Probing RNN Encoder-Decoder Generalization of Subregular Functions Using Reduplication

Max Nelson, Hossep Dolatian, Jonathan Rawski, Brandon Prickett

University of Massachusetts Amherst, Stony Brook University

January 5, 2020

Talk in a Nutshell

- Formal Languages/Automata:
  - Necessary and sufficient conditions on computable functions
  - Provide target function classes for generalization/learning
  - transparent, analytical guarantees independent of the machine
- Recurrent Neural Network/ finite-state connections
- What is the generalization capacity of RNN Encoder-Decoders?

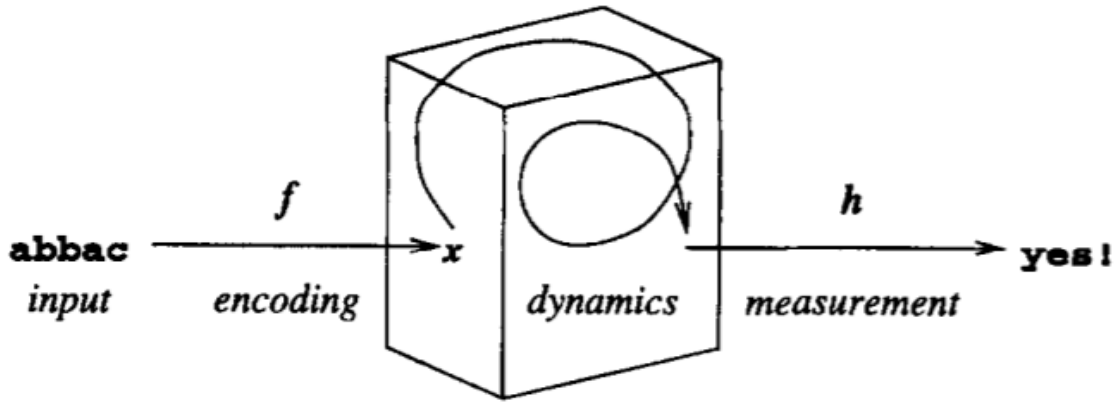
Encoder-decoders and Subregular Reduplication

- Reduplication: variable-length subregular copy functions
- Vanilla Encoder-Decoders struggle to capture generalizable reduplication, networks with attention reliably succeed
- Attention weights mirror subregular 2-way FST processing, suggests they are approximating them

RNN and regular languages

Language: *Does string w belong to stringset (language) L*

- Computed by different classes of grammars (acceptors)



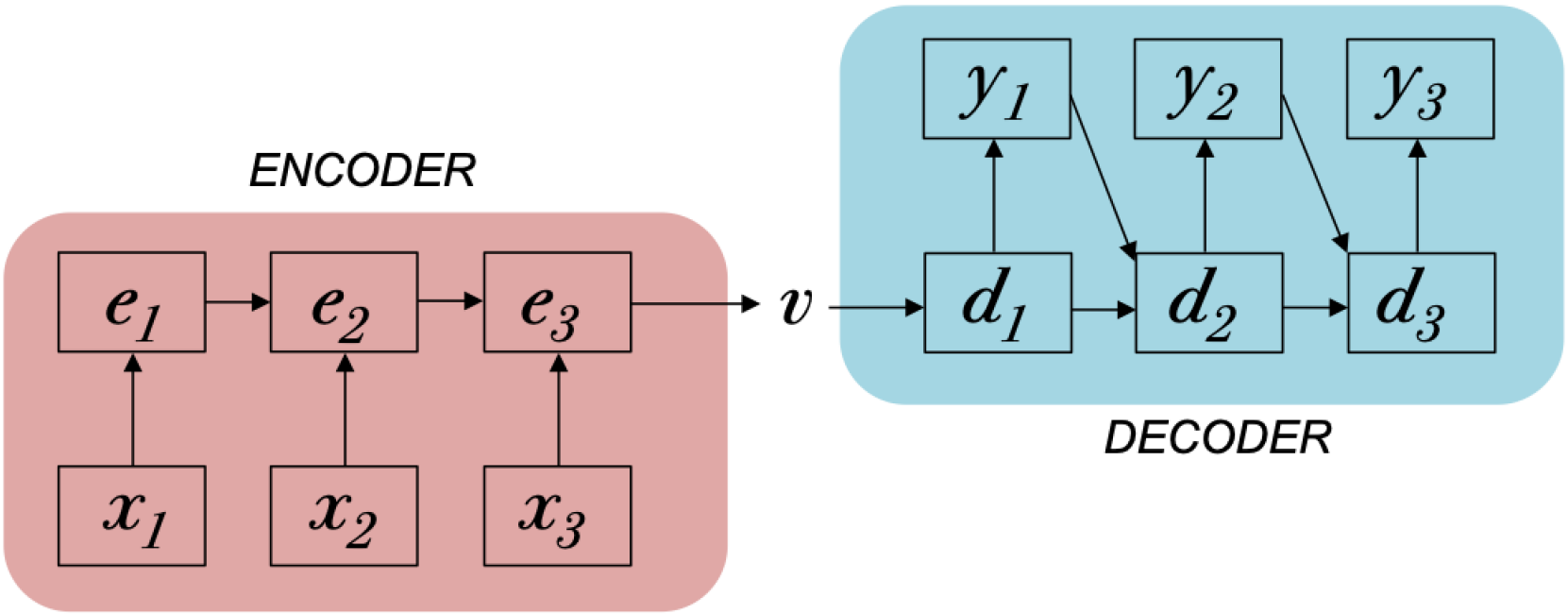
How expressive are RNNs?

