Weighted Automata Extraction from Recurrent Neural Networks via Regression on State Spaces

Takamasa Okudono, Masaki Waga, Taro Sekiyama, Ichiro Hasuo

SOKENDAI, the Graduate University for Advanced Studies, Japan

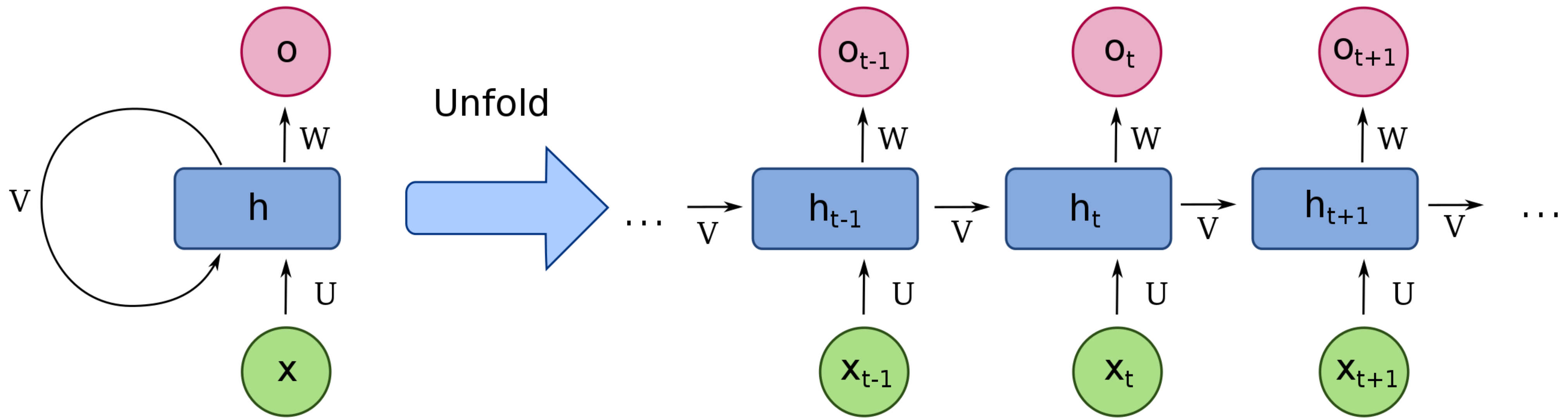
/National Institute of Informatics, Japan

LearnAut19, Vancouver, Canada

23 June 2019

RNN

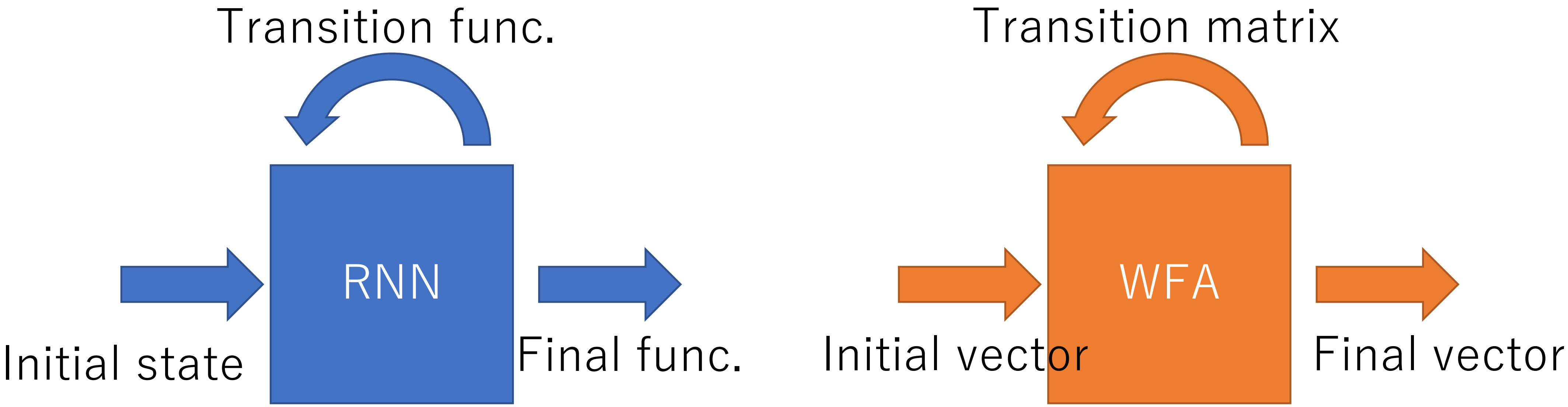
RNN is a neural network equipped with a internal state



Goal

Input: RNN R whose output is in $\mathbb{R}$ (defines $f_R : \Sigma^* \to \mathbb{R}$)

Output: WFA $A(R)$ (defines $f_{A(R)} : \Sigma^* \to \mathbb{R}$) s.t. $f_{A(R)} \simeq f_R$



Motivation

- Getting **lighter** (faster to infer) model of an RNN
  - Because the inference of RNNs are sometimes heavy
- Investigate the behavior of RNN R via the extracted WFA $A(R)$
  - WFA equips many operations and leads to model checking?
- In research line of RNN⇔DFA conversion as an acceptor
  - Ours is a quantitative extension

Contribution

- Proposed a method to apply Balle and Mohri’s algorithm for the extraction
  - The key is checking if $R \simeq A$ by using regression
- Our method extracts +7% more accurate models than the baseline
- The extracted WFAs are about 1,000 times faster to infer than the target RNNs

Def. of RNN (Mathematically, in this work)

RNN R (of alphabet Σ and dimension d) consists of

- $\alpha \in \mathbb{R}^d$: Initial state
- $\beta: \mathbb{R}^d \to \mathbb{R}$: Final function
- $g_R: \mathbb{R}^d \times \Sigma \to \mathbb{R}^d$: Transition function
  - $g_R: \mathbb{R}^d \times \Sigma^* \to \mathbb{R}^d$ is induced recursively■

Need not to be linear

$f_R: \Sigma^* \to \mathbb{R}$ is *induced by* $f_R(w_1 \dots w_N) = \beta \circ g_R(\alpha, w_1 \dots w_N)$

The *configuration* for $w_1 \dots w_N$ is defined by $\delta_R(w_1 \dots w_N) = g_R(\alpha, w_1 \dots w_N)$

“internal state”

Def. of Weighted Finite Automaton (WFA)

WFA A (of size n and alphabet Σ) consists of

- $\alpha \in \mathbb{R}^n$: Initial vector
- $\beta \in \mathbb{R}^n$: Final vector
- $A_\sigma \in \mathbb{R}^{n \times n}$: Transition matrix ($\sigma \in \Sigma$) ■

WFA A is a formalism to define $f_A\colon \Sigma^* \to \mathbb{R}$

(c.f.) A DFA is a formalism to define $f\colon \Sigma^* \to 2$

WFA is an extension of DFA via the matrix representation.

Def. of WFA

- WFA A induces the function $f_A: \Sigma^* \to \mathbb{R}$ as
$$f_A(w_1 \dots w_N) = \alpha A_{w_1} \dots A_{w_N} \beta$$
- The *configuration* ("internal state") of WFA A *is*
$$\delta_A(w_1 \dots w_N) = \alpha A_{w_1} \dots A_{w_N} \in \mathbb{R}^n$$

<u>For example:</u>

- $\Sigma = \{0, 1\}, \alpha = \begin{pmatrix} 0.8 & 0.2 \end{pmatrix}, \beta = \begin{pmatrix} 0.9 \\ 0.7 \end{pmatrix}, A_0 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, A_1 = \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix}$
- $f_A(10) = \begin{pmatrix} 0.8 & 0.2 \end{pmatrix} \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0.9 \\ 0.7 \end{pmatrix} = 0.736$
- $\delta_A(10) = \begin{pmatrix} 0.8 & 0.2 \end{pmatrix} \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0.18 & 0.82 \end{pmatrix}$

RNN and WFA

RNN R (of alphabet Σ and dimension d) consists of

- $\alpha \in \mathbb{R}^d$: Initial state
- $\beta: \mathbb{R}^d \to \mathbb{R}$: Final function
- $g_R: \mathbb{R}^d \times \Sigma \to \mathbb{R}^d$: Transition function ■

WFA A (of alphabet Σ and size n) consists of

- $\alpha \in \mathbb{R}^n$: Initial vector
- $\beta \in \mathbb{R}^n$: Final vector
- $A_\sigma \in \mathbb{R}^{n \times n}$: Transition matrix ($\sigma \in \Sigma$) ■

Similar formalism!
Can we approximate
RNN by WFA?

Goal and Our Approach

Goal

Input: RNN R whose output is in $\mathbb{R}$ (defines $f_R\colon \Sigma^* \to \mathbb{R}$)

Output: WFA $A(R)$ (defines $f_{A(R)}\colon \Sigma^* \to \mathbb{R}$) s.t. $f_{A(R)} \simeq f_R$

Approach: Use Balle and Mohri's algorithm

- The challenge is to give the procedure to check if $f_A \simeq f_R$ for a candidate WFA A

Balle and Mohri's Algorithm

An extension of Angluin's L* Algorithm <u>for WFA</u>

- Input:
  - Membership query procedure $\mathbf{m}\colon \Sigma^* \to \mathbb{R}$
  - Equivalence query procedure $\mathbf{e}\colon \{\mathbf{WFAs}\} \to \{\mathbf{Equivalent}\} \sqcup \Sigma^*$
- Output:
  - Minimal WFA A'
- Property: Given WFA A, if $m = f_A$ and

$$e(\tilde{A}) = \begin{cases} \text{Equivalent}\,; f_A = f_{\tilde{A}} \\ w\,; f_A(w) \neq f_{\tilde{A}}(w) \end{cases}$$

Called "Counterexample"

then, it terminates by calling m, e polynomial times and $f_A = f_{A'}$

Idea of Overall Architecture (Detailed)

Implement

- Membership query m to be the RNN's induced function f_R
- Equivalence query e to be

$$e(\tilde{A}) = \begin{cases} \text{Equivalent}\,; f_R \simeq f_{\tilde{A}} \\ w\,; f_R(w) \neq f_{\tilde{A}}(w) \end{cases}$$

Generally it cannot be "="

Then we would be able to get a WFA $\tilde{A}$ s.t. $f_R \simeq f_{\tilde{A}}$!

But how can we implement such an equivalence query e?

