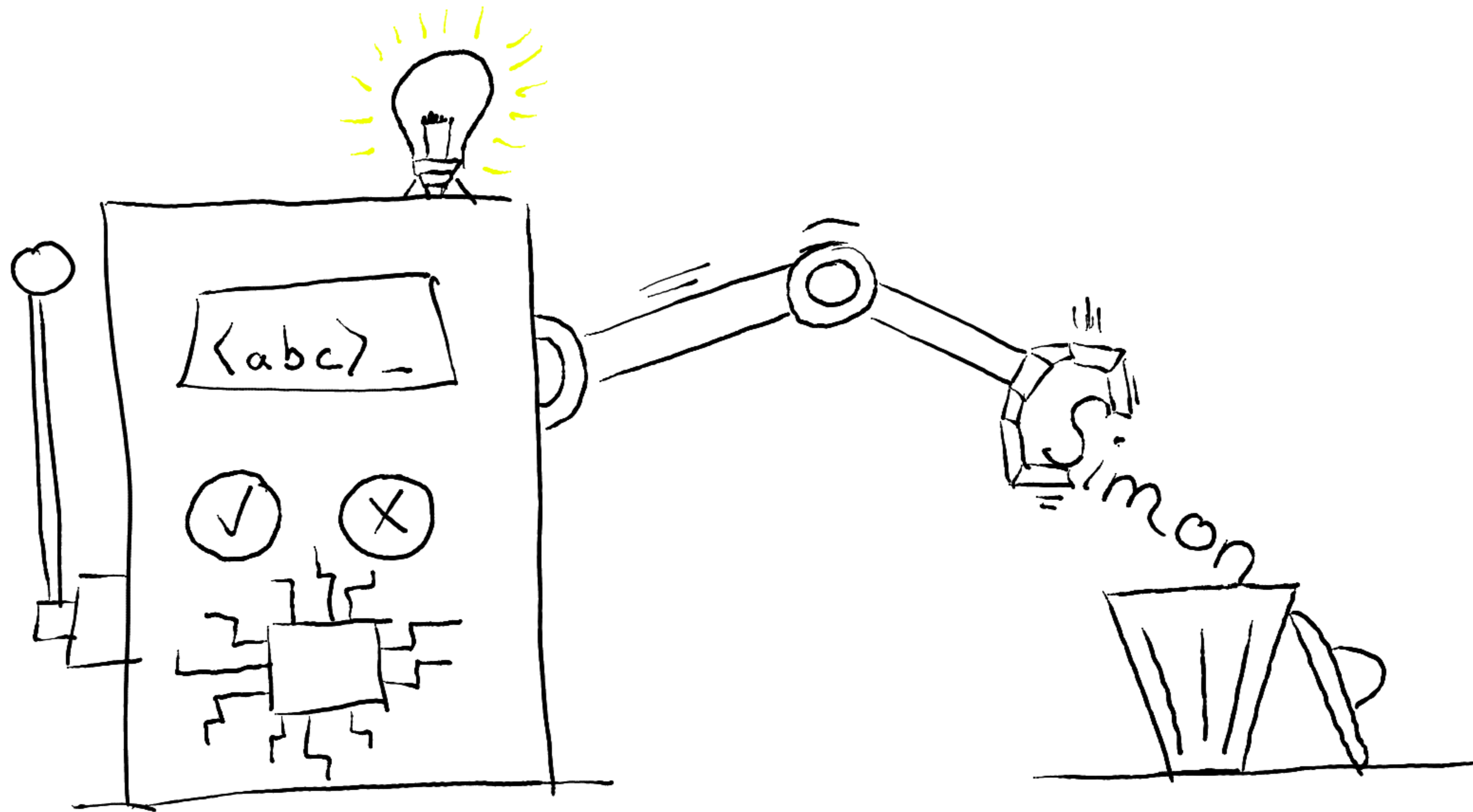


Nominal Automata with Name Deallocation



On my current work together with Lutz and Stefan

Motivation

Modern, high-level
programming languages:

- implicit
memory management
- garbage collection,
region inference, etc.
- allocation commands,
names bound to scoping
- e.g. Java, Python,
Scala, Haskell,
OCaml, ...

Motivation

Modern, high-level programming languages:

- implicit memory management
- garbage collection, region inference, etc.
- allocation commands, names bound to scoping
- e.g. Java, Python, Scala, Haskell, OCaml, ...

(And then there's Rust doing some weird ownership stuff)

However...

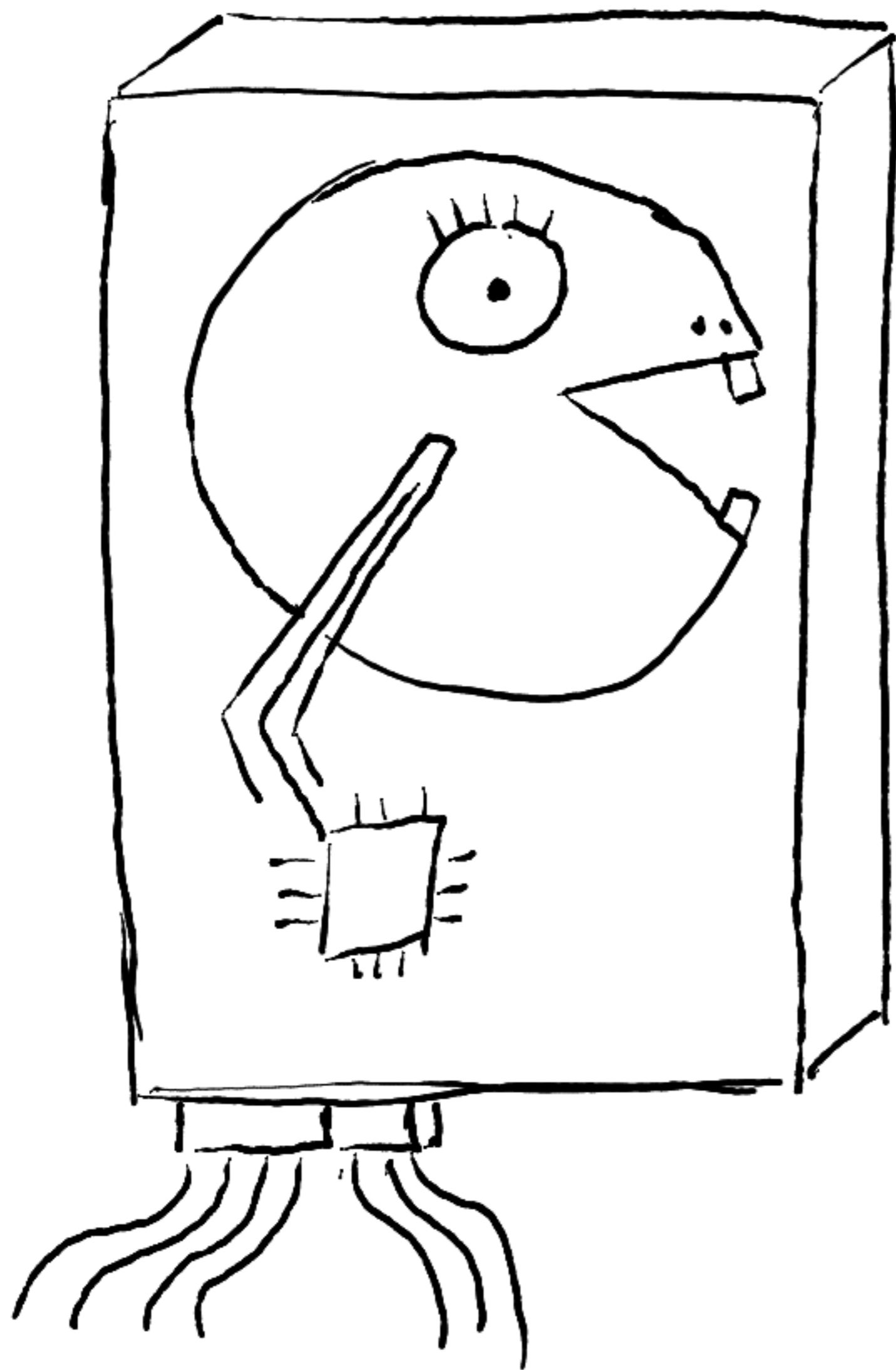
Classic, low-level programming languages:

- explicit memory management
- data remains in memory if not removed
- allocation and de-allocation commands
- e.g. Assembly, C, C++, BASIC