Turing complete	infinite precision+time	(Siegelmann, 2012)
counter languages	LSTM/ReLU	(Weiss et al., 2018)
Regular	SRNN/GRU	(Weiss et al., 2018)
	asymptotic acceptance	(Merrill, 2019)
Weighted FSA	Linear 2nd Order RNN	(Rabusseau et al., 2019)
Subregular	LSTM problems	(Avcu et al., 2017)

pic credit: Casey 1996

RNN Encoder-Decoder and Transducers

- Function: *Given string w, generate f (w) = v*
  - = accepted pairs of input & output strings
  - Computed by di□erent classes of grammars (transducers)
- Recurrent encoder maps a sequence to $v \in \mathbb{R}^n$, recurrent decoder language model conditioned on v (Sutskever et al., 2014)
- How expressive are they?



Brief typology of reduplication

- Reduplication is typologically common[1]
- Basic division: partial vs. total reduplication

(1) Partial reduplication = bounded copy

a.	CV:	guyon → gu□guyon 'to jest'→'to jest repeatedly'	(Sundanese)
b.	Foot:	(gindal)ba → gindal□gindalba 'lizard sp.' → 'lizards'	(Yidin)
c.	Syllable	vam.se → vam□vamse 'hurry' → 'hurry (habitual)'	(Yaqui)

(2) Total reduplication = unbounded copy

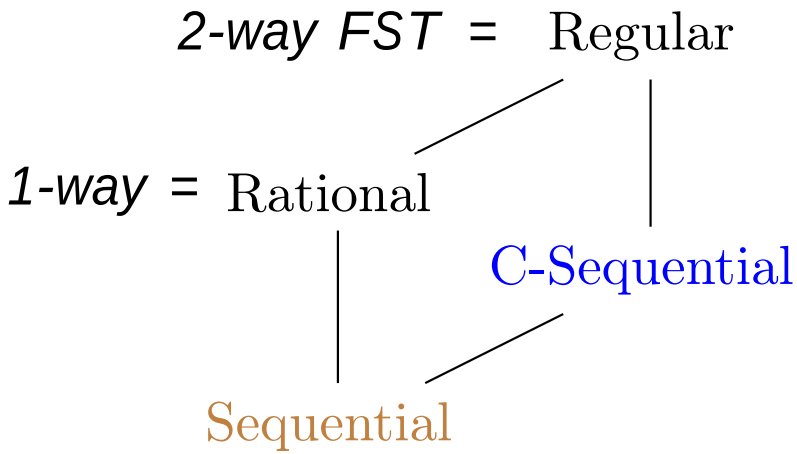
a.		wanita→wanita□wanita 'woman'→'women'	(Indonesian)

[1] (Moravcsik, 1978; Rubino, 2013)

Subregular computing of reduplication

- Why reduplication (Red)?
  - inhabits subclasses of regular string-to-string functions
  - computed by restricted types of Finite-State Transducers

1. 1-way FST: reads input once in one direction
   - computes Rational functions
     e.g., Sequential functions like partial Red
2. 2-way FST: reads multiple times, moves back and forth
   - computes Regular functions
     e.g., Concatenated-Sequential functions like partial & total Red