How do we know $f_R \simeq f_A$?

$$f_R(w) \simeq f_A(w)$$
$$\Leftrightarrow \beta_R \circ \delta_R(\alpha_R, w_1 \dots w_n) \simeq \delta_A(w_1 \dots w_n)\beta_A$$

Both calculate their configurations
("internal states")

If there is a "good" relation between δ_R and δ_A,
A and R would behave similarly

“Good” relation between $\boldsymbol{\delta_R}$ and $\boldsymbol{\delta_A}$

- This work views $p\colon \mathbb{R}^d \to \mathbb{R}^n$ satisfying the following property as a good relation:

$$\forall w \in \Sigma^*.\ \mathrm{p}\big(\delta_R(\mathrm{w})\big) \simeq \delta_A(w)$$

Equivalence Query by approximating $\boldsymbol{p}$

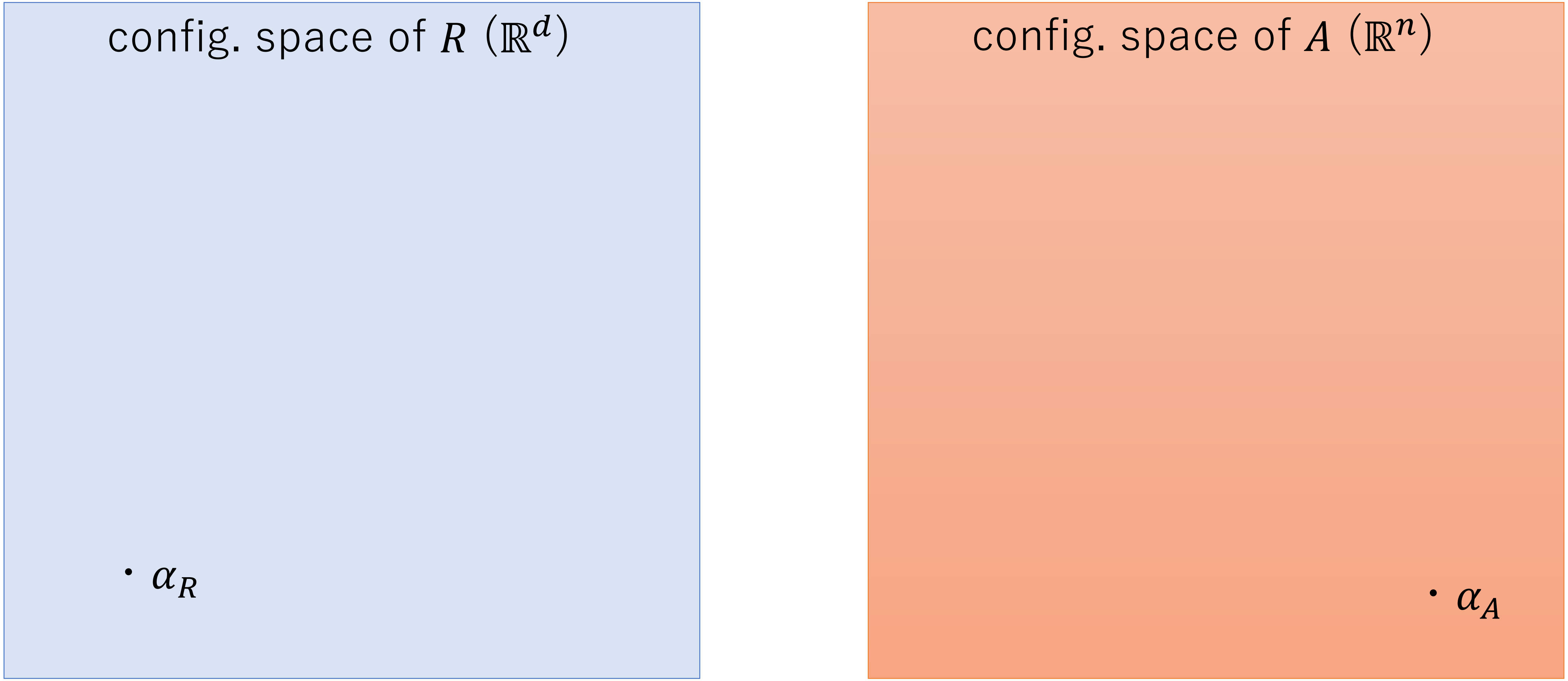
Let's approximate *configuration translator* $p: \mathbb{R}^d \to \mathbb{R}^n$ such that

$$\forall w \in \Sigma^*.\ \mathrm{p}\big(\delta_R(\mathrm{w})\big) \simeq \delta_A(w)$$

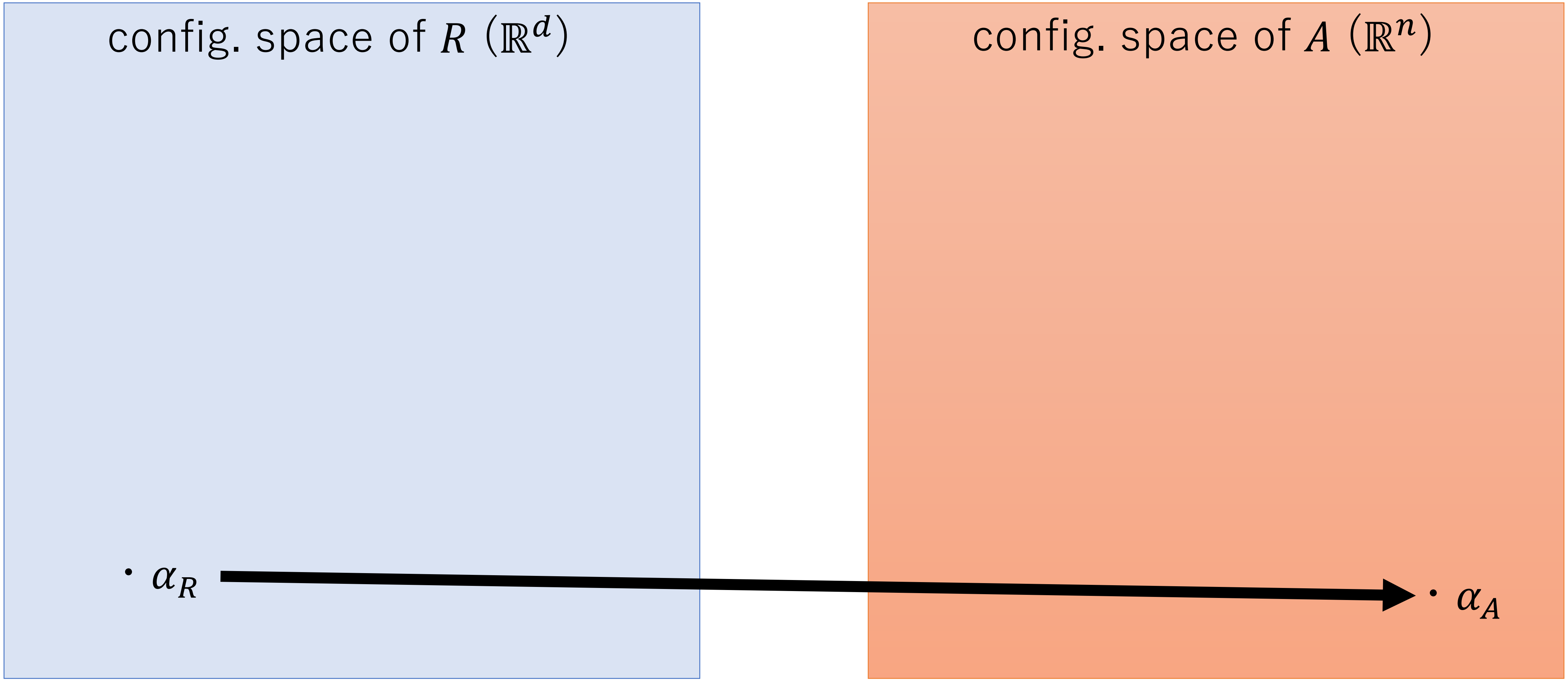
by applying **regression** on sampled data.

The data is sampled by observing Σ^* in Breadth-First Search.

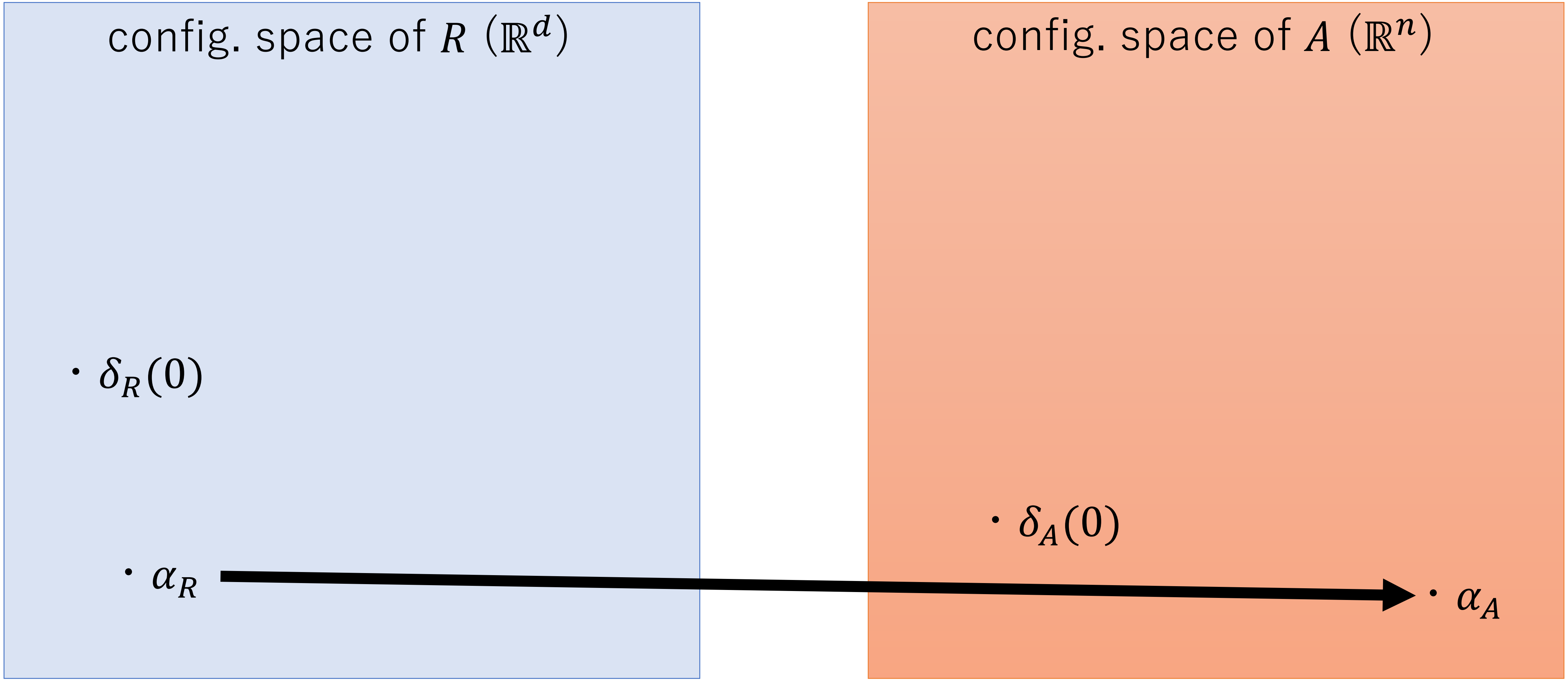
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