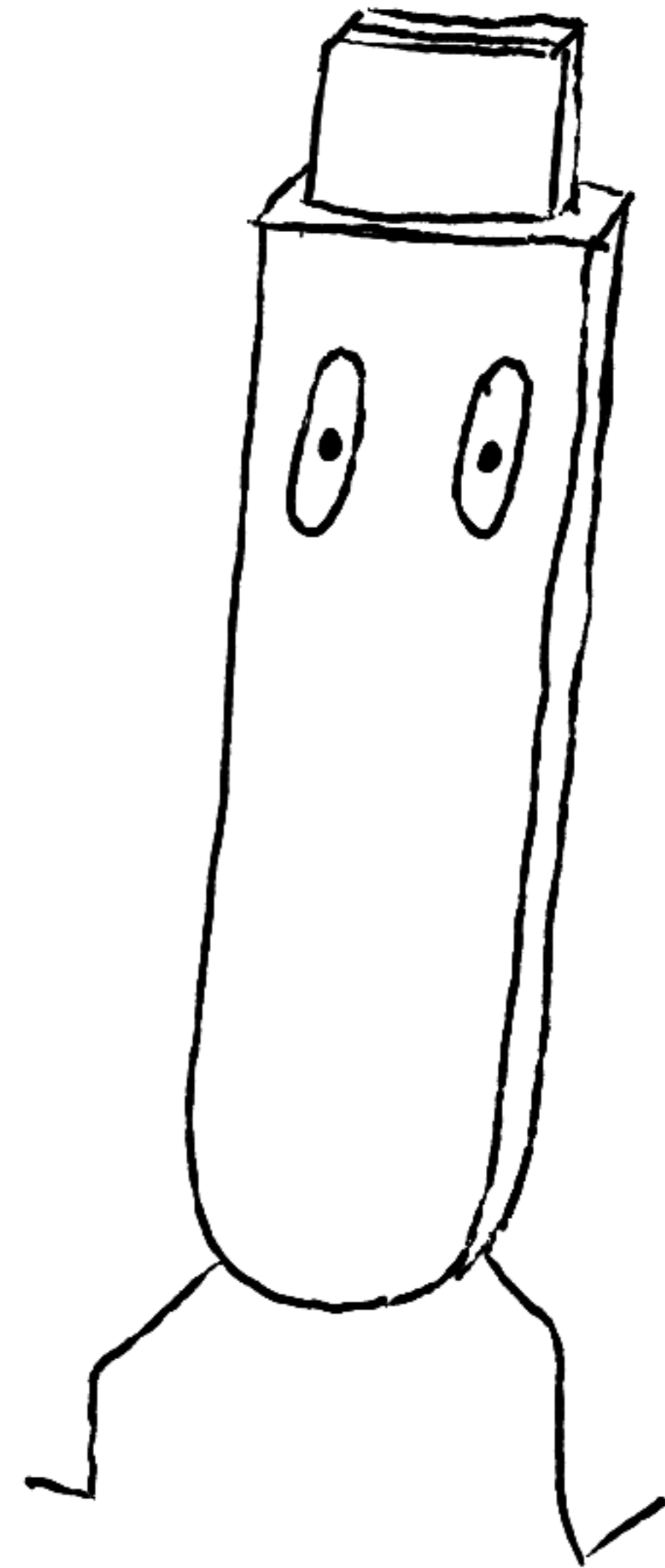
Agenda

- Fundamentals for data languages
- Deallocation Languages
- Deallocation automata
- Kleene theorem and determinisation

Fundamentals for Data Languages



Beep
Boop



What are we talking about?

- Data languages: languages over infinite alphabets

- Can be used for modelling description languages, calculi, etc.

- Usually involve binding mechanisms:

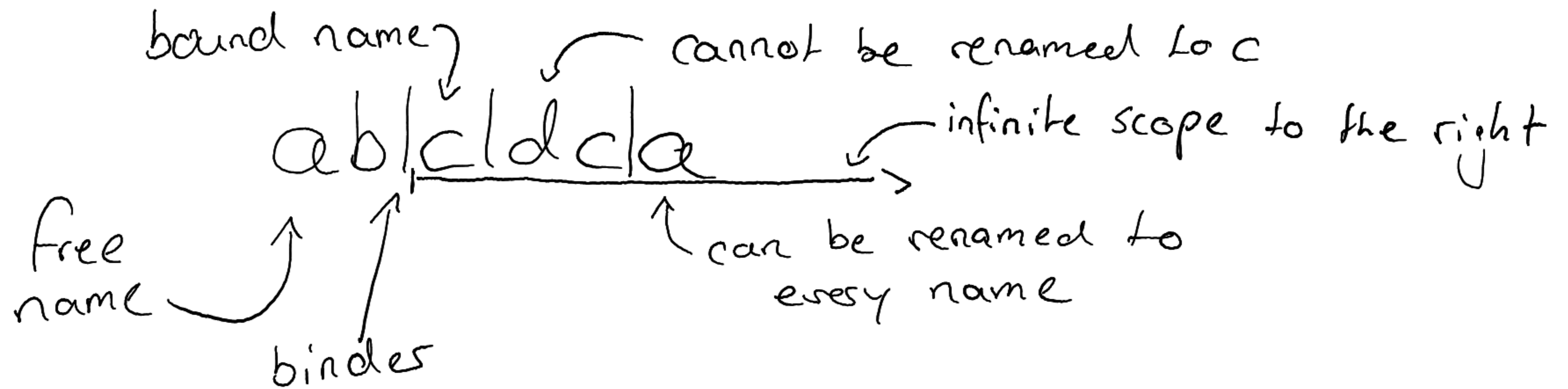
• λ -calculus: $\lambda a b. f b a$

• π -calculus: $\nu x. \bar{x}(a). 0 \mid x(b). 0$

• C-pointers: `int *val = malloc(sizeof(int));`

How can we represent data languages?

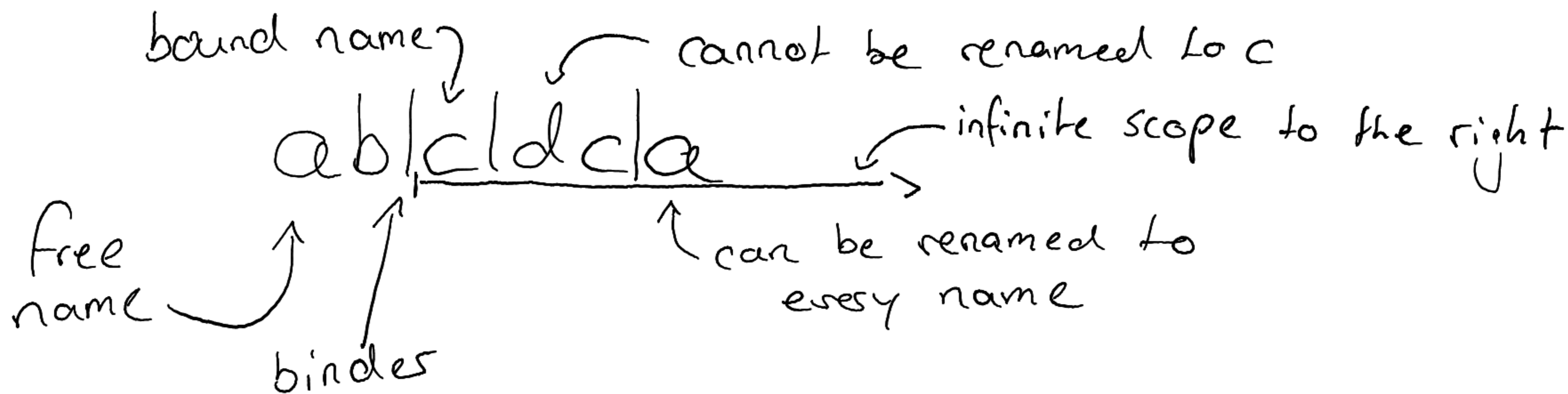
- Bas words[†]:



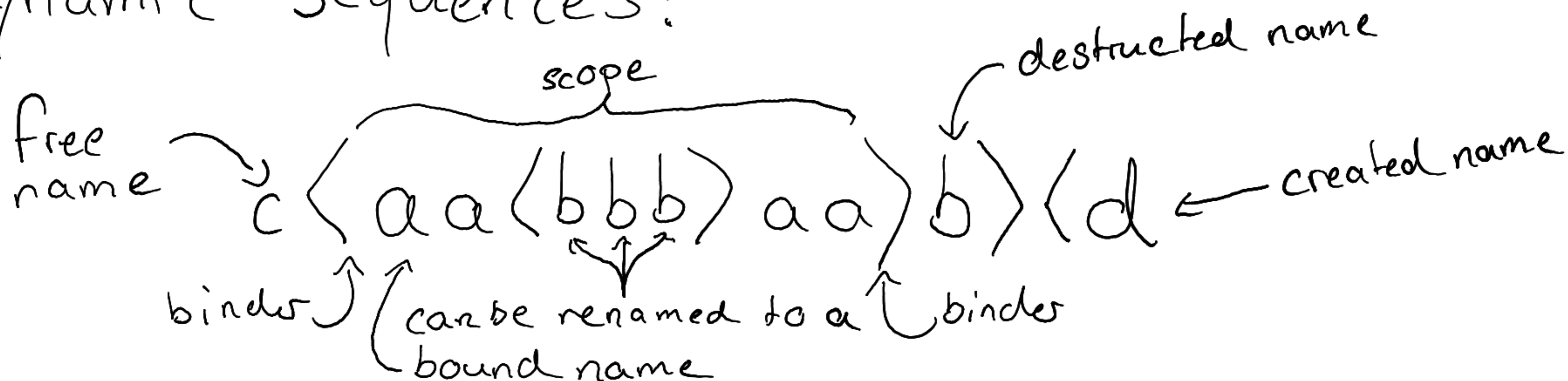
[†] Schröder, Kozel, Milius, Vißmann (FOSSaCS 2017)

How can we represent data languages?

- Bas words[†]:



- Dynamic sequences^{*}:



[†] Schröder, Kozel, Milius, Vißmann (FOSSaCS 2017)

^{*} Gabbay, Ghica, Petrişan (CSL 2015)

Can we have some examples?

$$a \not\equiv_4 b$$

$$|a| \equiv_4 |b|$$

$$a|a| \equiv_4 a|b|$$

$$|a|b|a|b| \not\equiv_4 |a|a|a|a|$$

$$|a|b|b| \equiv_4 |a|a|a|a|$$

$$|a|b|c|b|a|a|c|a|a| \equiv_4 |f|b|c|g|f|c|f|c|$$

Can we have some examples?