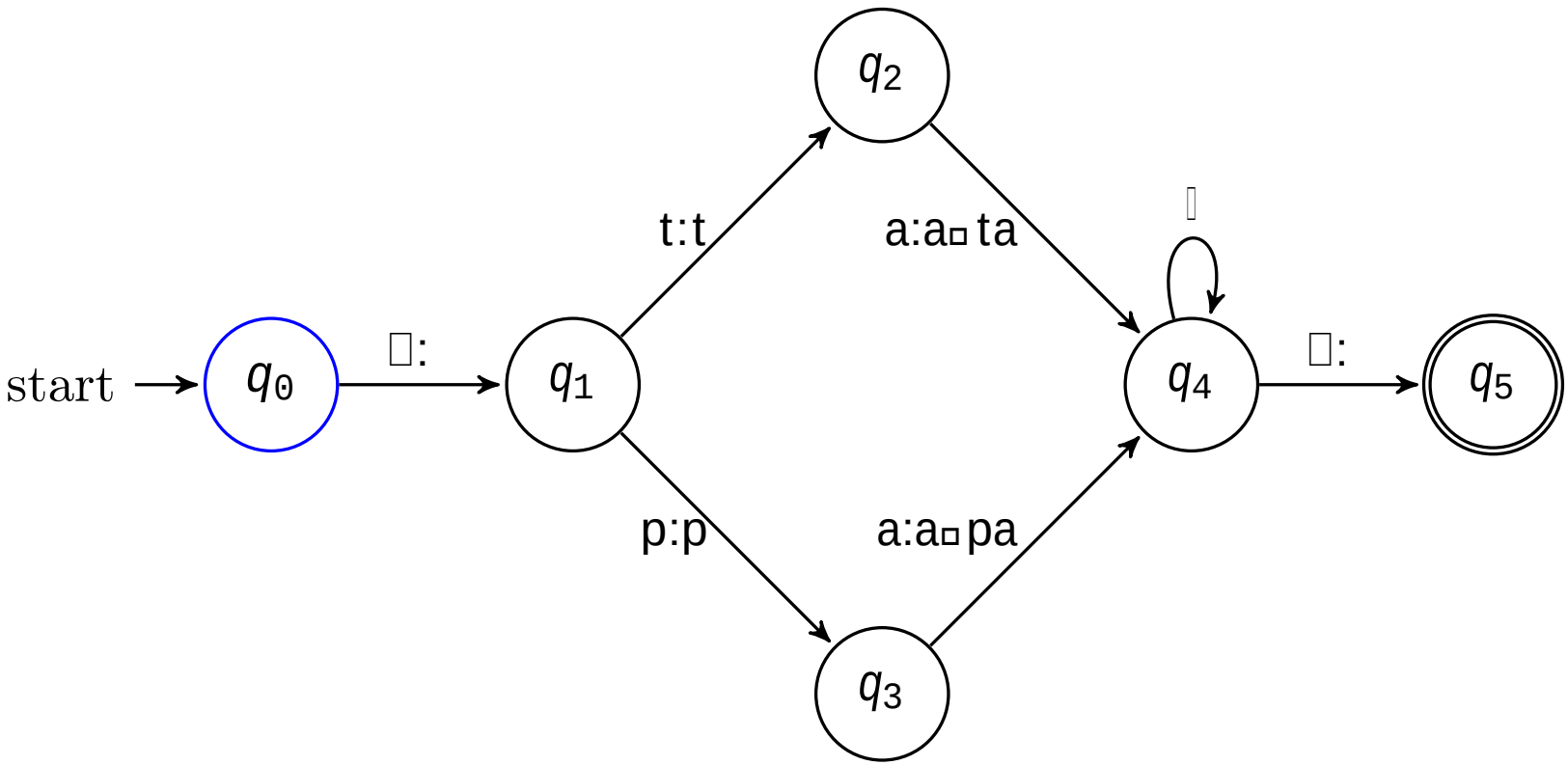


Partial reduplication with 1-way FSTs

- Working example: pat→[pa□pat]