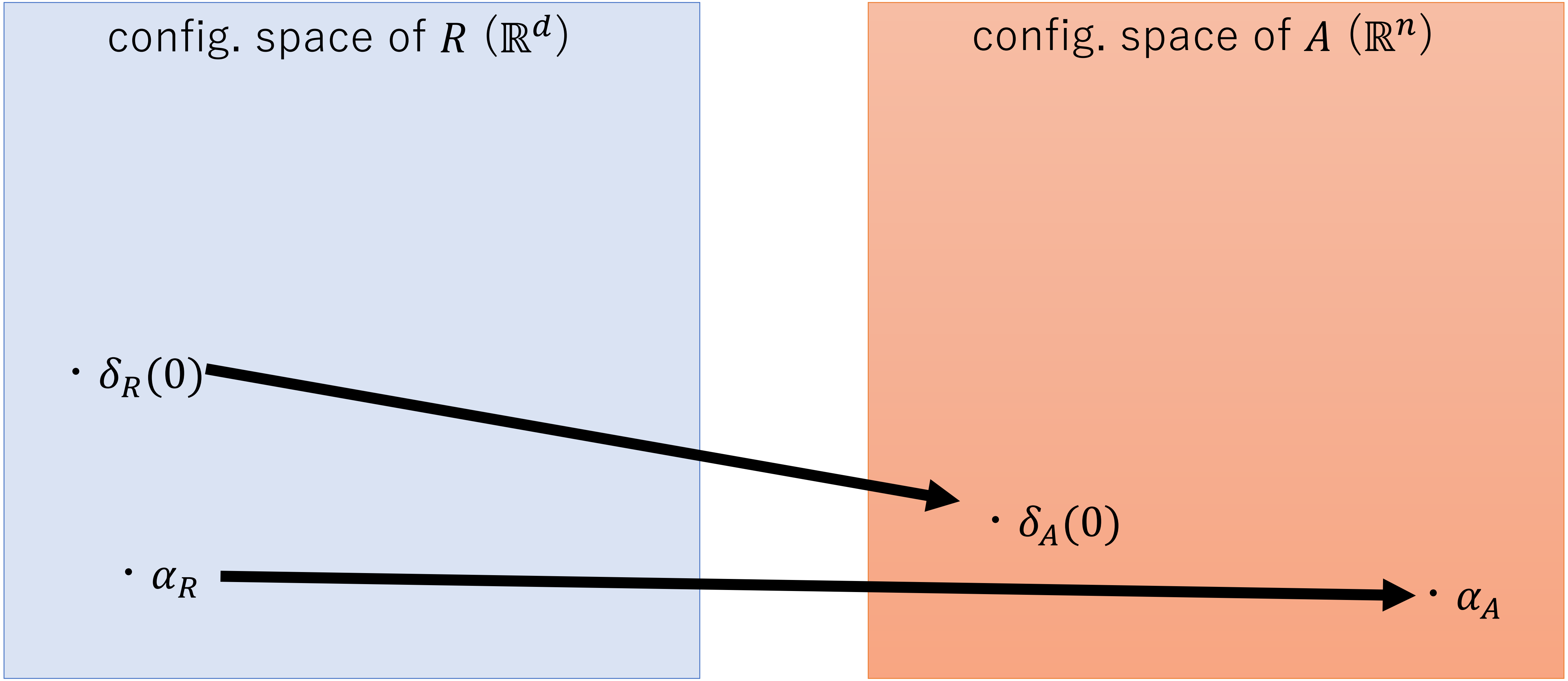
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



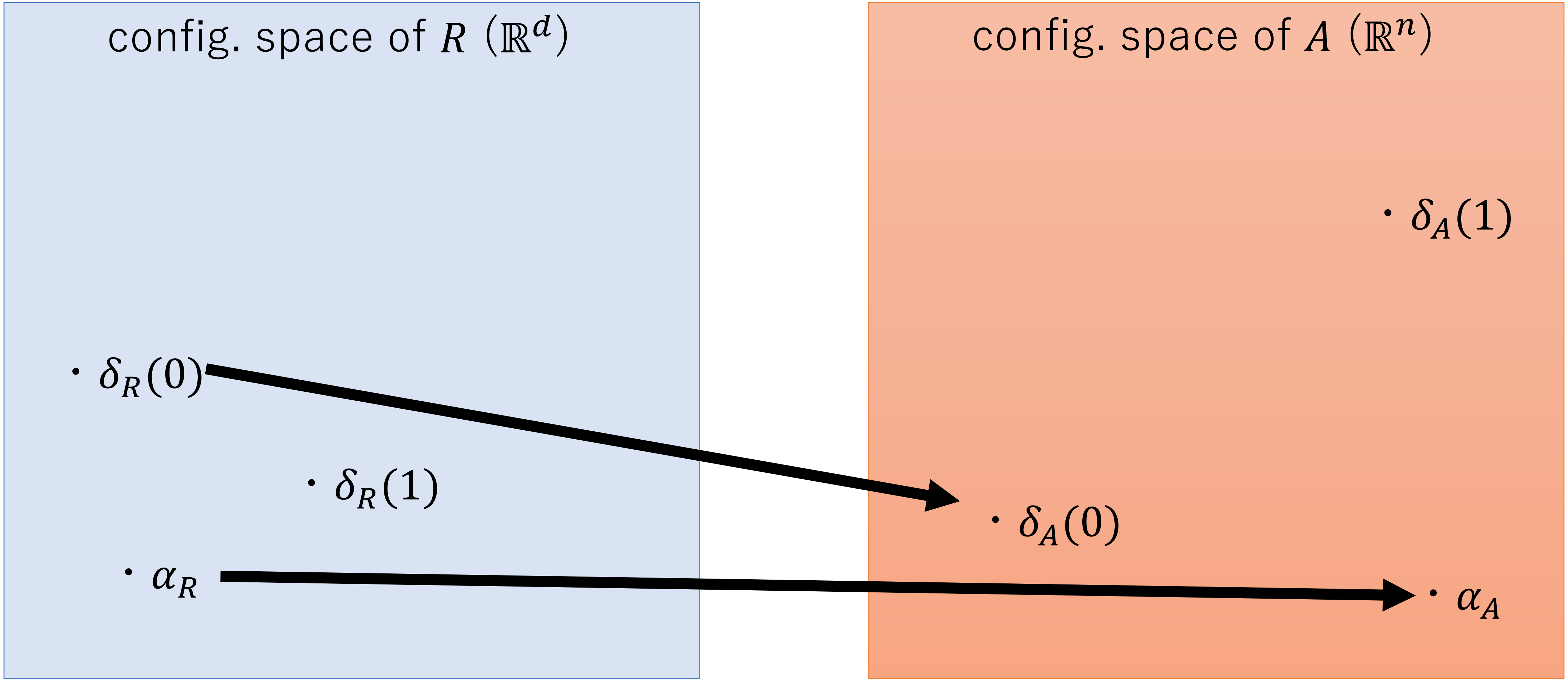
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



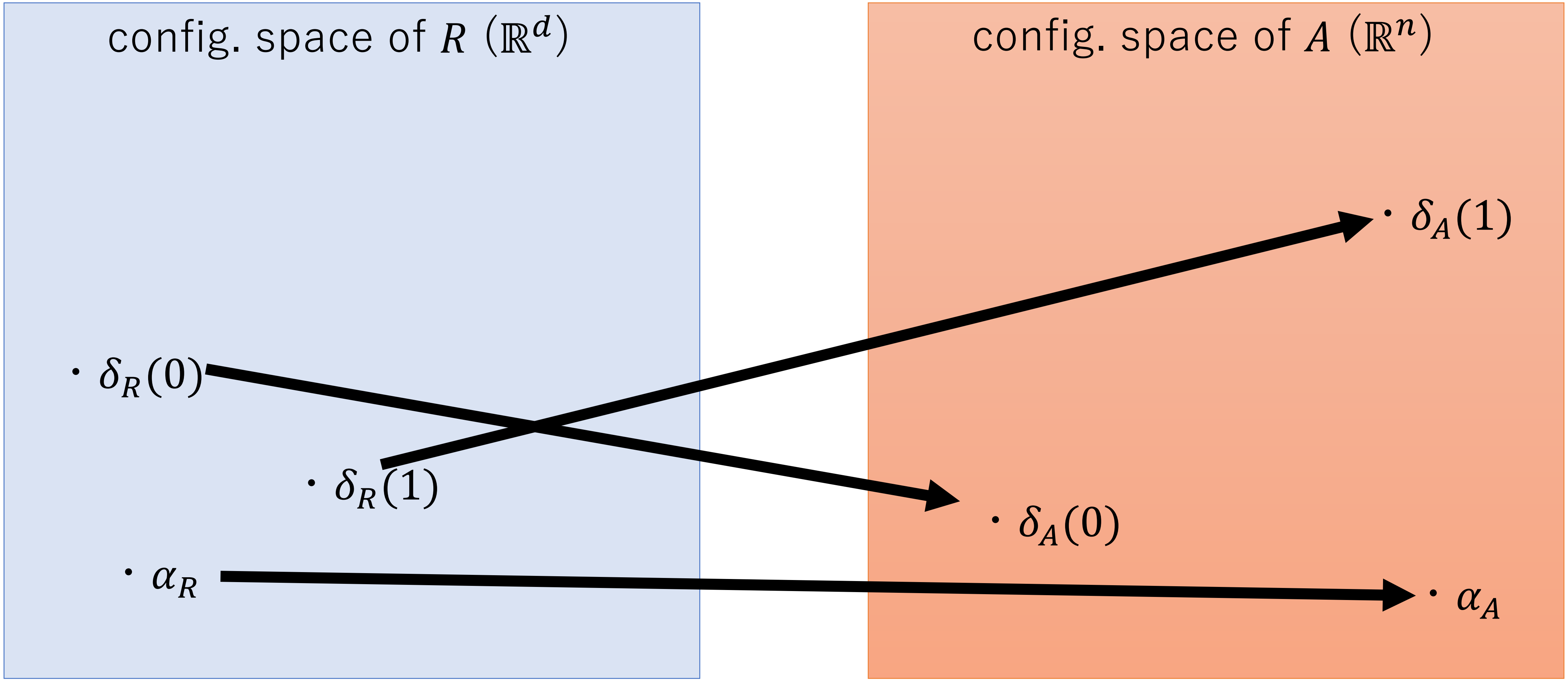
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



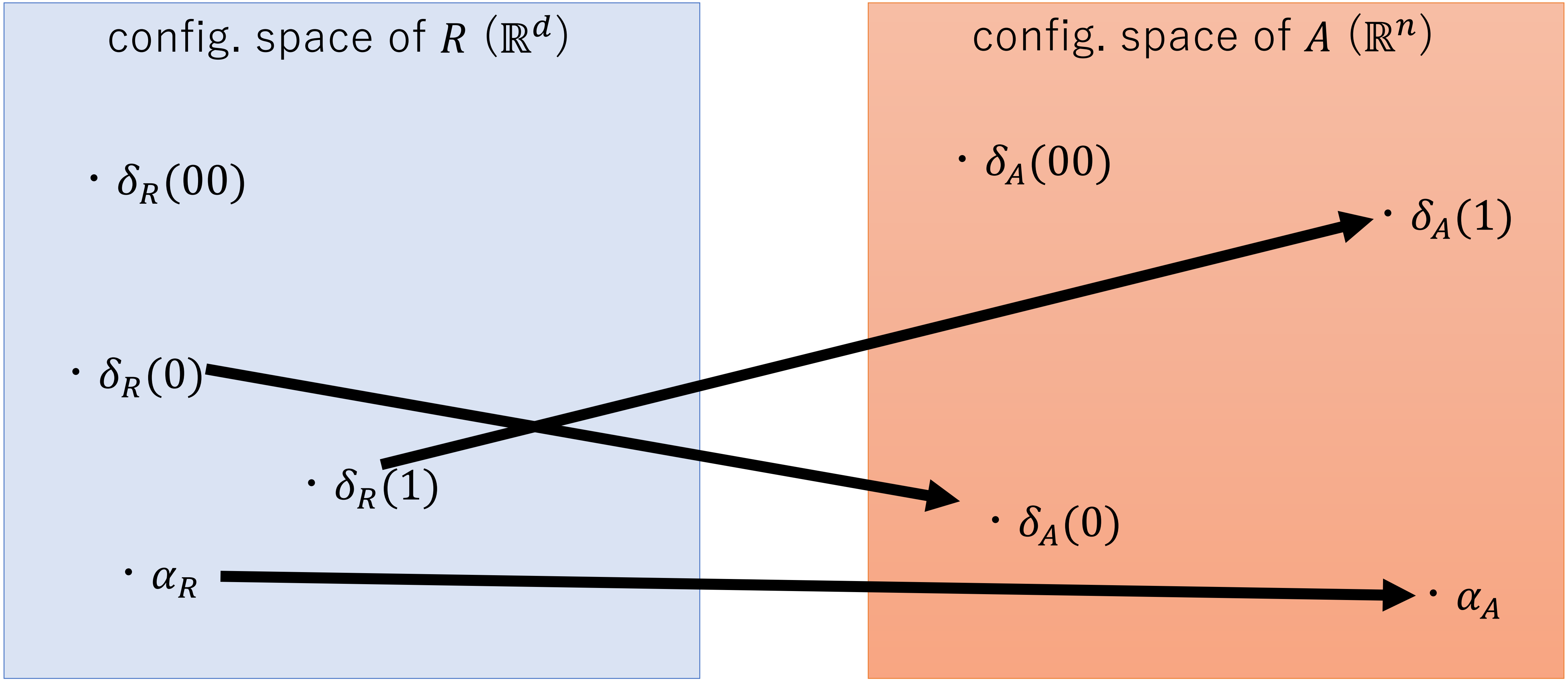
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



