea tuturor limbajelor recognoscibile

$$L \in \mathbf{RE} \Leftrightarrow \exists M.\, L = \mathcal{L}(M)$$

(unde M e o mașină Turing)

Limbaje decidabile

Dacă M este o mașină Turing și M se oprește (acceptă sau rejectează) pentru fiecare șir de intrare primit, spunem că M este un *decident*.

Un limbaj L este *decidabil* dacă există o mașină Turing decidentă M astfel încât $L = \mathcal{L}(M)$.

Pt. orice $w \in \mathcal{L}(M)$, M acceptă w.
Pt. orice $w \notin \mathcal{L}(M)$, M rejectează w .

Clasa **R** reprezintă mulțimea tuturor limbajelor decidabile

$$L \in \mathbf{R} \Leftrightarrow L \textit{ decidabil}$$

Mașina Turing universală

Teoremă (Turing, 1936)

Există o mașină Turing universală U_{TM} care, dacă este rulată pe o intrare de forma $\langle M, w \rangle$, unde M este o mașină Turing și w este un șir, *simulează* execuția lui M pe w, acceptând, rejectând sau ciclând infinit, după cum M acceptă, rejectează sau ciclează infinit pe șirul w.

- M acceptă $w \Rightarrow U_{TM}$ acceptă $\langle M, w \rangle$
- M respinge $w \Rightarrow U_{TM}$ respinge $\langle M, w \rangle$
- M ciclează infinit pe $w \Rightarrow U_{TM}$ ciclează infinit pe $\langle M, w \rangle$

U_{TM} acceptă $\langle M, w \rangle \Leftrightarrow M$ acceptă w

Mașina Turing universală. Limbajul lui U_{TM}

... e, practic, o mașină Turing *programabilă*, astfel încât să poată recunoaște *orice* limbaj *recognoscibil*

U_{TM} poate efectua *orice calcul* care poate fi efectuat de către *orice dispozitiv de calcul fezabil*

U_{TM} acceptă $\langle M, w\rangle \Leftrightarrow M$ acceptă w

Limbajul mașinii Turing universale:

$$\mathcal{L}(U_{TM}) = \{\langle M, w\rangle \mid M \textit{ masina Turing } \wedge\ w \in \mathcal{L}(M)\}$$

Notăm $A_{TM} = \mathcal{L}(U_{TM})$ limbajul lui U_{TM}.

Este A_{TM} decidabil?

U_{TM} este un *recognizer* pt. A_{TM}, deci $A_{TM} \in \mathbf{RE}$ (recognoscibil)

Presupunem că $A_{TM} \in \mathbf{R}$ (limbaj decidabil)

$\Rightarrow$ există o mașină Turing DU_{TM} *decidentă* pt. A_{TM}

$\Rightarrow$ pt. un input $\langle M, w \rangle$ DU_{TM}:
acceptă dacă M acceptă w
rejectează dacă M nu acceptă w

Putem construi o mașină Turing *MWEIRD* care, folosind rezultatul returnat de DU_{TM}, să *rejecteze* acele șiruri w pentru care DU_{TM} a *acceptat* $\langle MWEIRD, w \rangle$, și să le accepte pe cele rejectate!!

$\Rightarrow$ Imposibil!! $\Rightarrow$ A_{TM} nedecidabil

A_{TM} nedecidabil

$\Rightarrow$ Nu există niciun algoritm care poate determina dacă o mașină Turing oarecare va accepta un șir

$\Rightarrow$ Singurul mod în care putem afla ce face un program oarecare pentru un input e să îl rulăm

A_{TM} nedecidabil

$A_{TM} \in \mathbf{RE}$ și $A_{TM} \notin \mathbf{R}$, deci $\mathbf{R} \subset \mathbf{RE}$ și $\mathbf{R} \neq \mathbf{RE}$
(există limbaje *recognoscibile* care nu sunt *decidabile*!!)

O problemă e în clasa **R** dacă există un *algoritm* pentru rezolvarea sa (se termină întotdeauna).

O problemă e în clasa **RE** dacă există un *semialgoritm* pentru rezolvarea sa (dacă răspunsul e "da" se termină, poate cicla la infinit altfel).

$\mathbf{R} \neq \mathbf{RE} \Rightarrow$ există situații când putem *verifica* dacă un răspuns e corect, dar nu avem un algoritm pt a-l *determina*

Nu tot ce e adevărat poate fi *descoperit* ca fiind adevărat...

Problema terminării (Halting Problem)

În formularea pentru programe:

Nu există algoritm (program) care ia un program arbitrar P și un set de date D și determină dacă $P(D)$ (rularea lui P cu datele D) s-ar termina (opri) sau ar rula la infinit.

Problema terminării – demonstrație

Presupunem că ar exista un astfel de program *CheckHalt(P, D)*.

Deci, *CheckHalt*(X, X) spune ce face prog. X cu *textul său* ca date

Construim un "program imposibil" care face opusul a ceea ce face!

Întâi, definim programul *Test(X)* având ca intrare un program *X*:

dacă *CheckHalt*(X, X) decide `"halt"`, atunci **ciclează la infinit**
dacă *CheckHalt*(X, X) decide `"ciclează"`, atunci **stop**

Deci *CheckHalt*(X, X) spune ce face X(X) iar *Test*(X) face opusul

Se oprește *Test*(Test)? Răspunsul e dat de *CheckHalt*(Test,Test).
dar *Test*(Test) (cu X=*Test*) face *opusul* lui *CheckHalt*(Test,Test)

$\Rightarrow$ *contradicție*, deci nu poate exista *CheckHalt*!

$\exists$ limbaje L a.î. $L \notin$ **RE**

... cu alte cuvinte, există limbaje *nerecognoscibile*

... care *nu* pot fi descrise de nicio gramatică (*negramaticale*)

... pentru care *nu* există nicio mașină Turing care să poată confirma că un șir w aparține limbajului L (pe *cazul general*)

Exemplu: Mulțimea tuturor mașinilor Turing care nu își acceptă propria descriere. (demo prin reducere la absurd: dacă am avea un recognizer R al limbajului rezultă $\langle R \rangle \in \mathcal{L}(R) \Leftrightarrow \langle R \rangle \notin \mathcal{L}(R)$...)

$\Rightarrow$ **există afirmații adevărate care nu pot fi dovedite!**
($\Leftrightarrow$ prima teoremă de incompletitudine a lui Gödel)