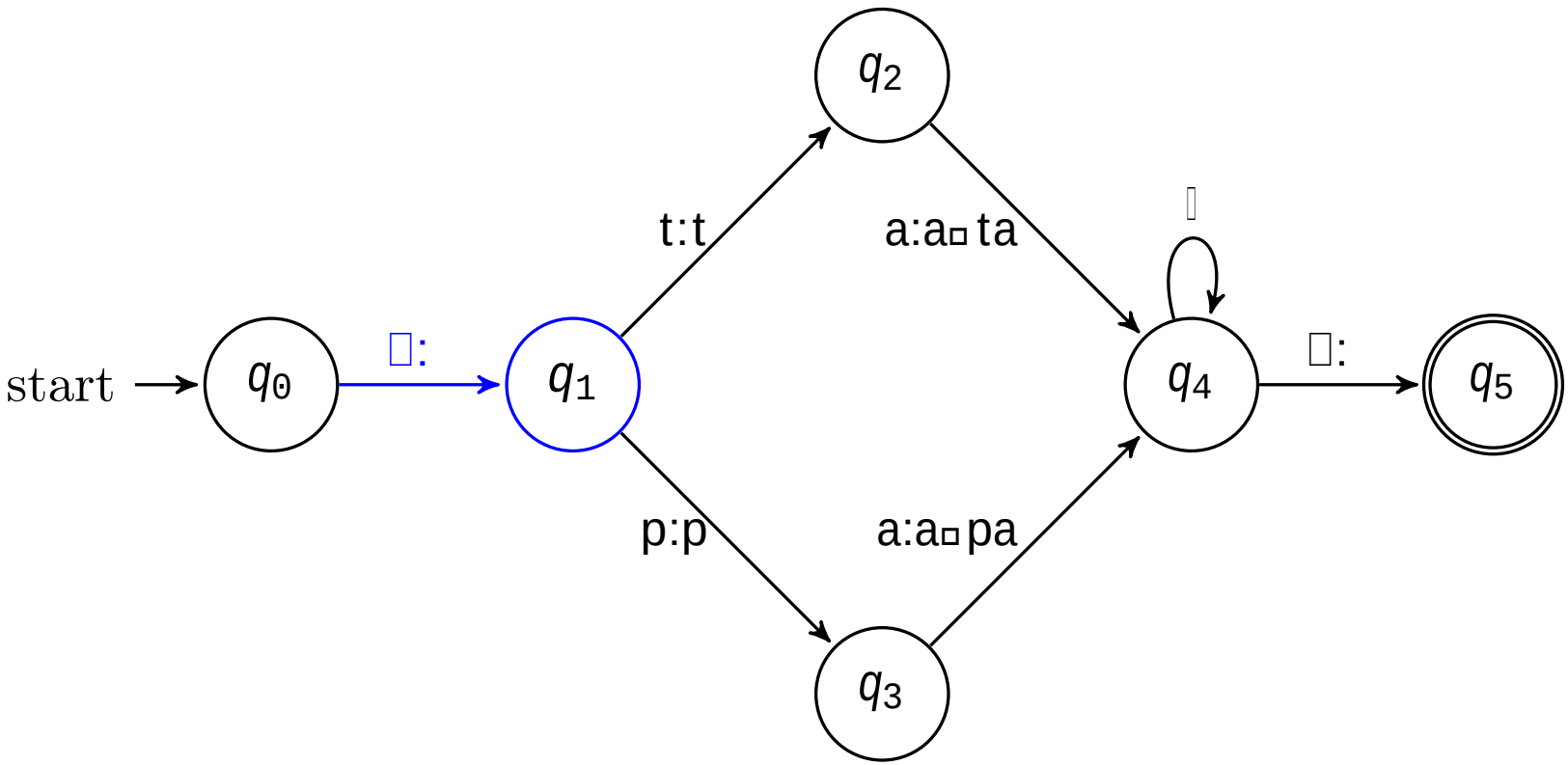
Partial reduplication with 1-way FSTs

- Working example: pat→[pa□pat]
  Input: □ p a t □
  Output:



Partial reduplication with 1-way FSTs

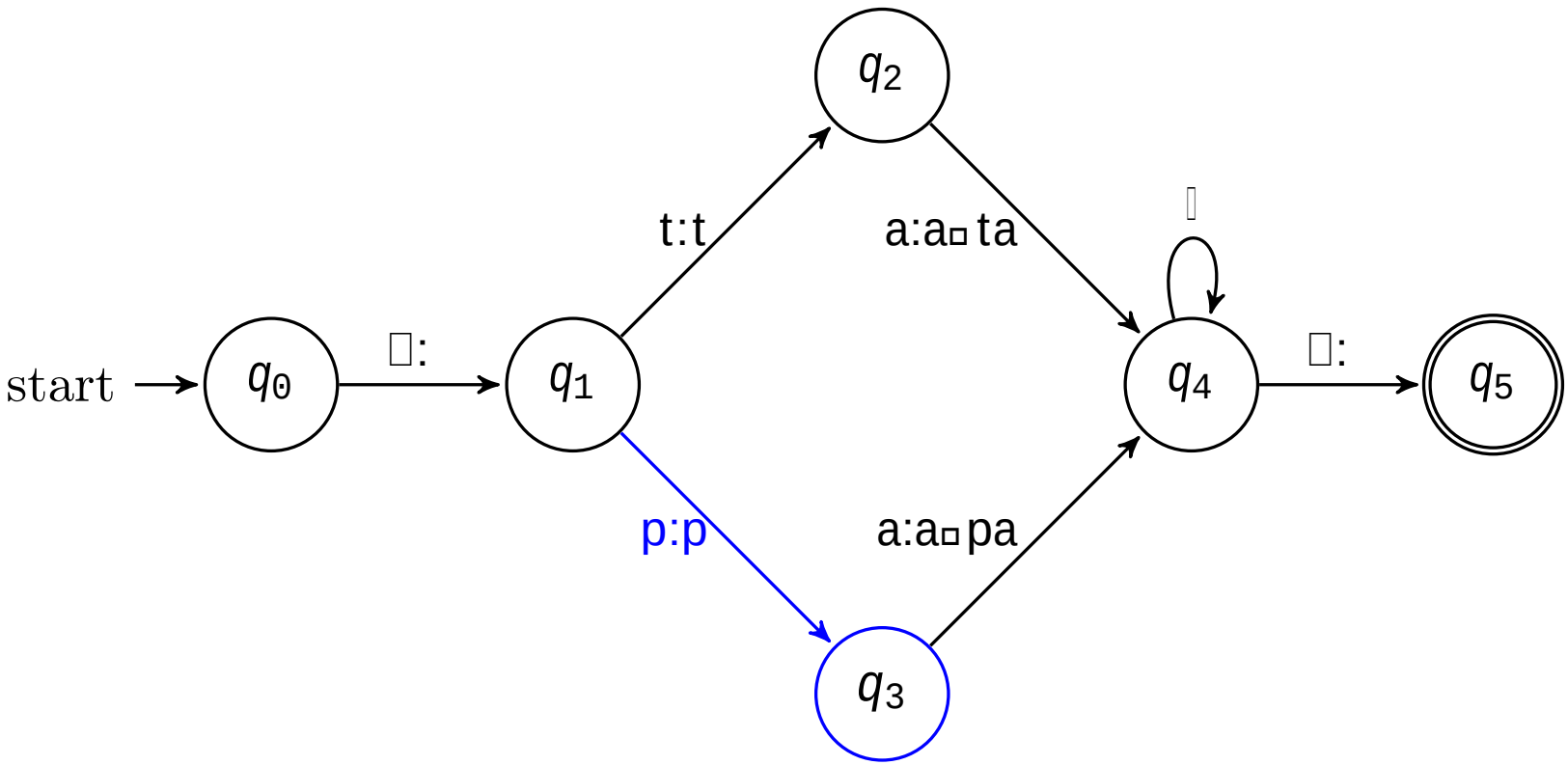
- Working example: pat→[pa□pat]
  Input: □ p a t □
  Output:



Partial reduplication with 1-way FSTs

- Working example: pat→[pa□pat]