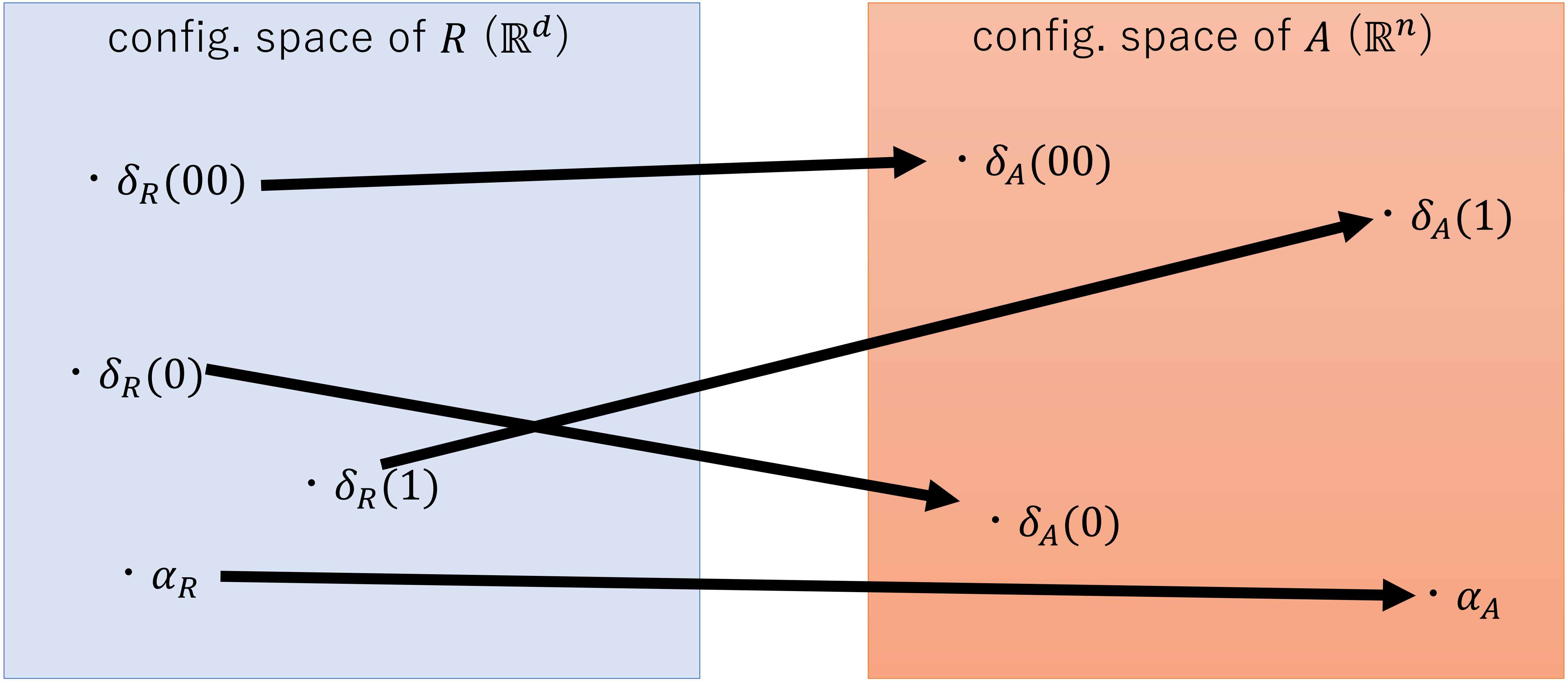
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



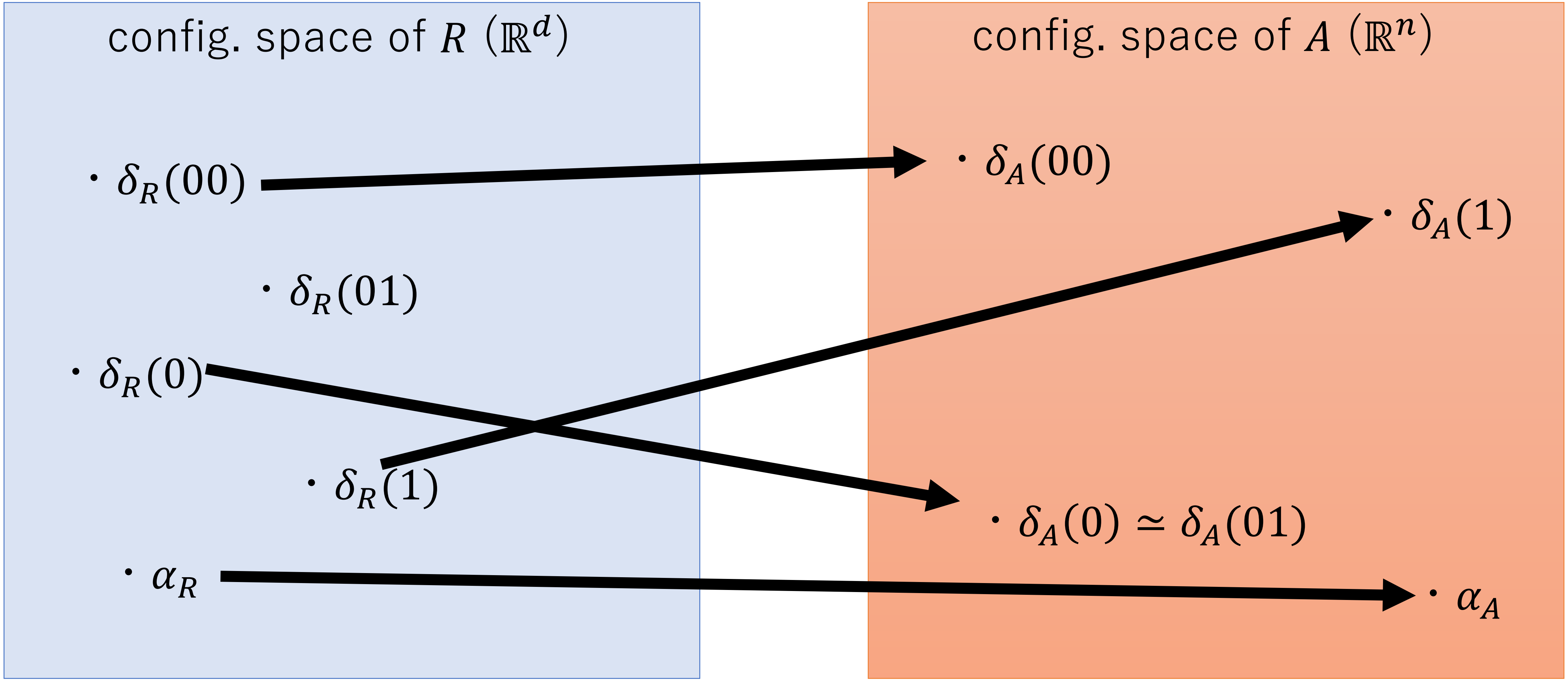
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



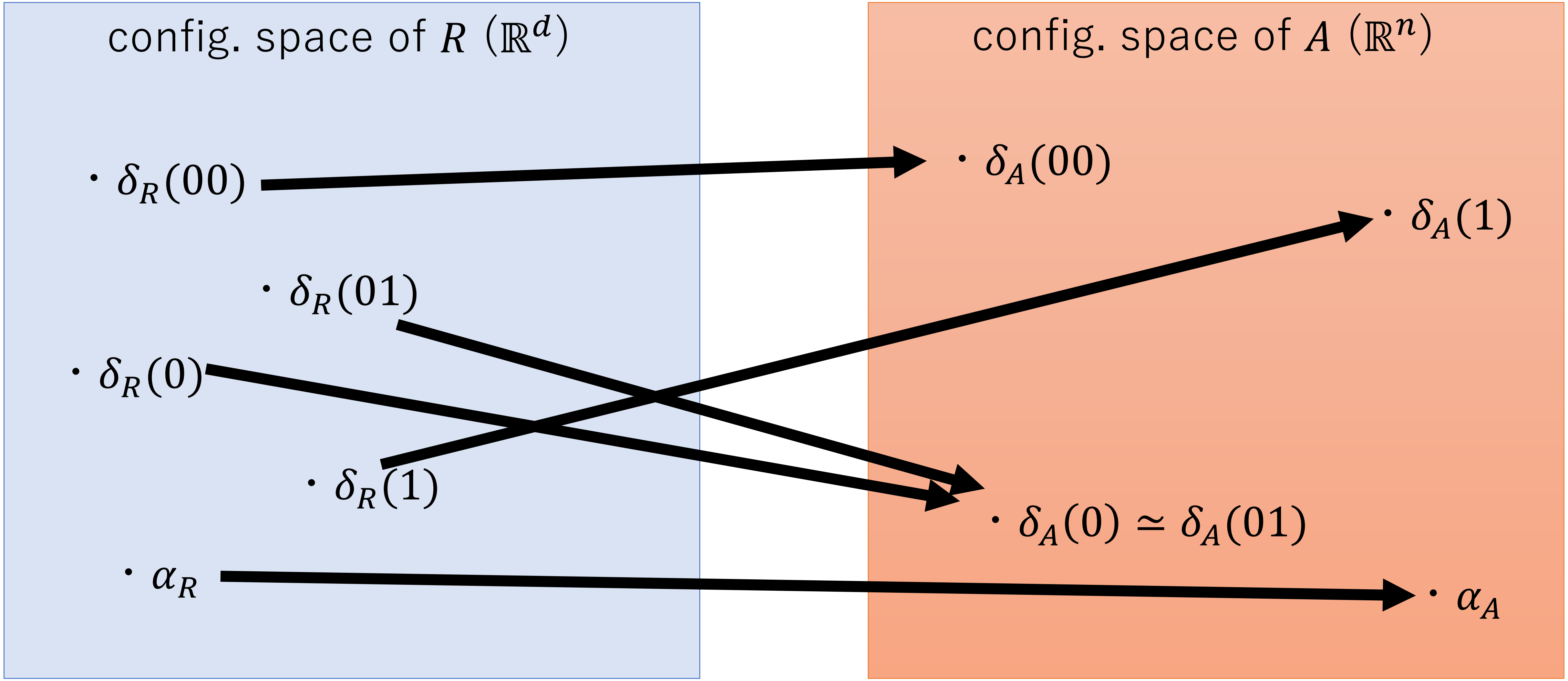
Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