$$a \not\equiv_4 b \quad |a \equiv_4 |b \quad a|a \equiv_4 a|b$$

$$|a|b|a|b \not\equiv_4 |a|a|a|a \quad |a|b|b \equiv_4 |a|a|a|a$$

$$|a|b|c|b|a|a|c|a|a \equiv_4 |f|b|c|g|f|f|c|f|c$$

$$\langle a \equiv_4 \langle a$$

$$\langle b \not\equiv_4 \langle a$$

$$a) \equiv_4 a)$$

$$b) \not\equiv_4 a)$$

$$\langle a|a|a \rangle \equiv_4 \langle b|b|b \rangle$$

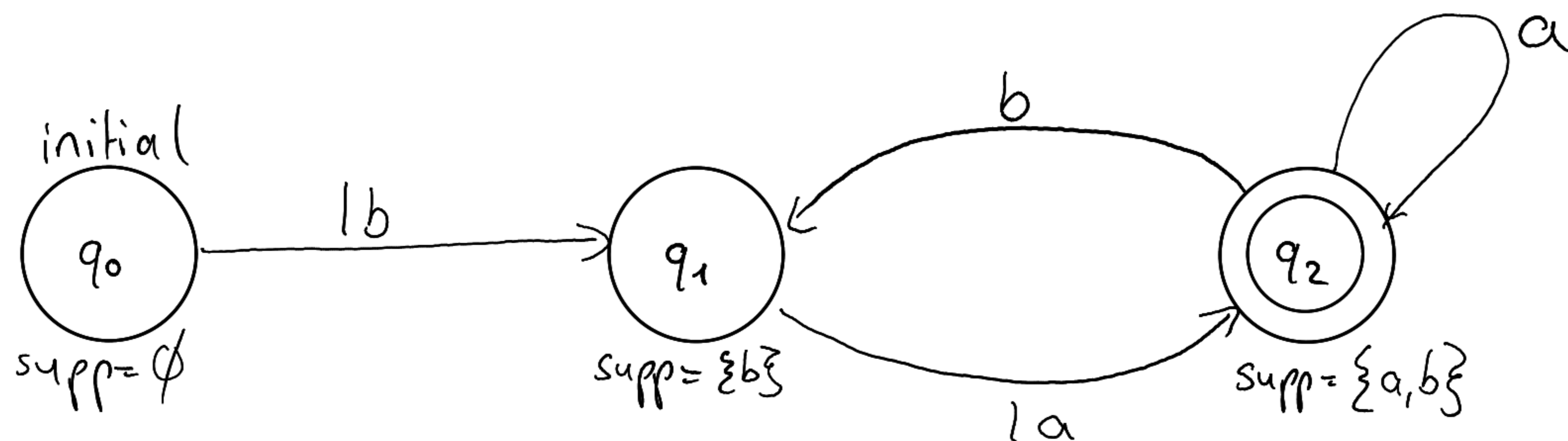
$$\langle a(\langle a|a|a \rangle)a \rangle \equiv_4 \langle b(\langle c|c|c \rangle)b \rangle$$

$$\langle a(\langle b|a \rangle)b \rangle \not\equiv_4 \langle a(\langle a|a \rangle)a \rangle$$

Nice

Which automata exist for these representations?

- $NOFA^{\textcircled{1}}$: equivalent to bar languages⁺ and a flavour of dynamic sequences^{*}
- bar NFA^+ : literally just NFA for letters with bars (oL)
- $RNNA^+$ (Regular Nondeterministic Nominal Automaton):



↳ recognises the language $(|b|a a^* [b|a a^*])$

^① Bojańczyk, Klin, Lasota (LMCS 2014)

⁺ Schröder, Kozel, Milus, Vißmann (FOSSACS 2017) ^{*} Brunet, Silva (ICALP 2019)

De-allocation Languages



Trigger Warning

The next slide
contains a
functor!

Explicit Content - Parental Advisory

Raw De-allocation Sequences

- Elements of the initial algebra μF for

$$FX := X \times A + X * A + \llbracket A \rrbracket X$$

free names a allocated names $\langle a$ de-allocated names $a \rangle$

- Syntax like dynamic sequences
- Changed semantics:

Raw De-allocation Sequences

- Elements of the initial algebra μF for

$$FX := X \times A + X * A + \llbracket A \rrbracket X$$

free names a allocated names $\langle a$ de-allocated names $a \rangle$

- Syntax like dynamic sequences
- Semantics:

Raw De-allocation Sequences

- Elements of the initial algebra μF for

$$FX := X \times A + X * A + \llbracket A \rrbracket X$$

free names a allocated names $\langle a \text{ de-allocated names } a \rangle$

- Syntax like dynamic sequences

- Semantics:

- Words are constructed left to right
- Opening brackets are clear
- Closing brackets are abstract
- Still words are read left to right

Examples:

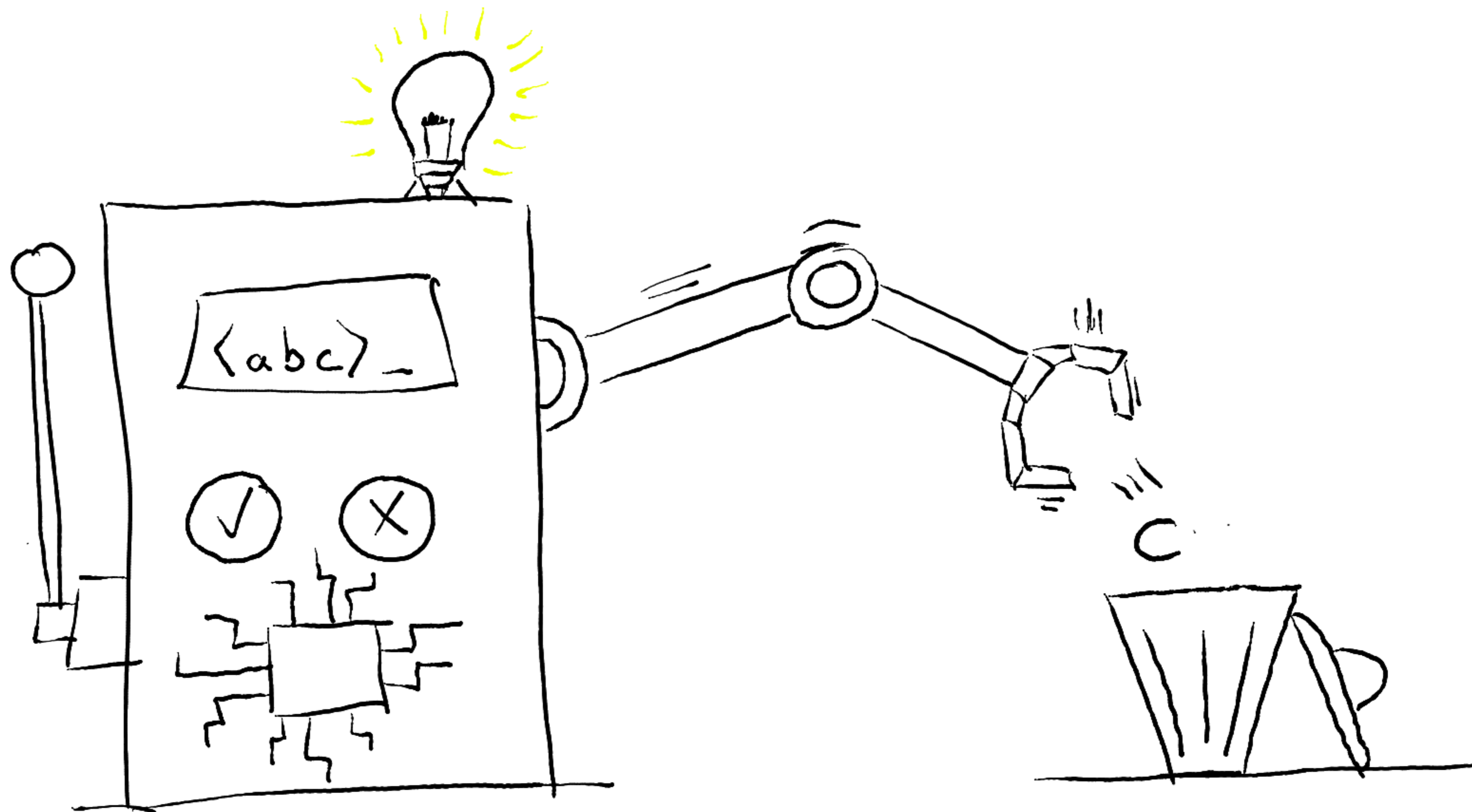
Why the hell
would we want
that?



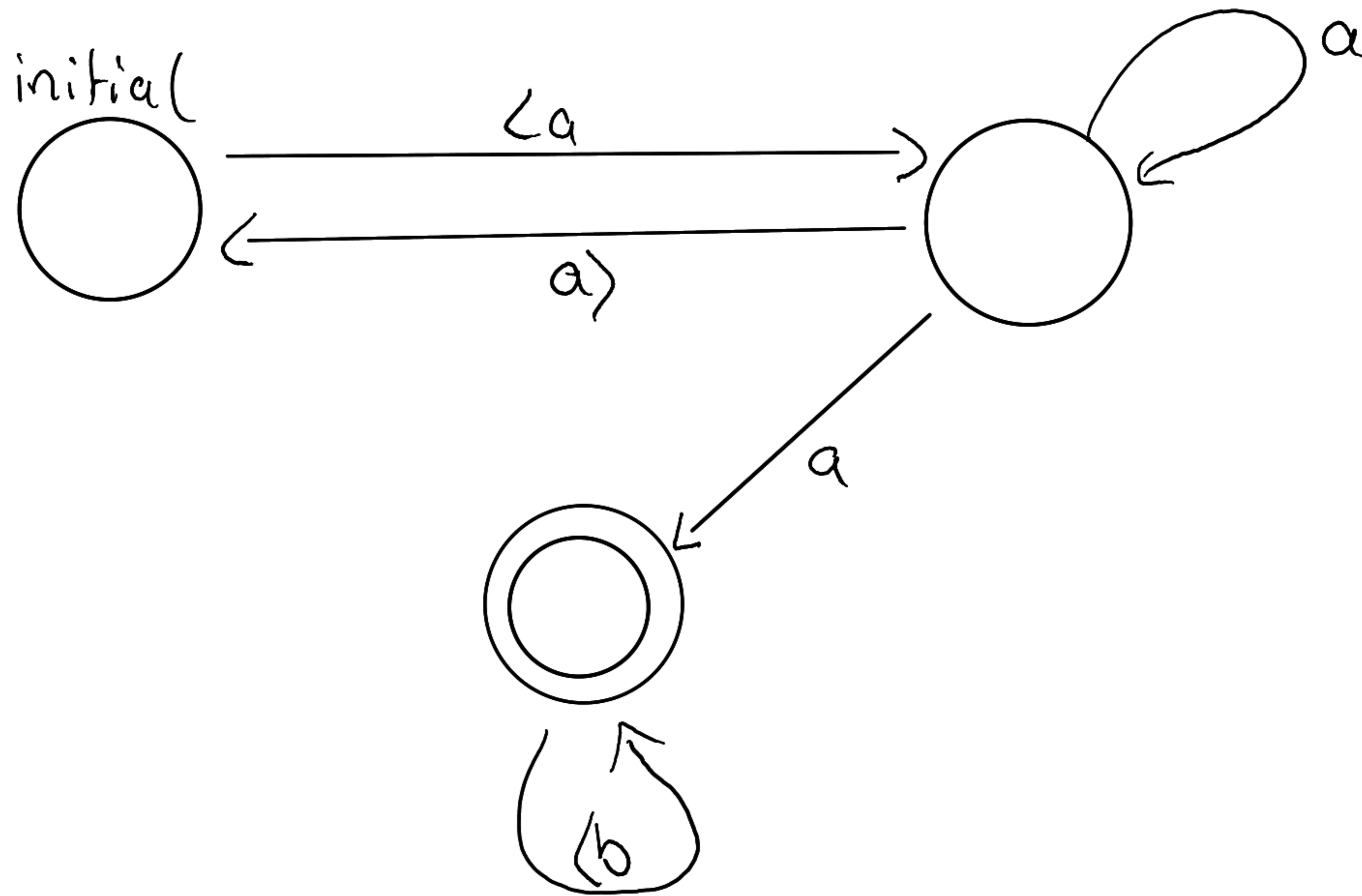
Low-level, imperative Intuition:

- Programmes are executed in the same order they are written in (top-down).
- De-allocating memory that we have never allocated is completely fine in C.
- Malloc will not return the same address twice, unless the address was freed.
- The identity of a freed address becomes irrelevant.

De-allocation Automata

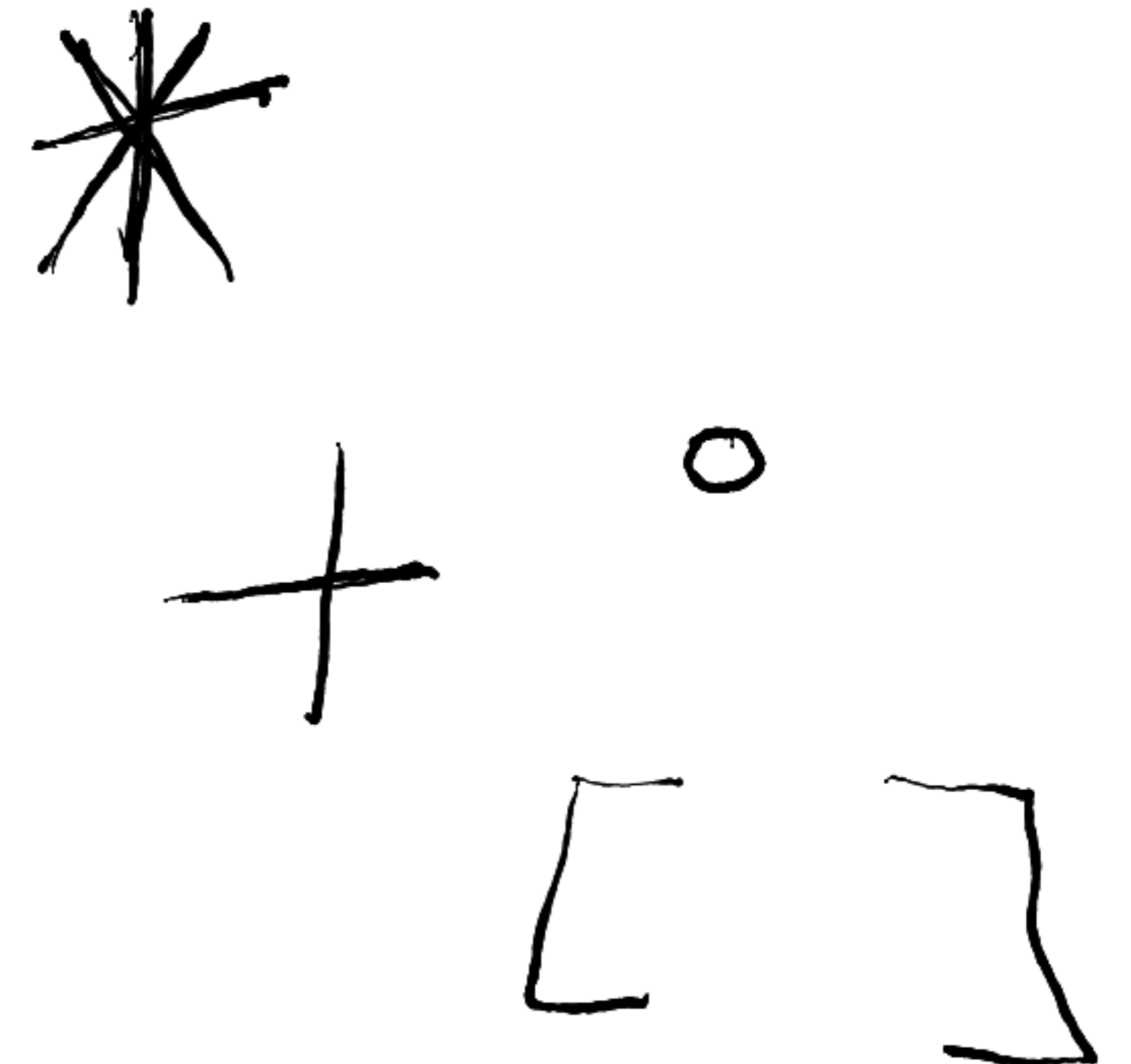
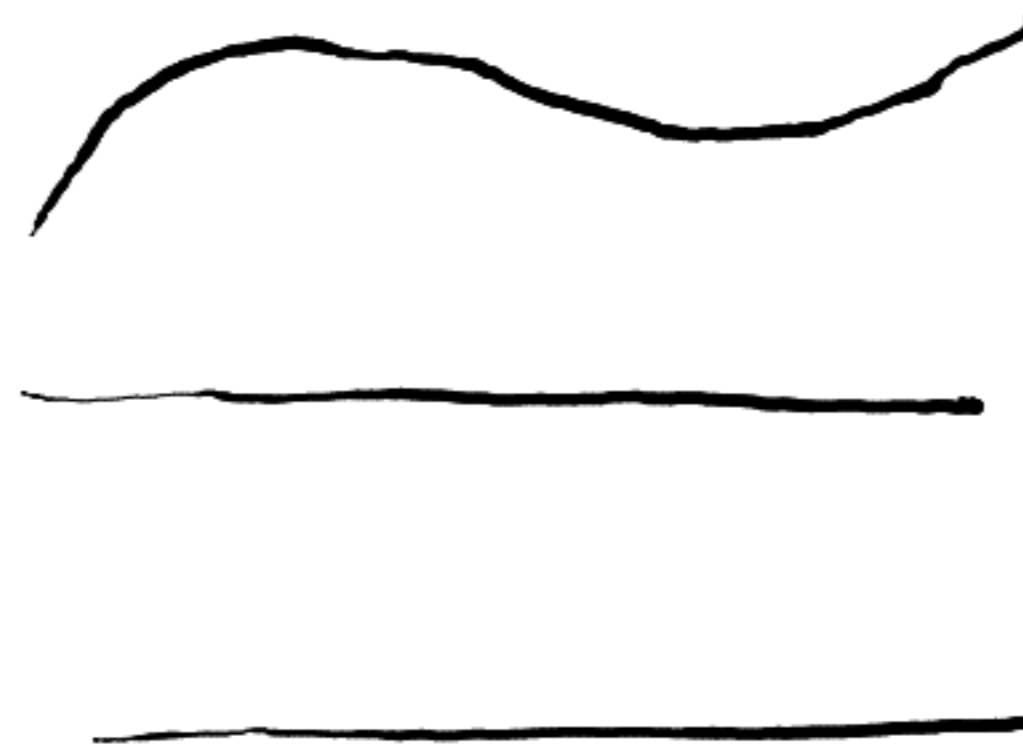
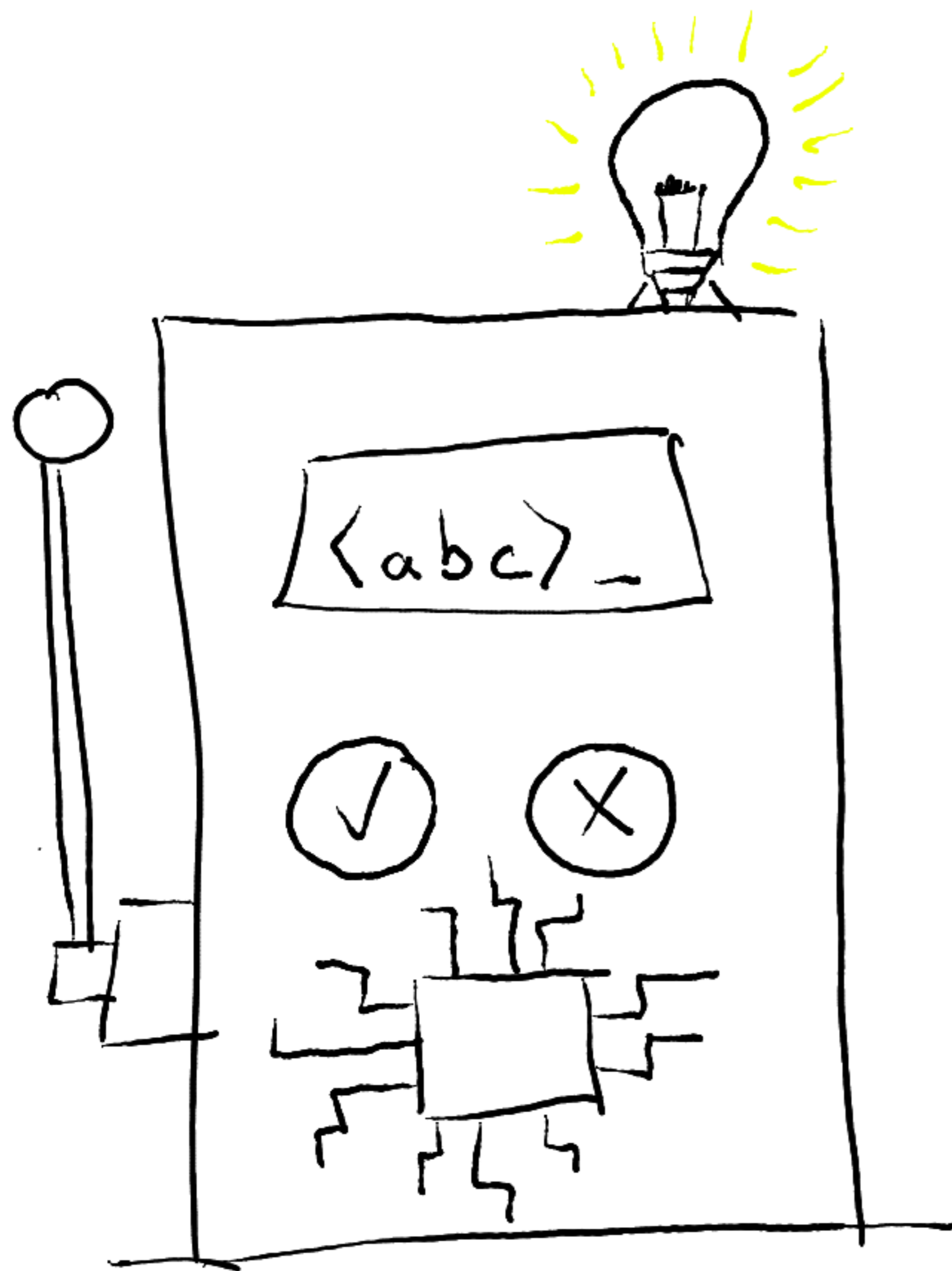