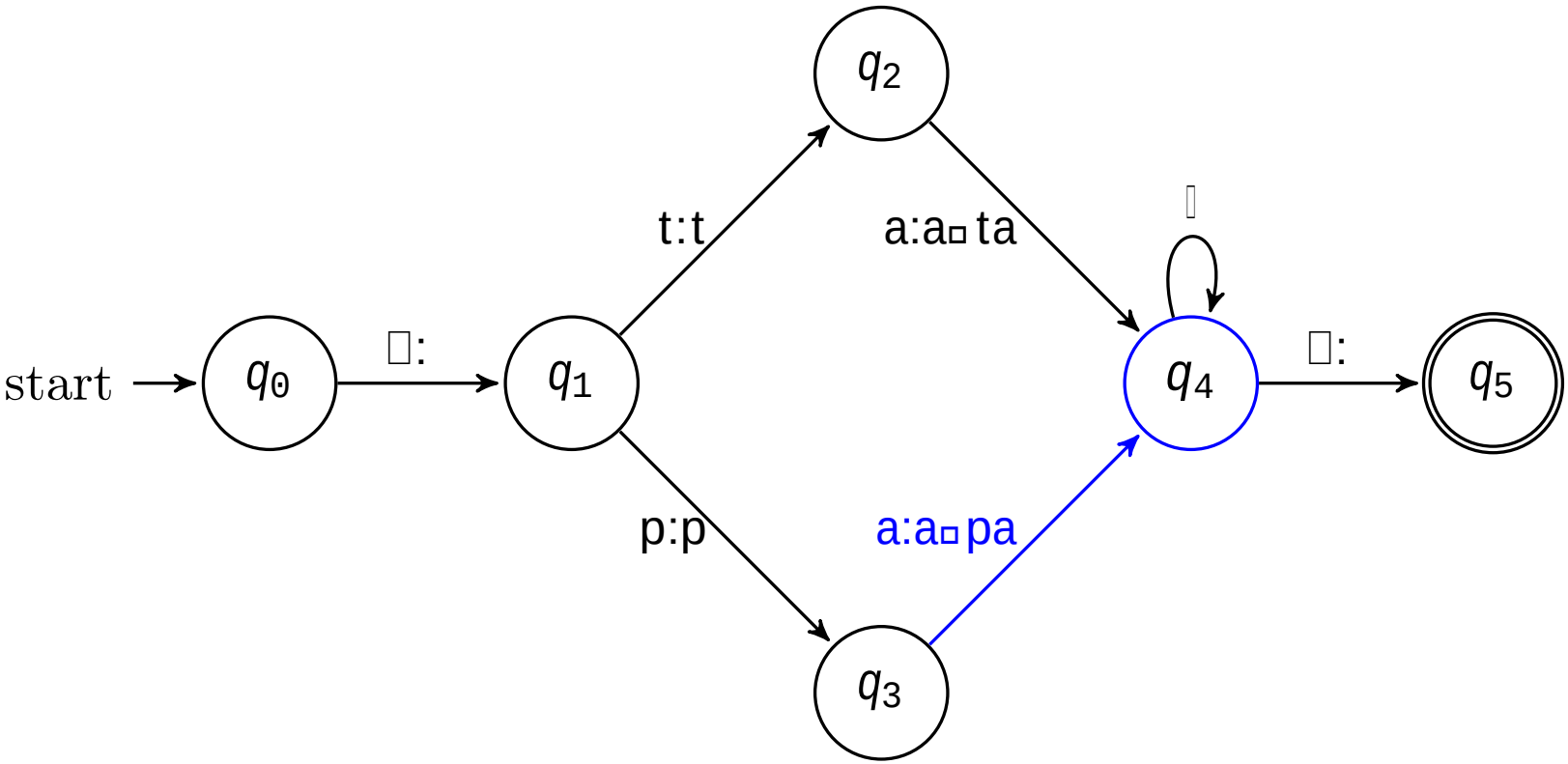
Input:	□	p	a	t	□
Output:		p			



Partial reduplication with 1-way FSTs

- Working example: pat→[pa□pat]