Relation $\boldsymbol{p}$ between $\boldsymbol{R}$ and $\boldsymbol{A}$



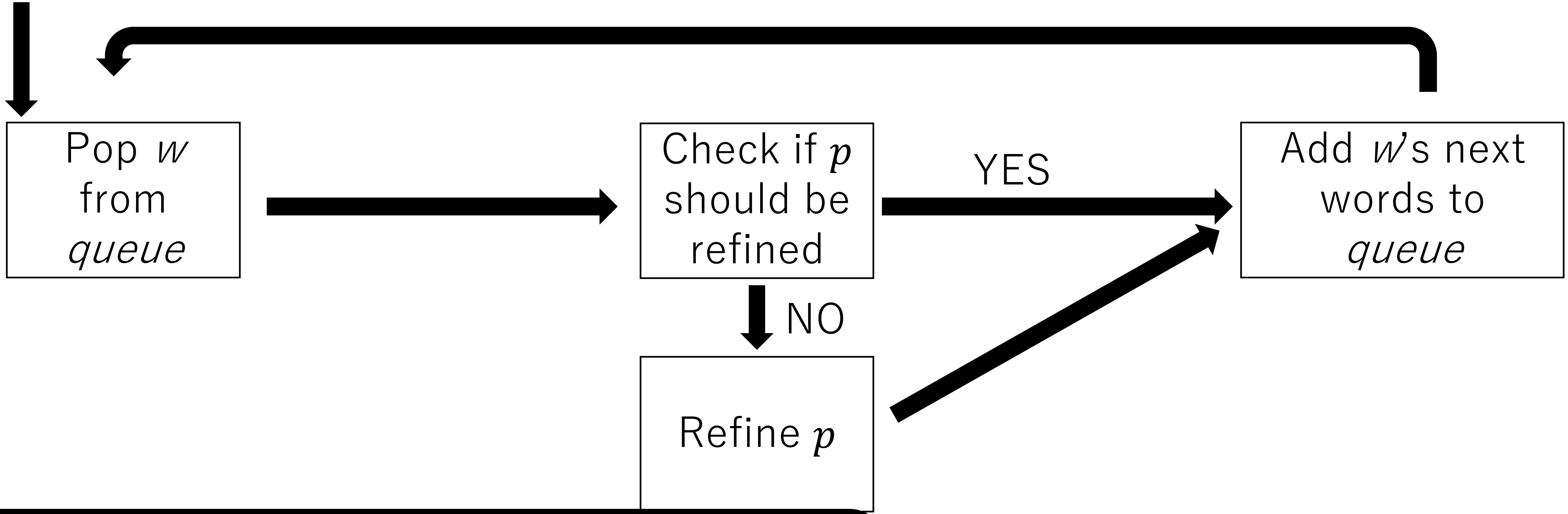
BFS-based Equivalence Query

Pop w from *queue*

Add w's next words to *queue*

Equivalence query proceeds based on Breadth-First Search

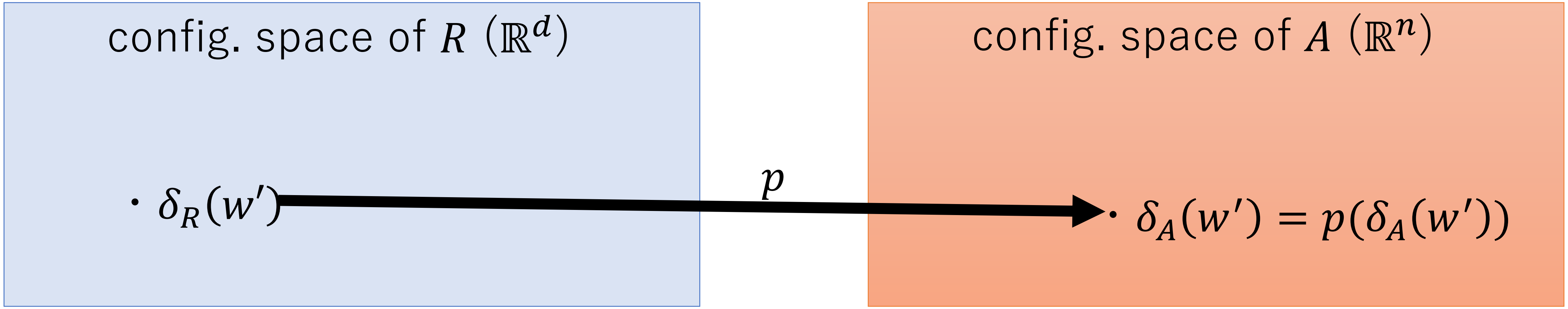
Maintaining p



We want it to satisfy

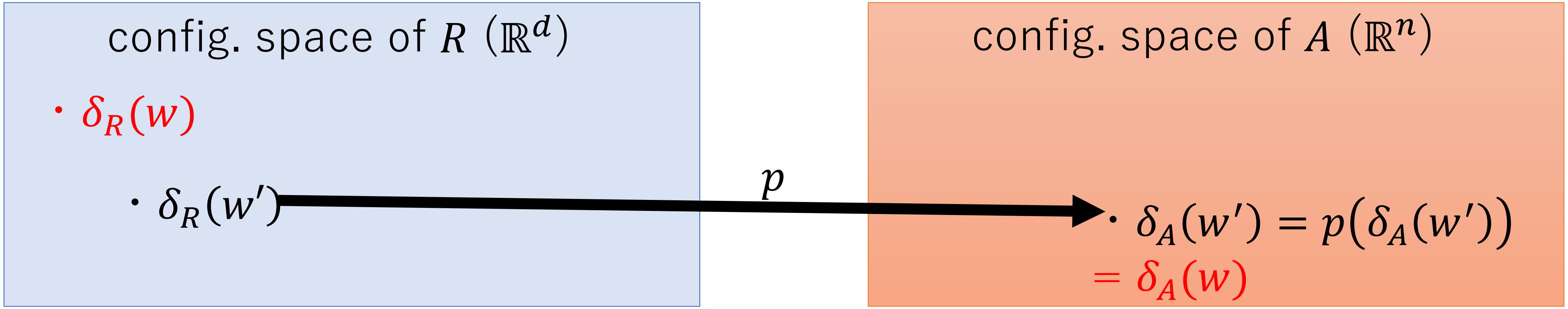
$$\forall w \in W.\ \mathrm{p}\big(\delta_R(\mathrm{w})\big) \simeq \delta_A(w)$$

Check if $\boldsymbol{p}$ should be refined



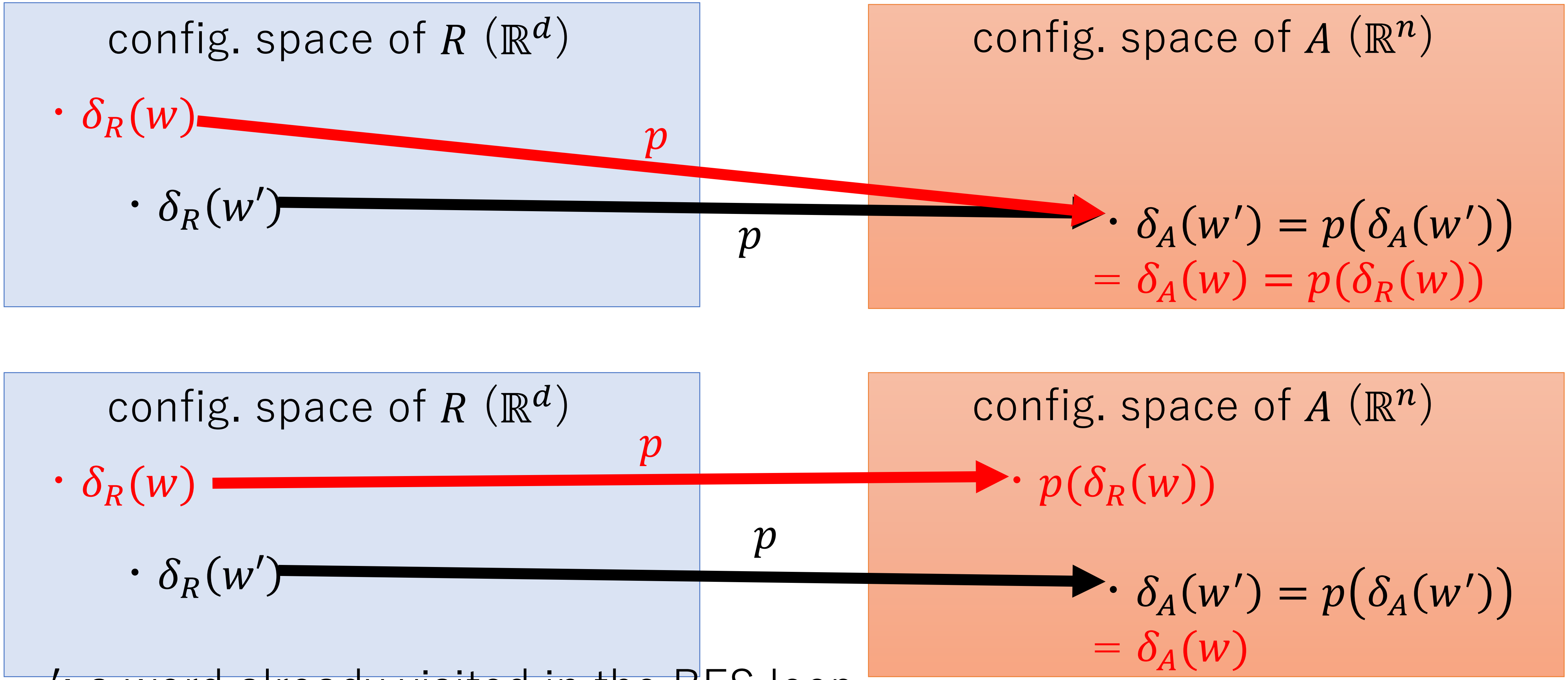
w': a word already visited in the BFS loop
w: a word just popped