Non-deterministic De-allocation Automaton (NDA)

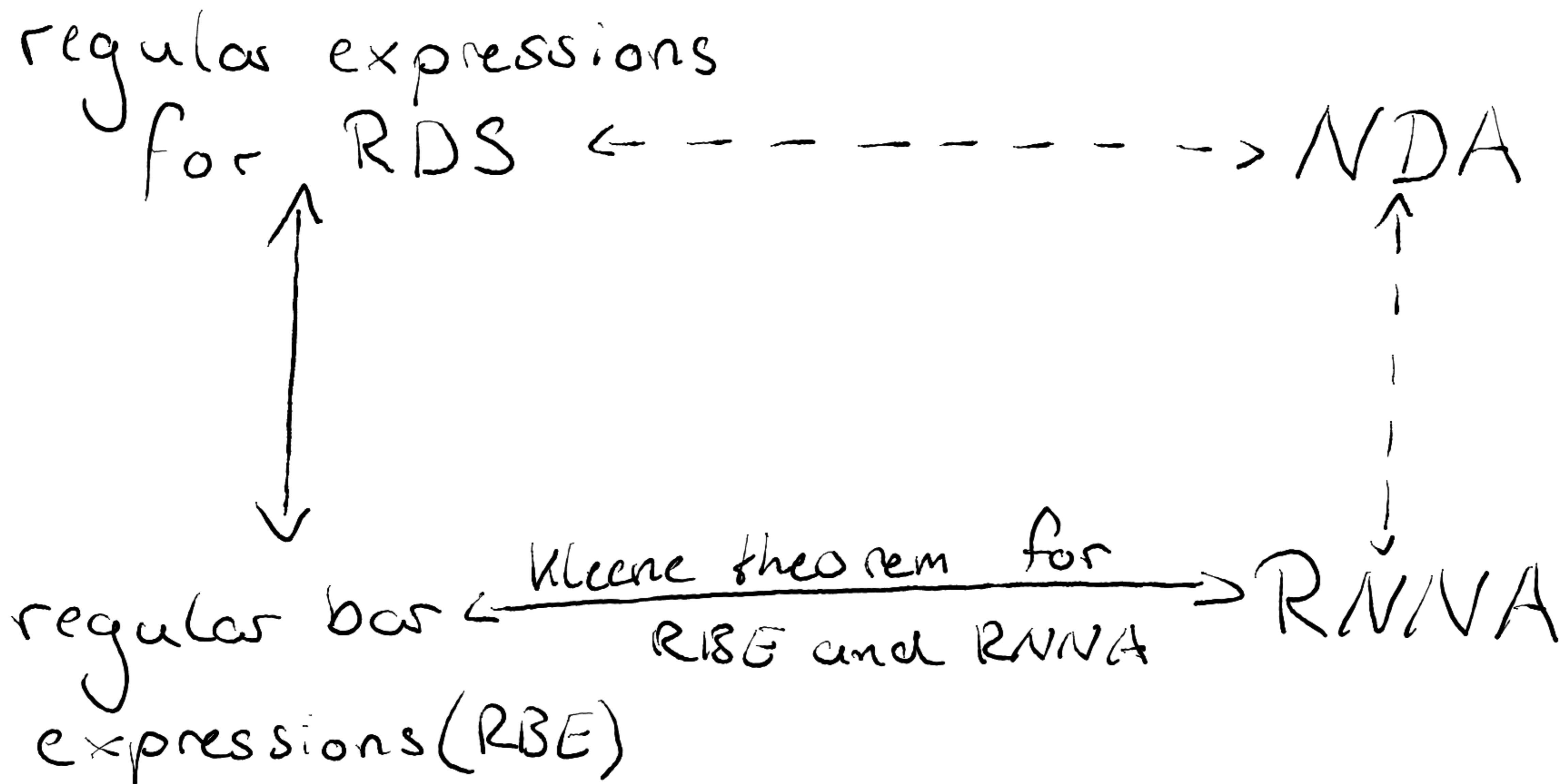


↳ accepts the language $L(\langle aa^*(a)(aa^*)^*a \langle b^* \rangle)$

Kleene theorem and determinisation



Kleene theorem for RDS and NDA



Mutual translation of RDS and bar strings

$$d2b: \hat{A}^* \rightarrow \bar{A}^*$$

$$d2b(\epsilon) = \epsilon, \quad d2b(s a) = |a| d2b(s),$$

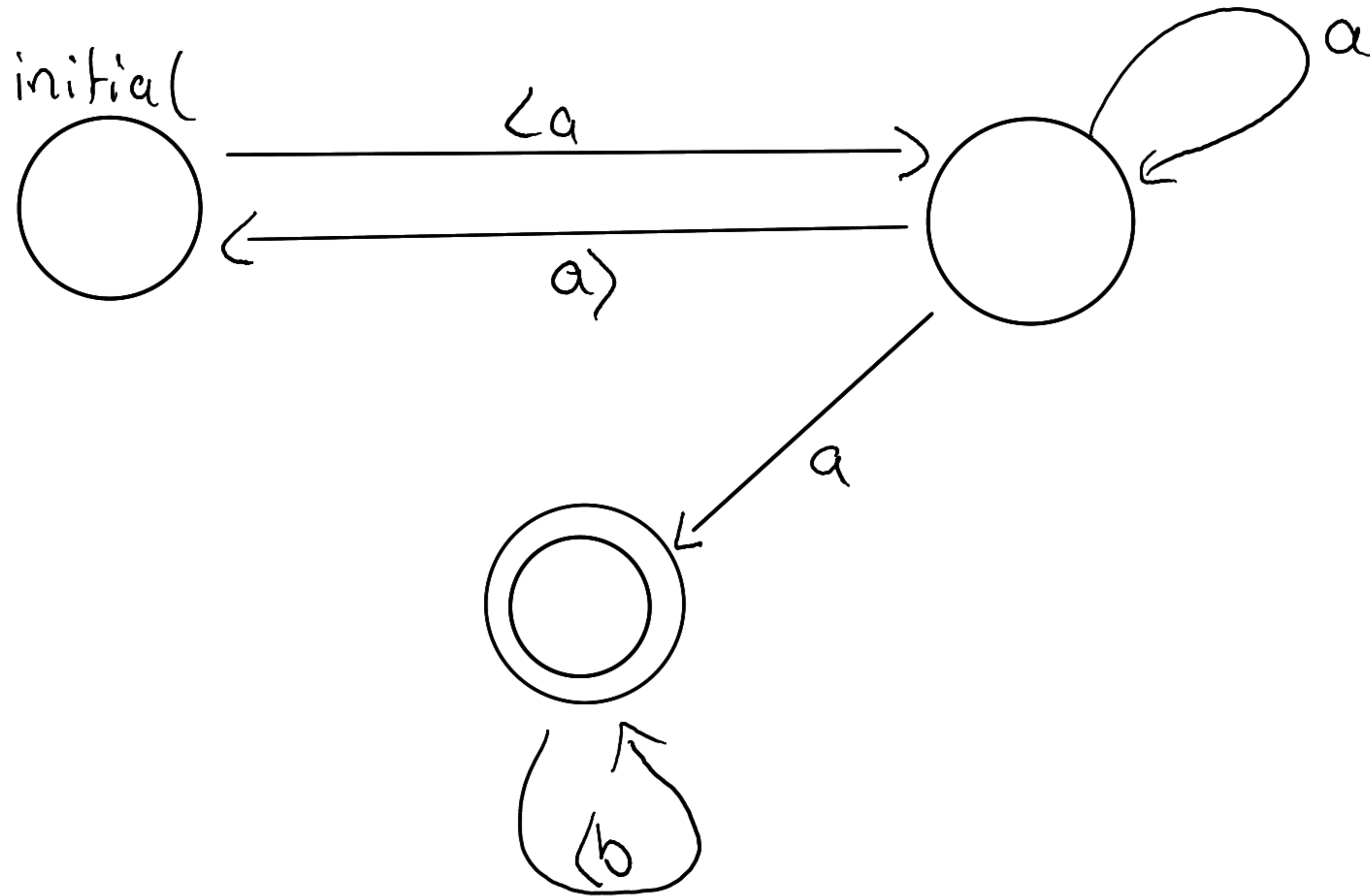
$$d2b(s a) = a d2b(s), \quad d2b(s \langle a) = a d2b(s)$$

$$b2d(\epsilon) = \epsilon, \quad b2d(|a s) = b2d(s) a),$$

$$b2d(a s) = \begin{cases} b2d(s) \langle a & \text{if } a \notin FN(s) \\ b2d(s) a & \text{otherwise} \end{cases}$$