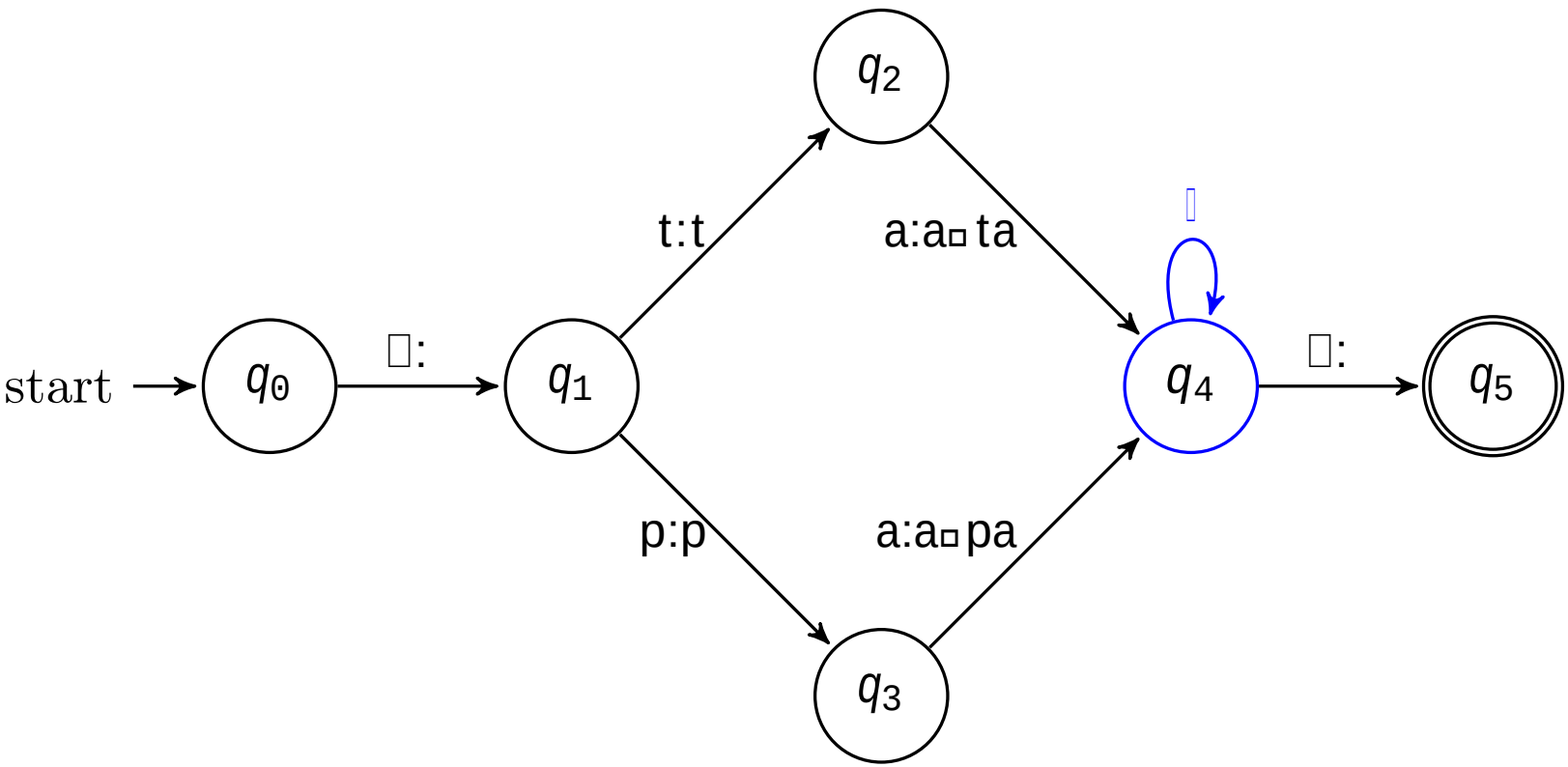
Input:	□	p	a	t	□
Output:		p	a□pa		



Partial reduplication with 1-way FSTs

- Working example: pat→[pa□pat]

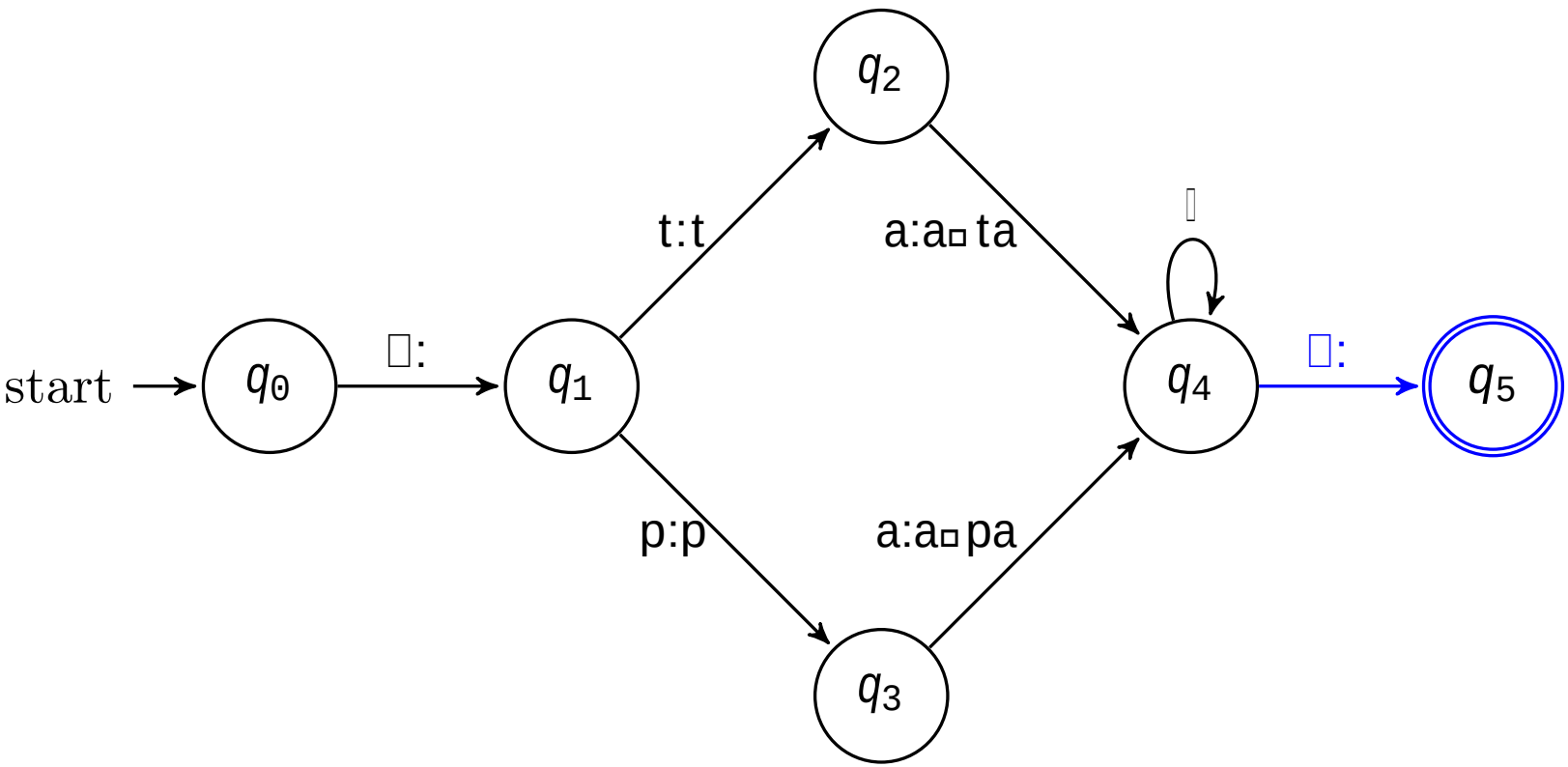
Input:	□	p	a	t	□
Output:		p	a□pa	t	



Partial reduplication with 1-way FSTs

- Working example: pat→[pa□pat]