Check if $\boldsymbol{p}$ should be refined



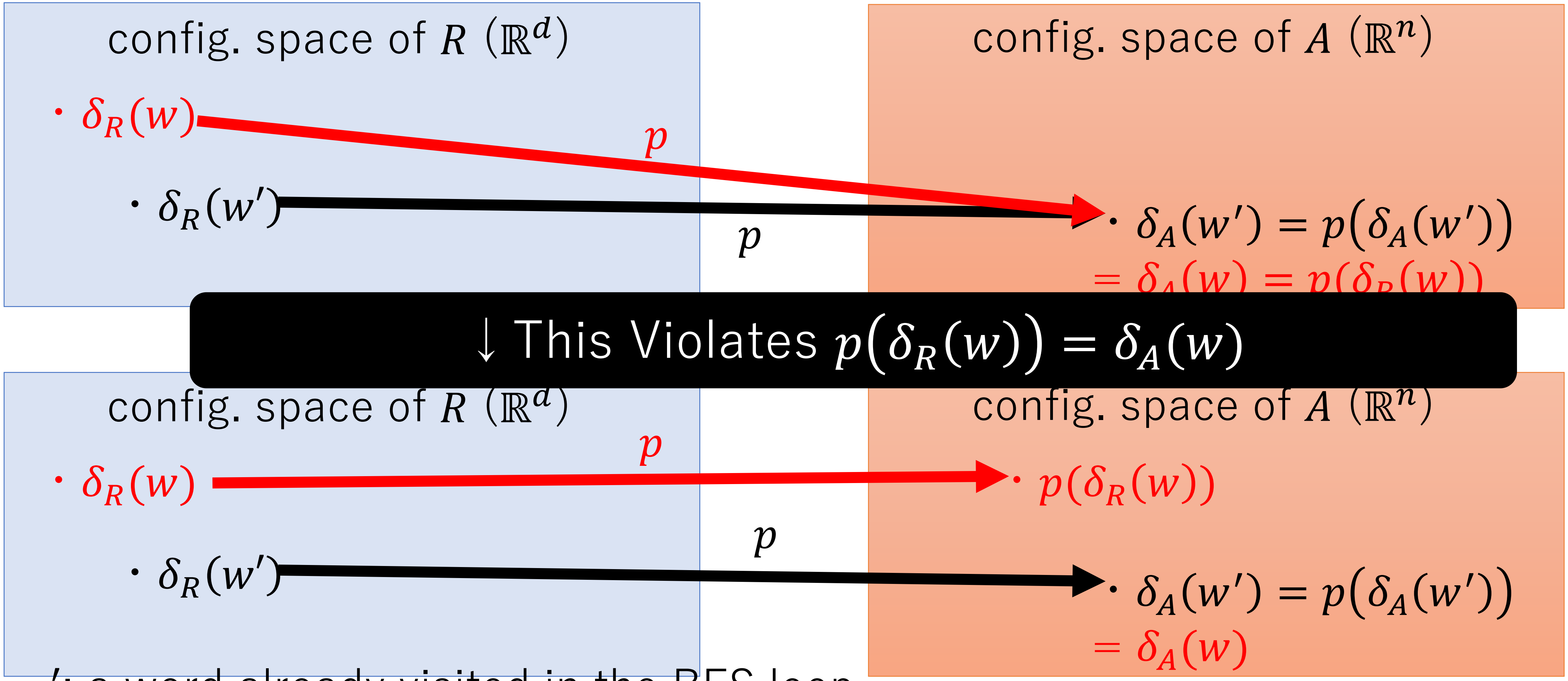
w': a word already visited in the BFS loop
w: a word just popped

Check if $\boldsymbol{p}$ should be refined



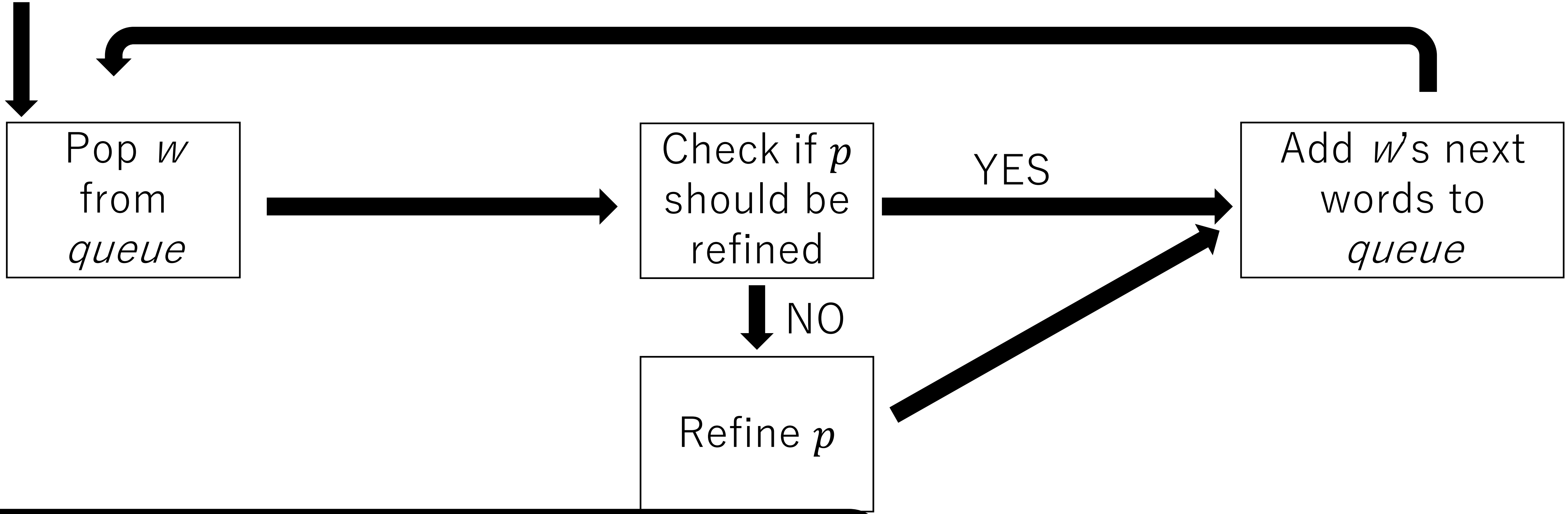
w': a word already visited in the BFS loop
w: a word just popped

Check if $\boldsymbol{p}$ should be refined



w': a word already visited in the BFS loop
w: a word just popped

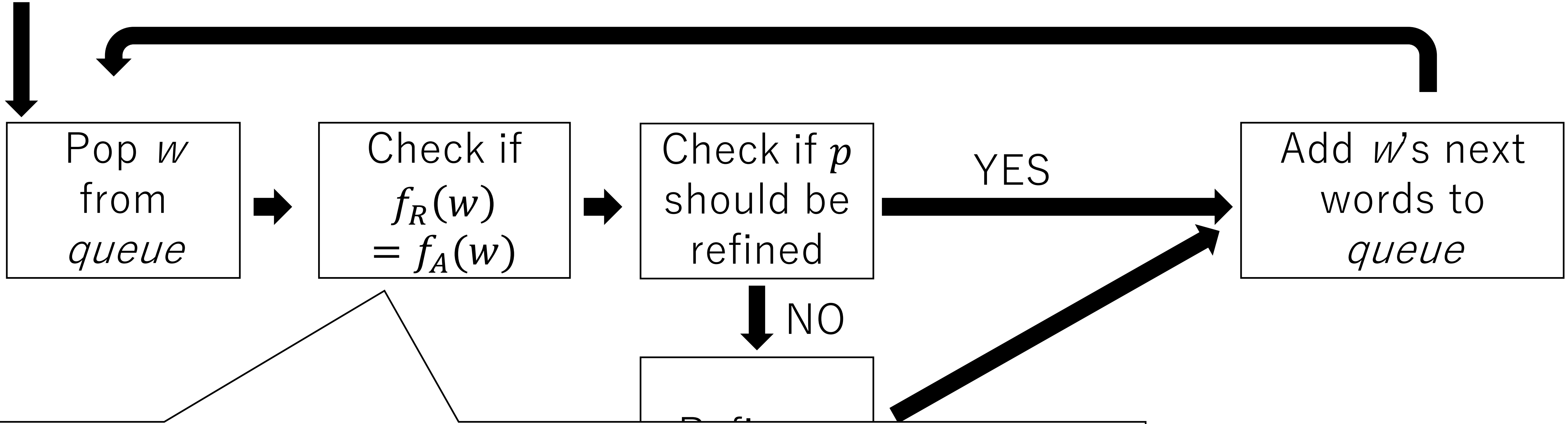
Maintaining $\boldsymbol{p}$



We want it to satisfy

$$\forall w \in W.\ \mathrm{p}\big(\delta_R(\mathrm{w})\big) \simeq \delta_A(w)$$

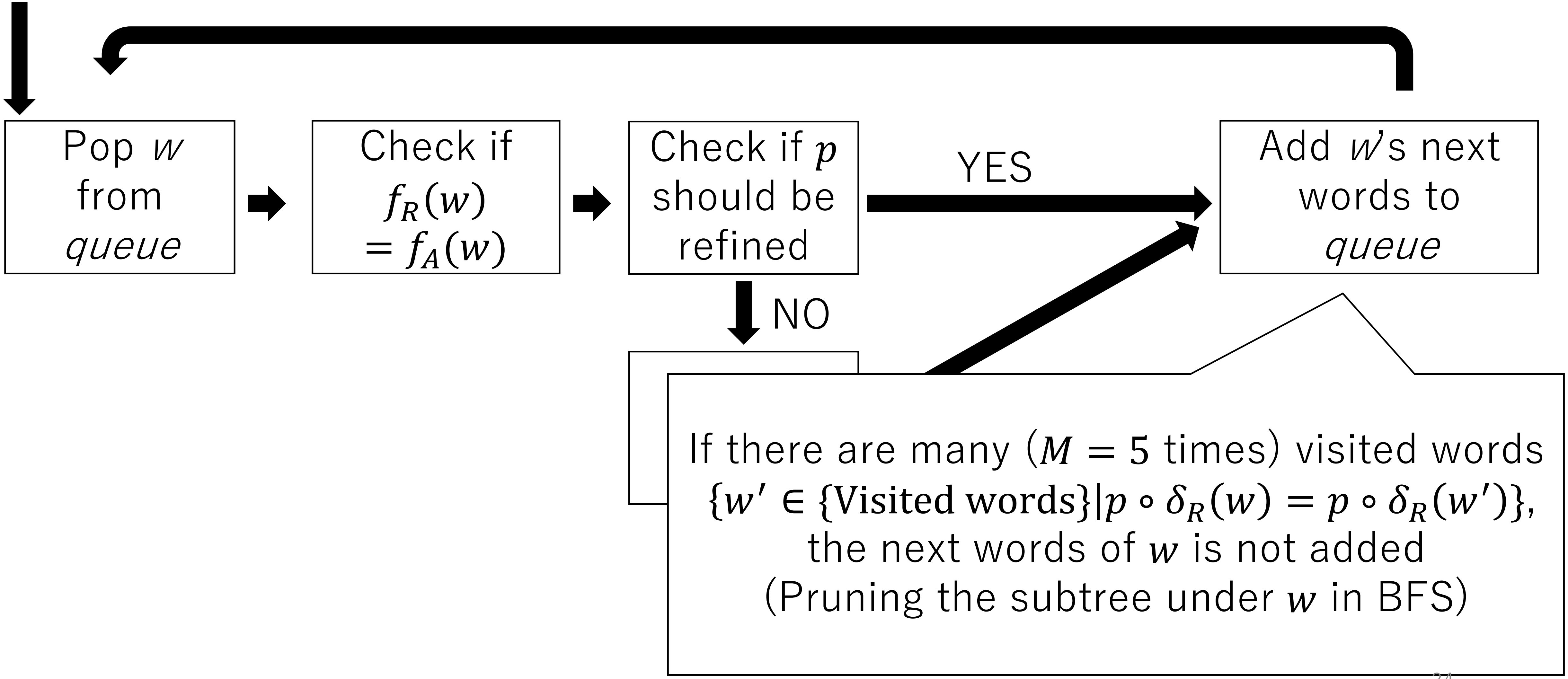
Finding Counterexample



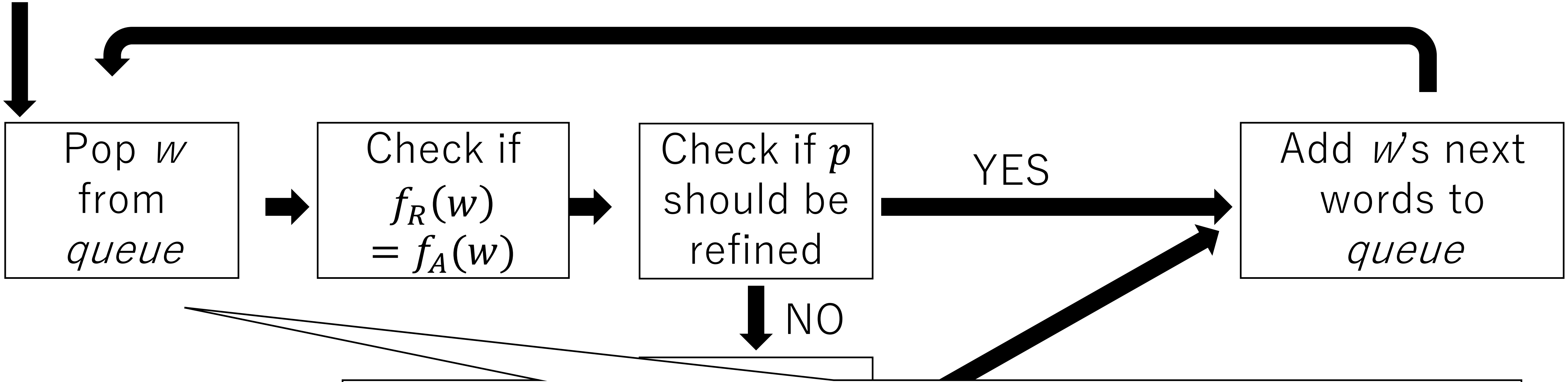
If $f_R(w) \neq f_A(w)$, returns w as a counterexample of the equivalence query.