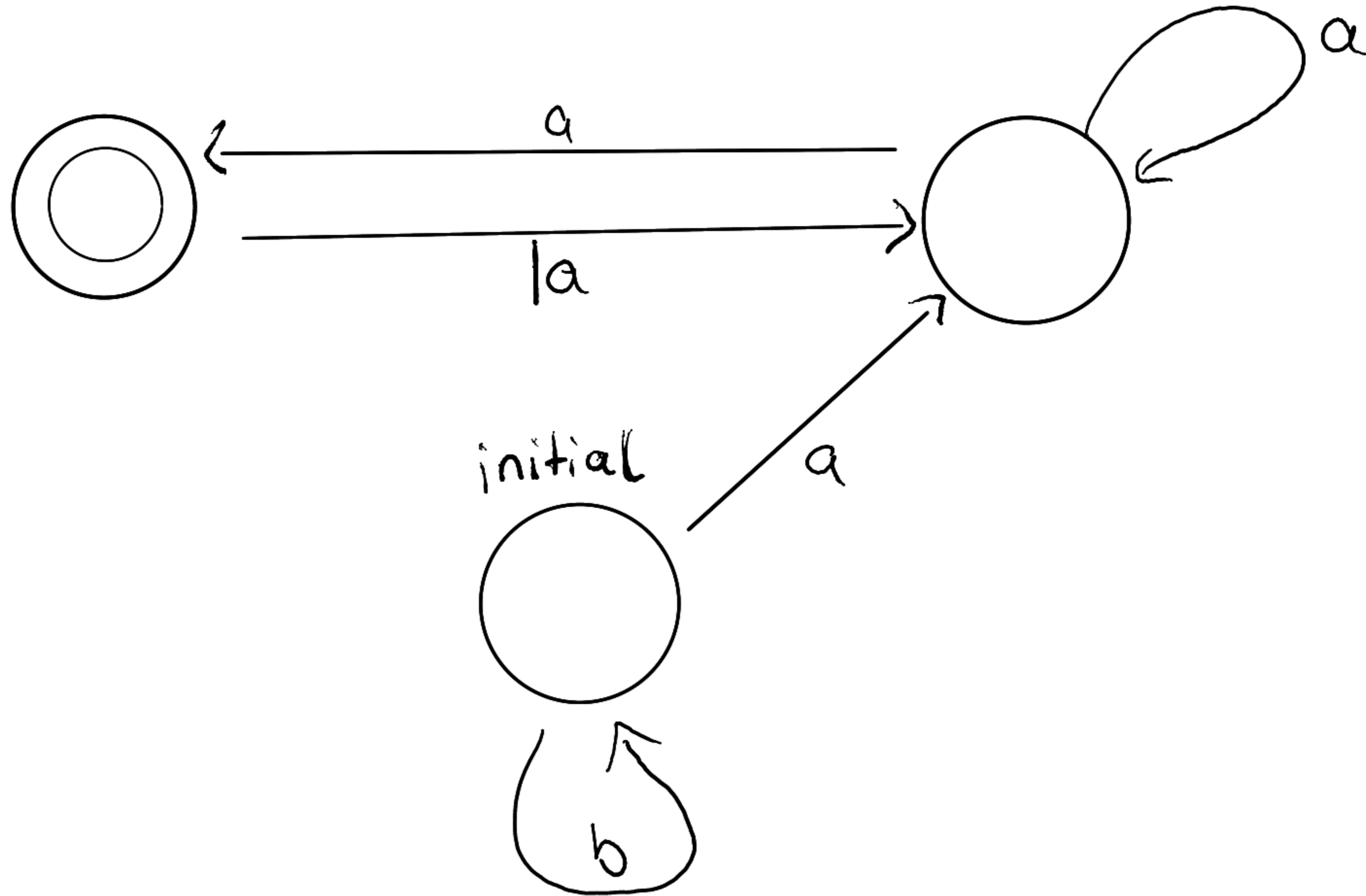
$$\langle a \langle b b b) a a) \hat{=} |a a| b b b a$$

NDA 2 RNNVA



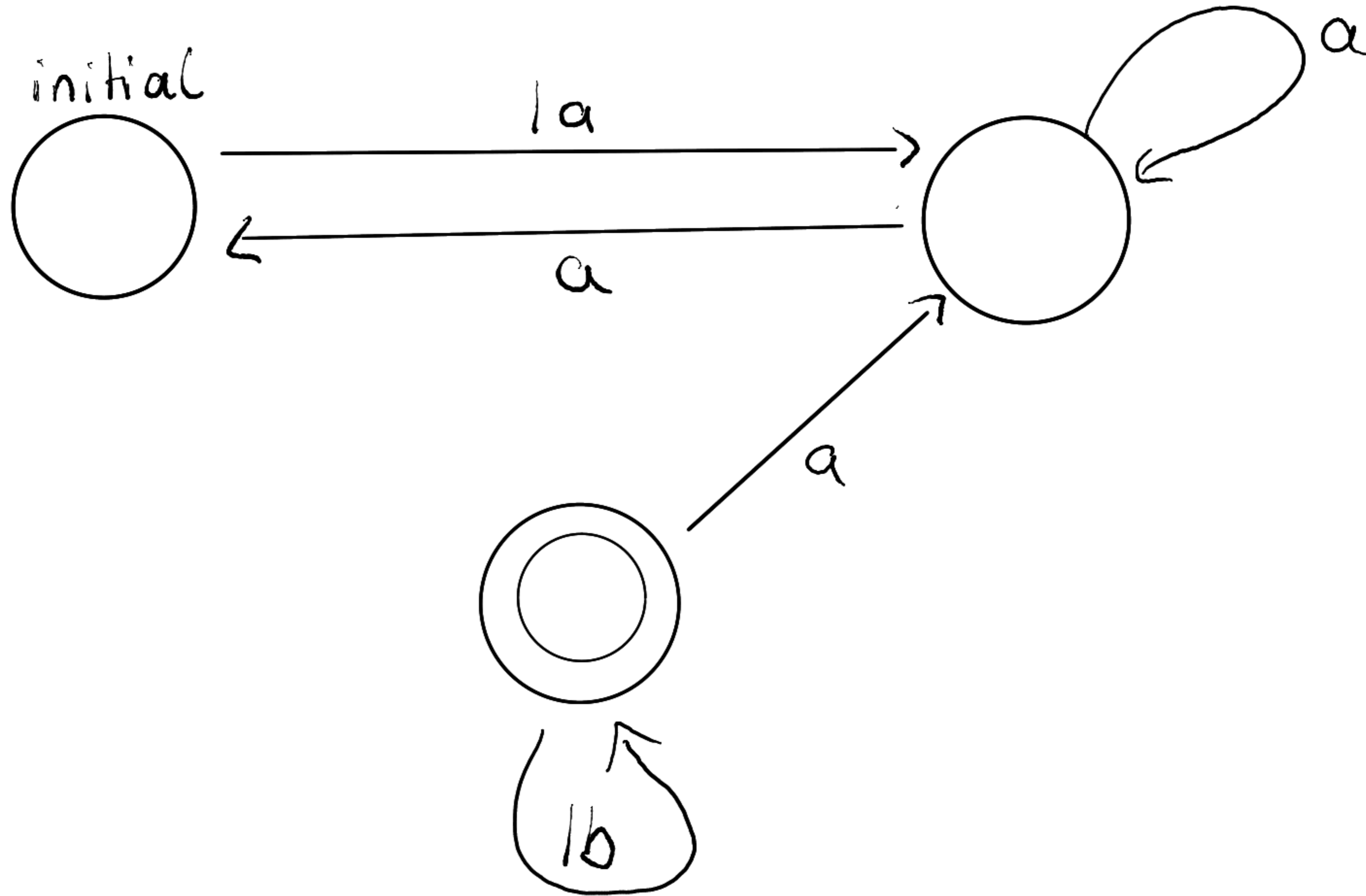
NDA 2 RNNVA

Ideal:

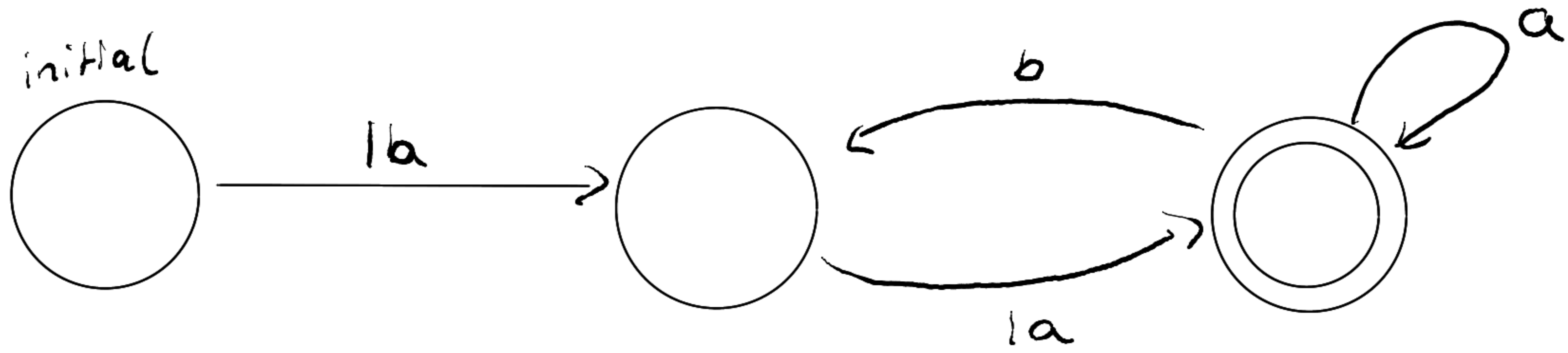


NDA 2 RNNA

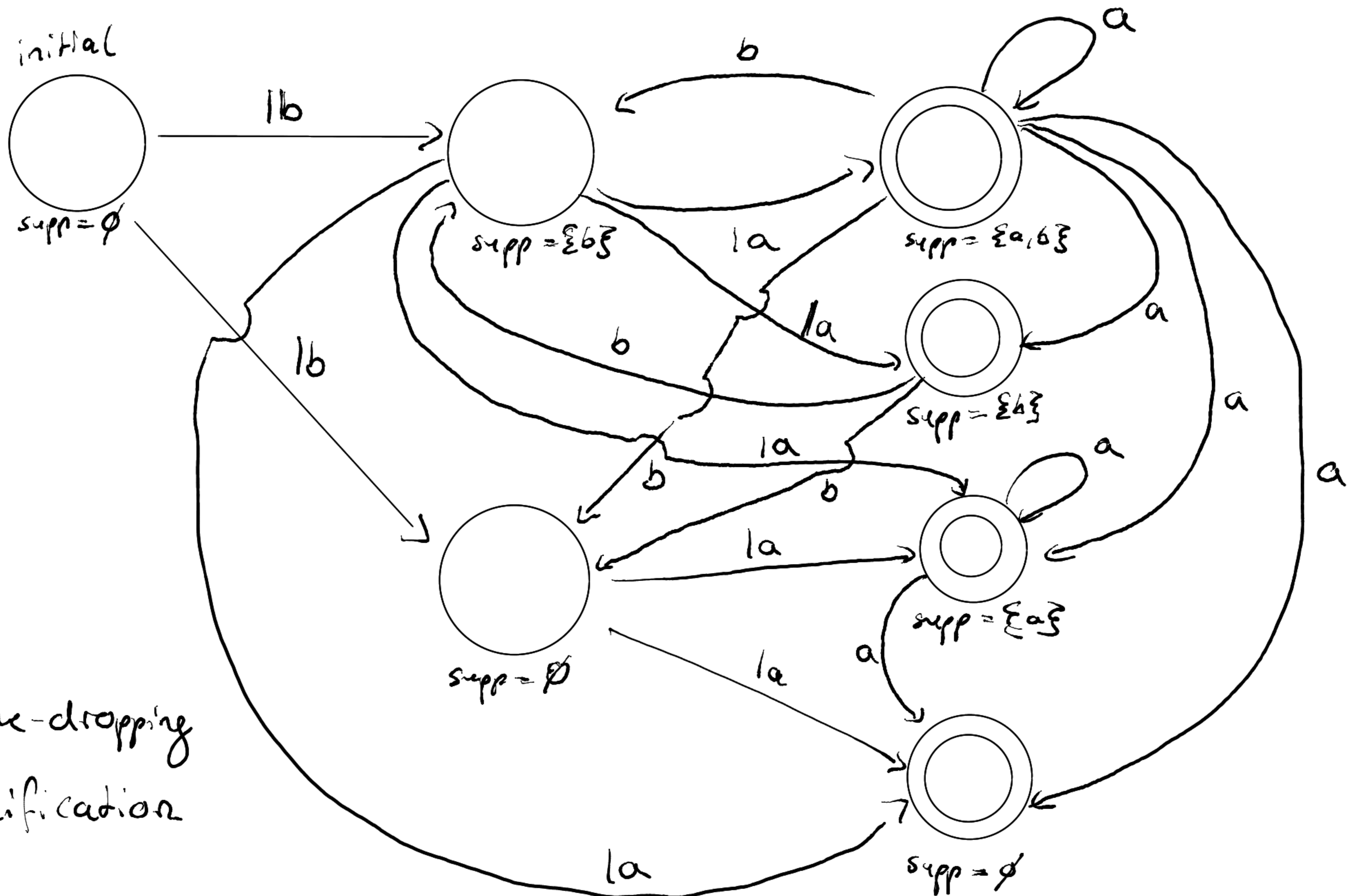
Idea 2:



RNNA 2 NDA

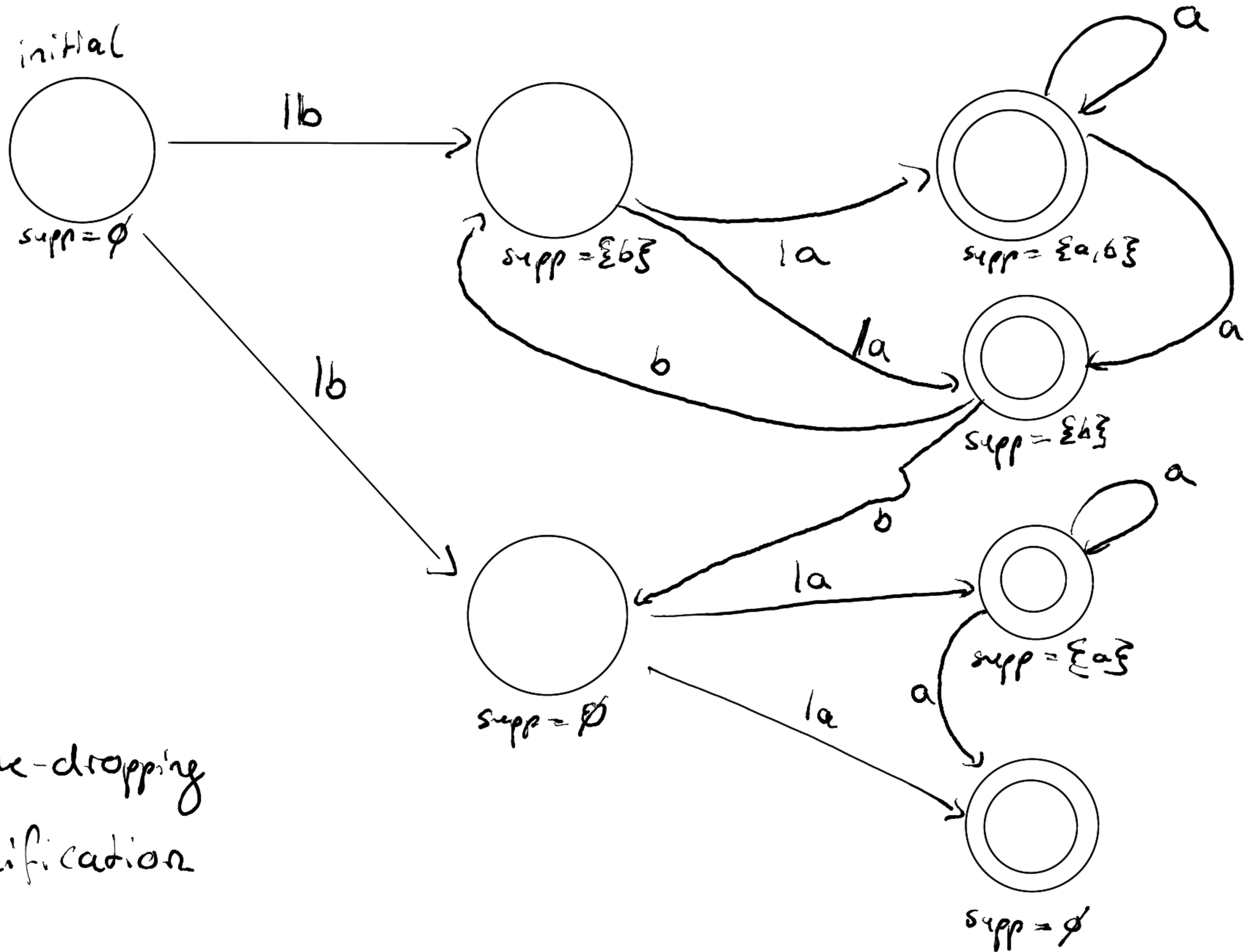


RNNA 2 NDA



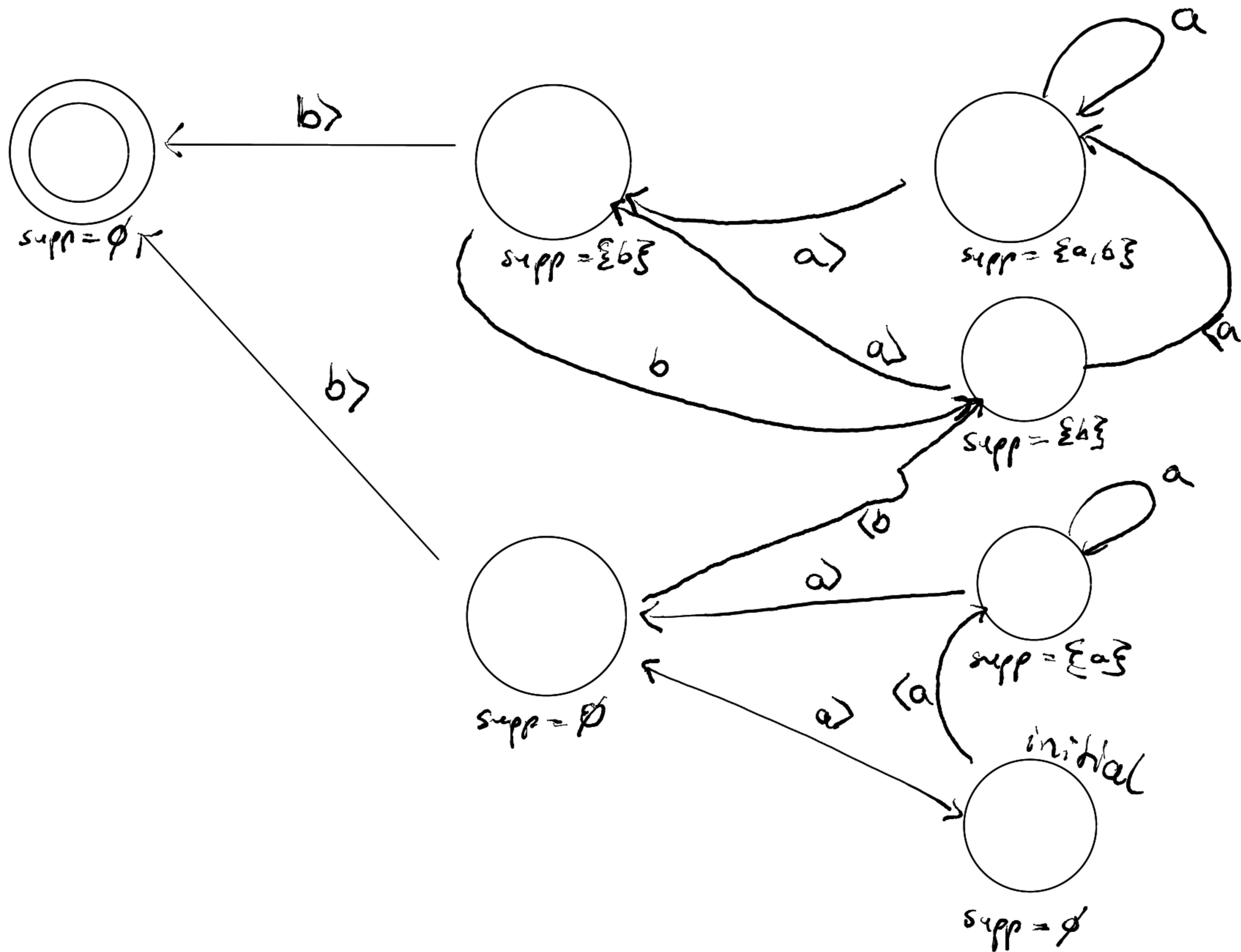
Name-dropping
modification

RNNA 2 NDA



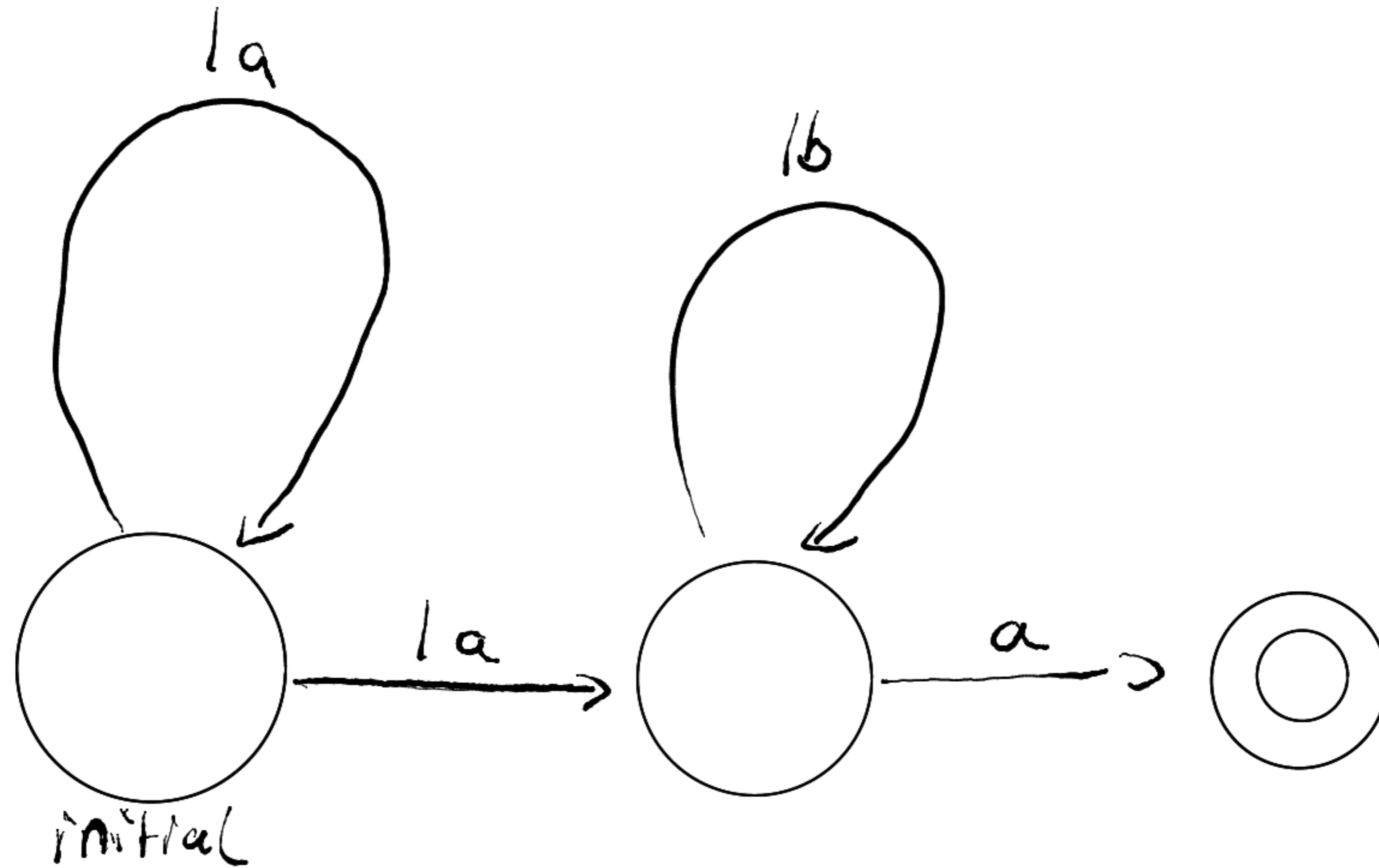
Name-dropping
modification

RNNA 2 NDA



Determinisation

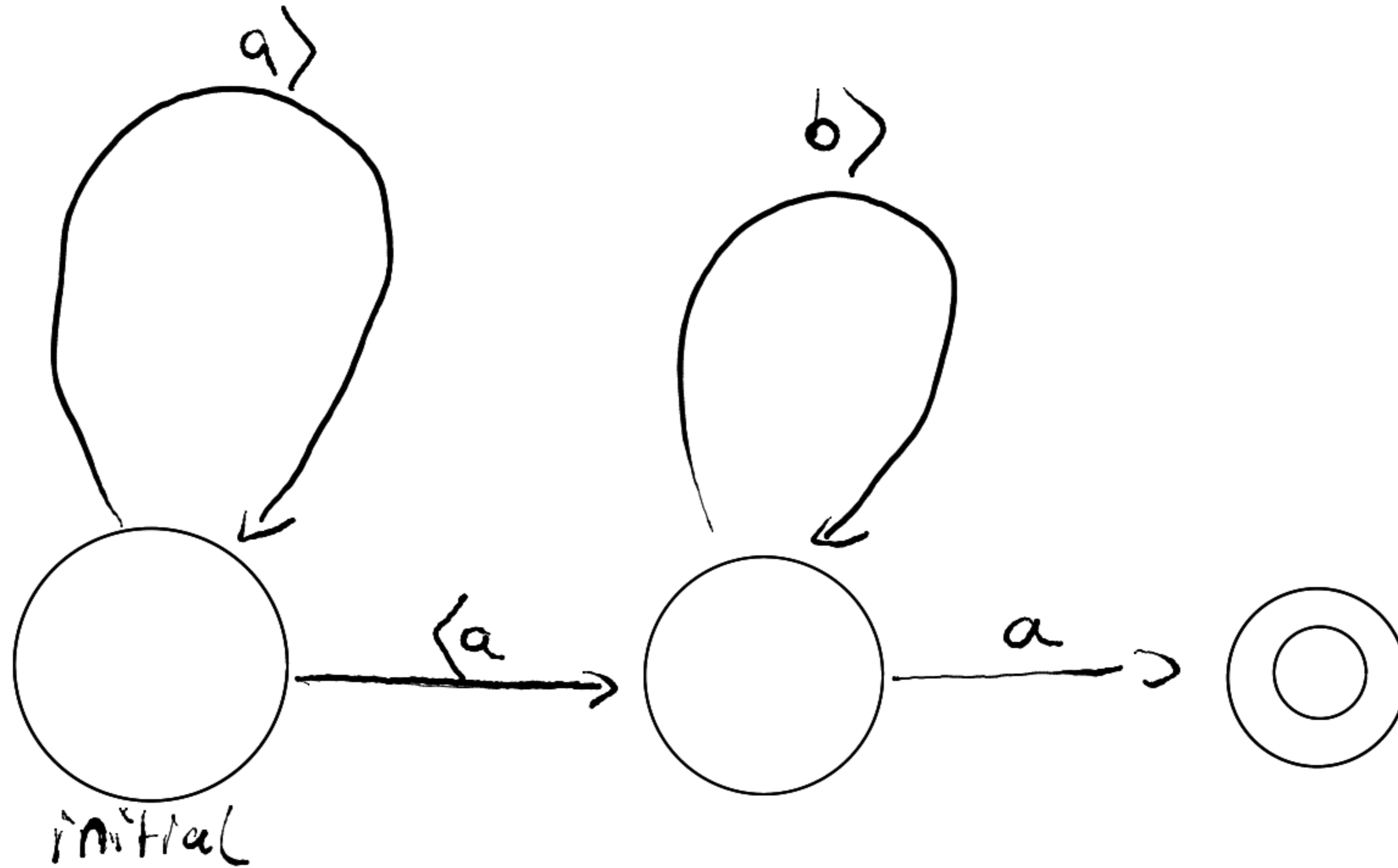
RNNA:



↳ not determinisable

Determinisation

NDA:



↳ deterministic NDA accepting
the same data language

Roadmap to determinisation