$$e(A) = \begin{cases} \text{Equivalent}\,;\, f_R \simeq f_A \\ w''\,;\, f_R(w'') \neq f_A(w'') \end{cases}$$

Returning “Equivalent”



Returning "Equivalent"



When the *queue* is empty, all the trees are pruned and it returns "Equivalent".

$$e(A) = \begin{cases} \text{Equivalent}\,; f_R \simeq f_A \\ w''\,; f_R(w'') \neq f_A(w'') \end{cases}$$

Experiments (Target RNNs)

90 target RNNs to evaluate our algorithm are made by

1. Generate a random WFA A of size $n \in \{10, 20, 30\}$ and alphabet Σ of size $\mathrm{a} \in \{10,15,20,30,40,50\}$
2. Learn RNN $R(A)$ from A
3. Repeat Step 1-2 for each (n, s) 5 times.

RNNs consist of two-stacked LSTM with 50 cells.

Experiments (Settings)

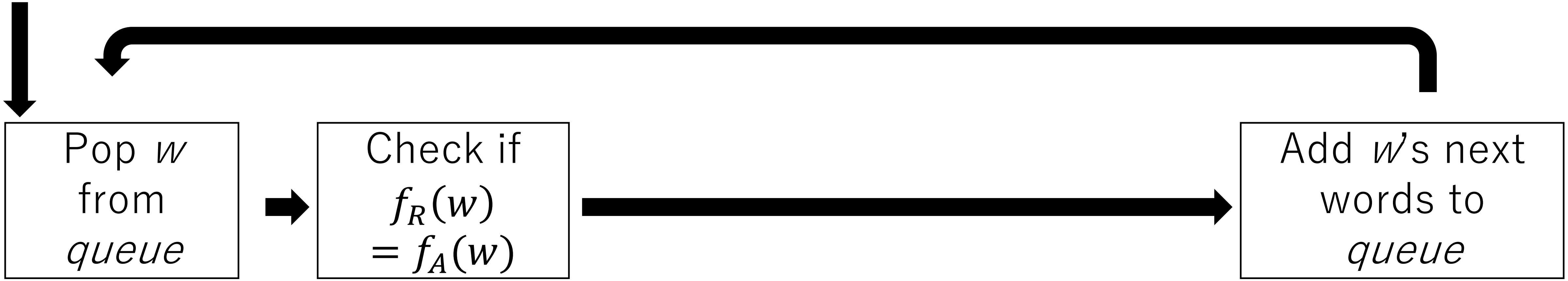
Methods

- Our algorithm with $M = 5$
- Baseline algorithm (comes later)

Evaluation

- Time to extract (timeout=10,000 sec.)
- Accuracy
  - If $\left|f_R(w) - f_{A(R)}(w)\right| < 0.05$ then it is “correct”
  - Calculated by randomly generating 1000 words
- Time to infer the words in $R(A), A(R(A))$

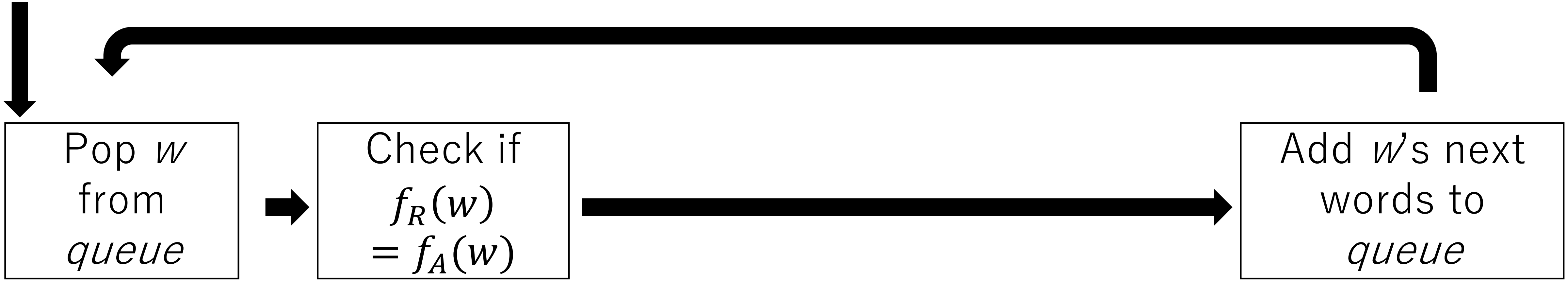
Experiments (Baseline algorithm)



If $f_R(w) \neq f_A(w)$, returns w as a counterexample of the equivalence query.

$$e(A) = \begin{cases} \text{Equivalent}\,;\, f_R \simeq f_A \\ w''\,;\, f_R(w'') \neq f_A(w'') \end{cases}$$

Experiments (Baseline algorithm)

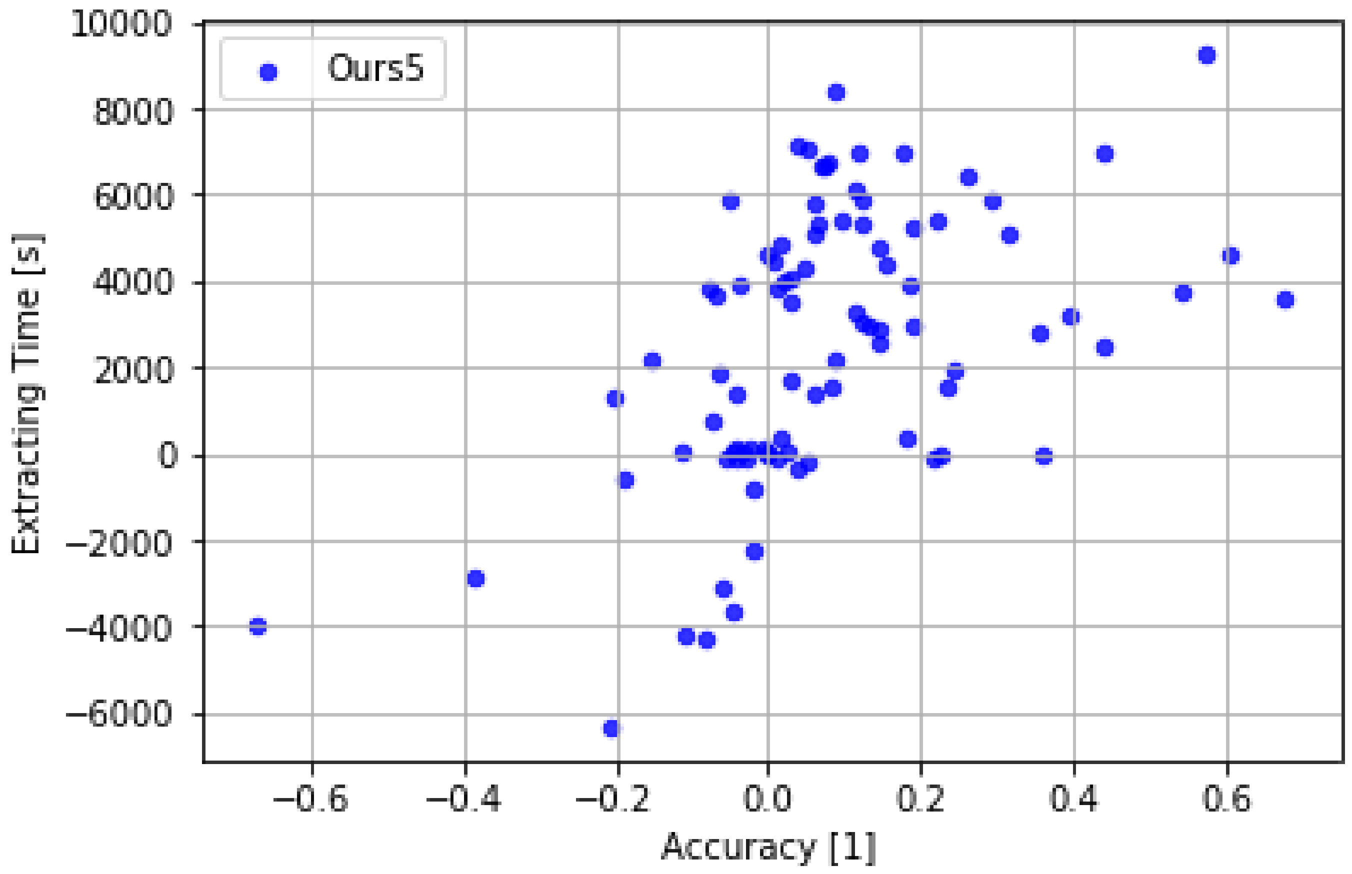


If $f_R(w) = f_A(w)$ in a row (1000 times), returns h as a counterexample of the equivalence query.

$$e(A) = \begin{cases} \text{Equivalent}\,;\, f_R \simeq f_A \\ w''\,;\, f_R(w'') \neq f_A(w'') \end{cases}$$

(If this happens, *queue* is preserved for the next invoke of eq-query)

Result (Overall)



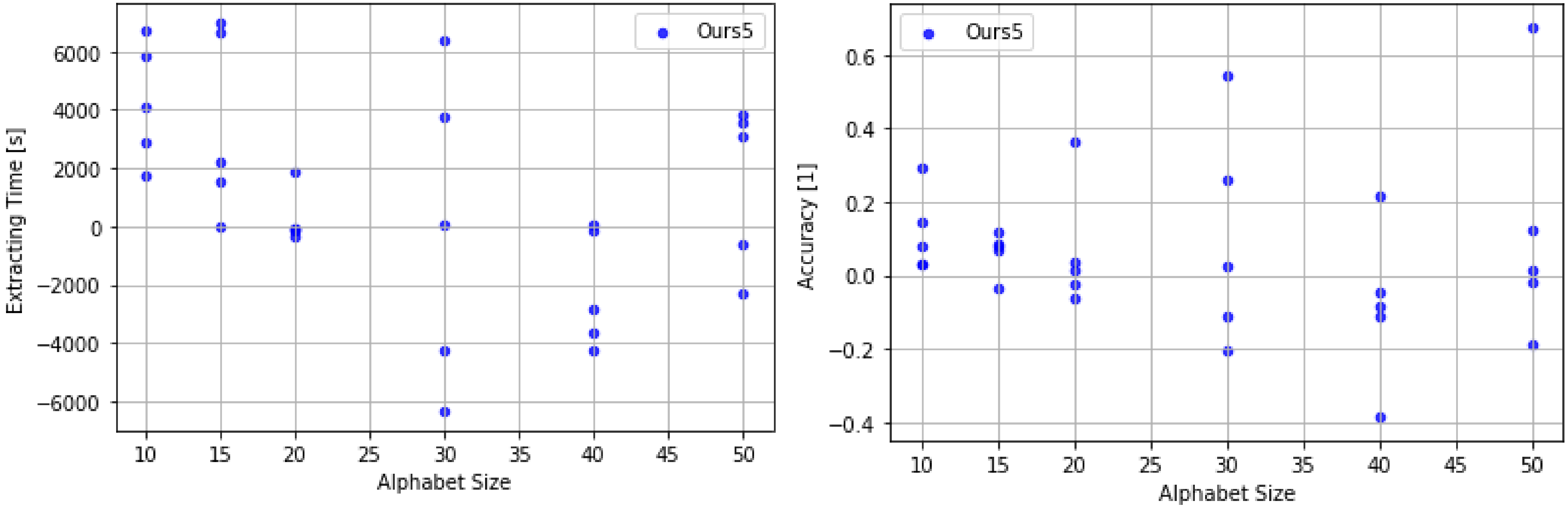
Difference of accuracy and extracting time between ours and baseline

Result (Overall)

Average (and Std)	Ours(M=5)	Baseline
Accuracy[%]	81.9% (std=18.8%)	74.1% (std=22.9%)
Time [s]	8805 (std=2220)	6277 (std=2966)

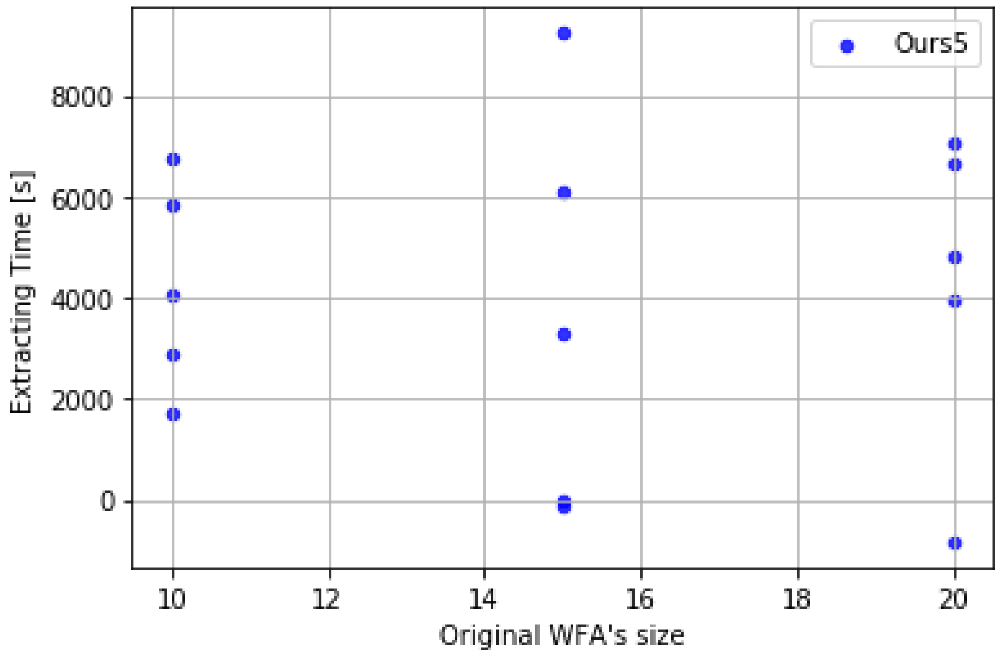
- The accuracy of "Ours (M=5)" exceeded those of "Baseline" in 59 tasks.
- The extracting time of "Ours (M=5)" longer than those of "Baseline" in 80 tasks.
- (90 tasks in total)

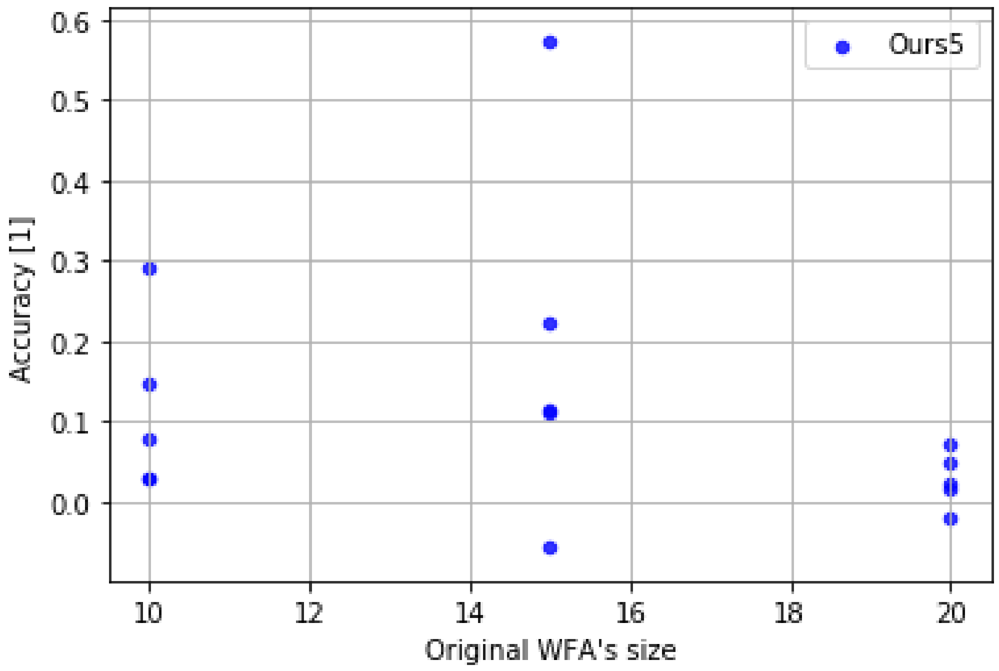
Result (WFA size $\boldsymbol{n = 10}$)



Difference of accuracy and extracting time between ours and baseline

Result (alphabet size $\boldsymbol{a = 10}$)





Difference of accuracy and extracting time between ours and baseline

Time to Infer a Value from a Word

- To test our motivation “Getting **lighter** (faster to infer) model of an RNN” is feasible.
- We compared the time to compute $f_R(w)$ and $f_{A(R)}(w)$ for 1,000 words whose lengths are ≤ 20.

	Average
Time on RNN R [s]	32.0 (std=2.0)
Time on WFA $A(R)$ [s]	0.028 (std=0.007)

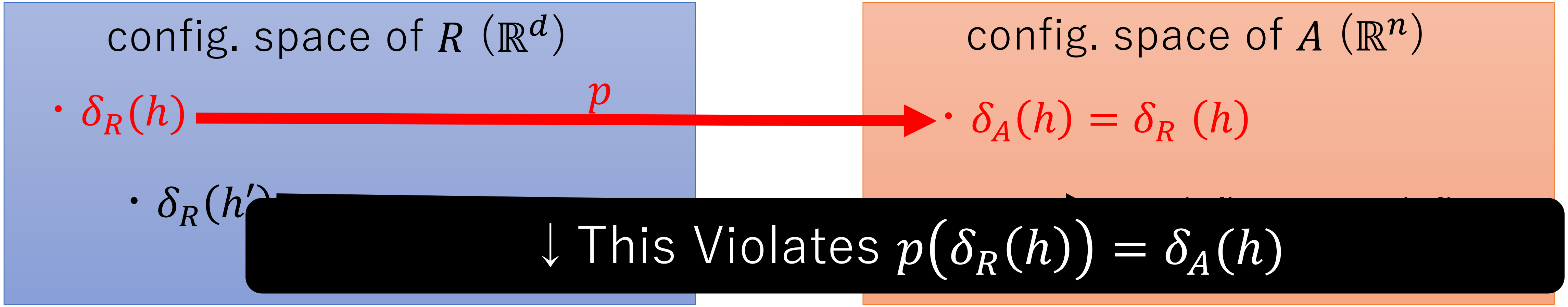
Conclusion

- Proposed a method to extract the WFA $A(R)$ from a given RNN R so that $f_{A(R)} \simeq f_R$.
- Compared our method to the baseline algorithm in the accuracy and time
  - Our algorithm achieved higher accuracy and took more time than the baseline.
- The extracted WFA $A(R)$ took less time to infer values than the original RNN R

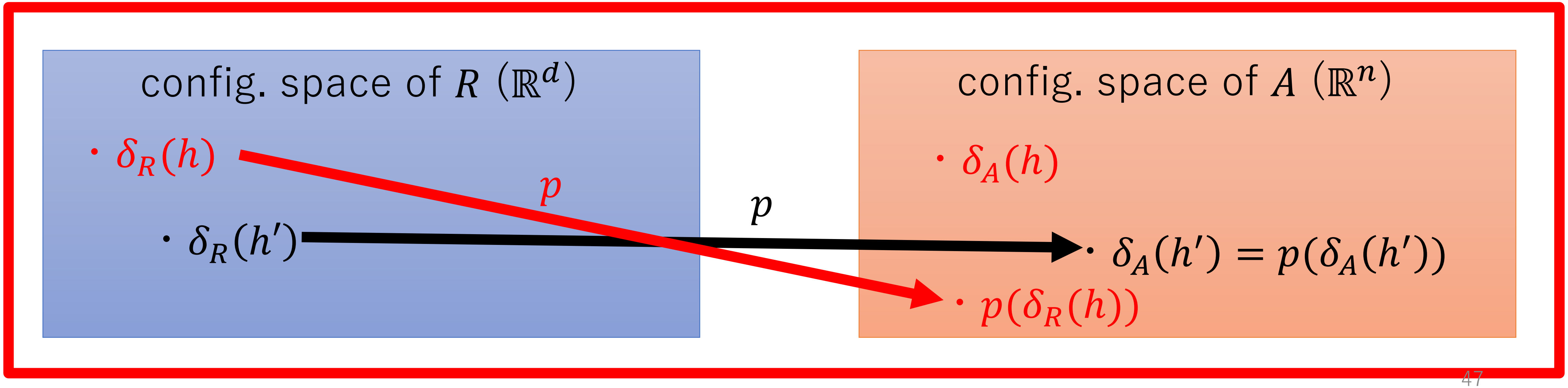
Future Work

- Adding experiment
  - To reveal the overall tendency clearly
  - To reveal what is happening when the accuracy is quite low
- Adding the idea of bisimulation to p
- Think of questionable parts in the loop?
  - Refining p at the different timing could be better?
- Modifying Balle and Mohri's algorithm to generate **probabilistic** WFA
- Finding good hyper parameter M experimentally or theoretically

“Checking if $\boldsymbol{p}$ is OK” could be like this?



↓ This Violates $p(\delta_R(h)) = \delta_A(h)$



Def. of WFA

- WFA A is *probabilistic* if
  - $\alpha \cdot \mathbf{1} = 1$
  - For all $\sigma \in \Sigma$, the sums of rows are 1
  - $0 \leq \beta \leq 1$ ■

<u>For example:</u>

- $\Sigma = \{0, 1\}, \alpha = \begin{pmatrix} 0.8 & 0.2 \end{pmatrix}, \beta = \begin{pmatrix} 0.9 \\ 0.7 \end{pmatrix}, A_0 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, A_1 = \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix}$