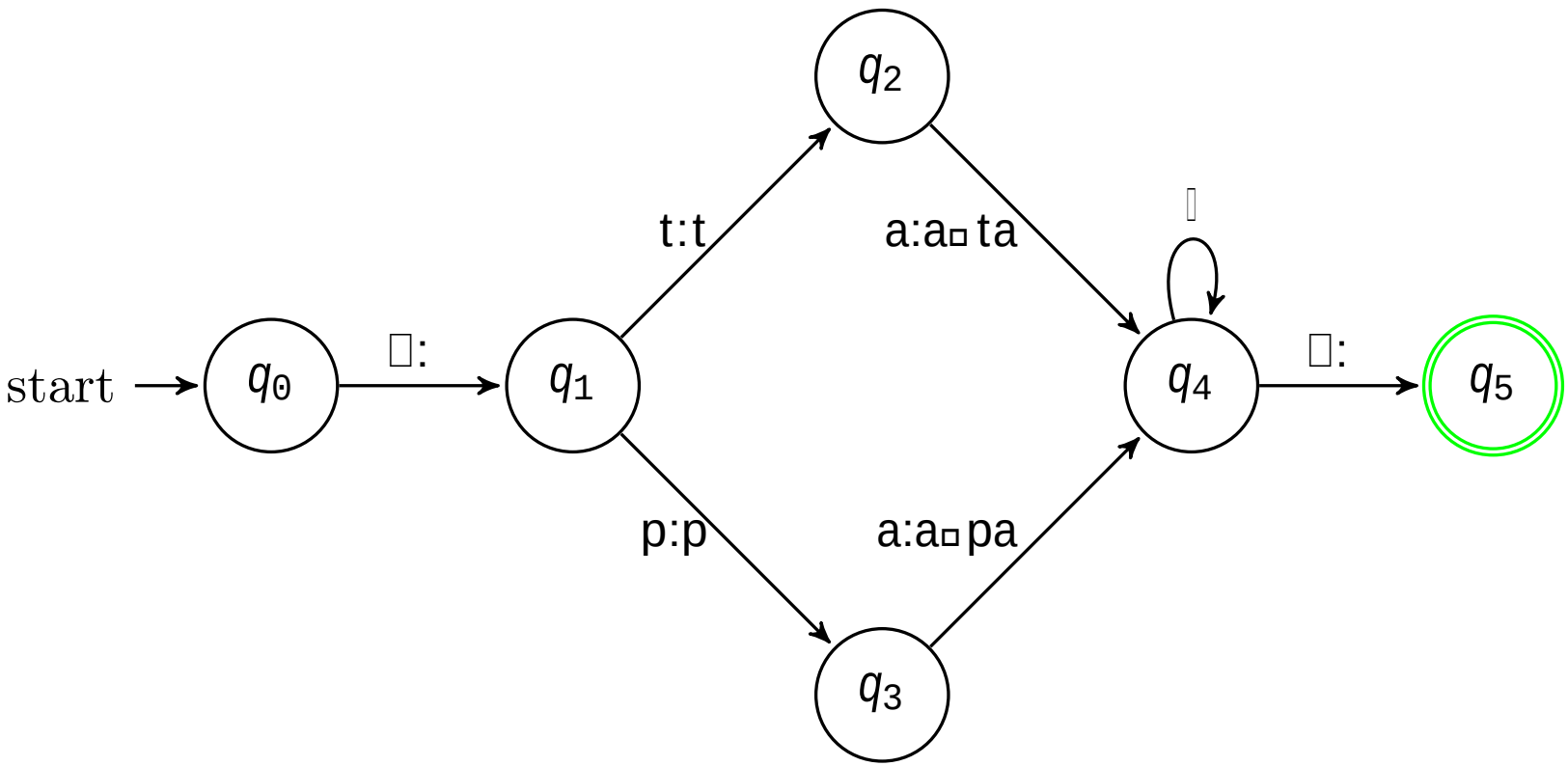
Input:	□	p	a	t	□
Output:		p	a□pa	t	



Partial reduplication with 1-way FSTs

- Working example: pat→[pa□pat]

Input:	□	p	a	t	□
Output:		p	a□pa	t	



1-way FST Limitations

- How does a 1-way FST handle reduplication?
  - → memorizes all possible reduplicants
- Many limitations:

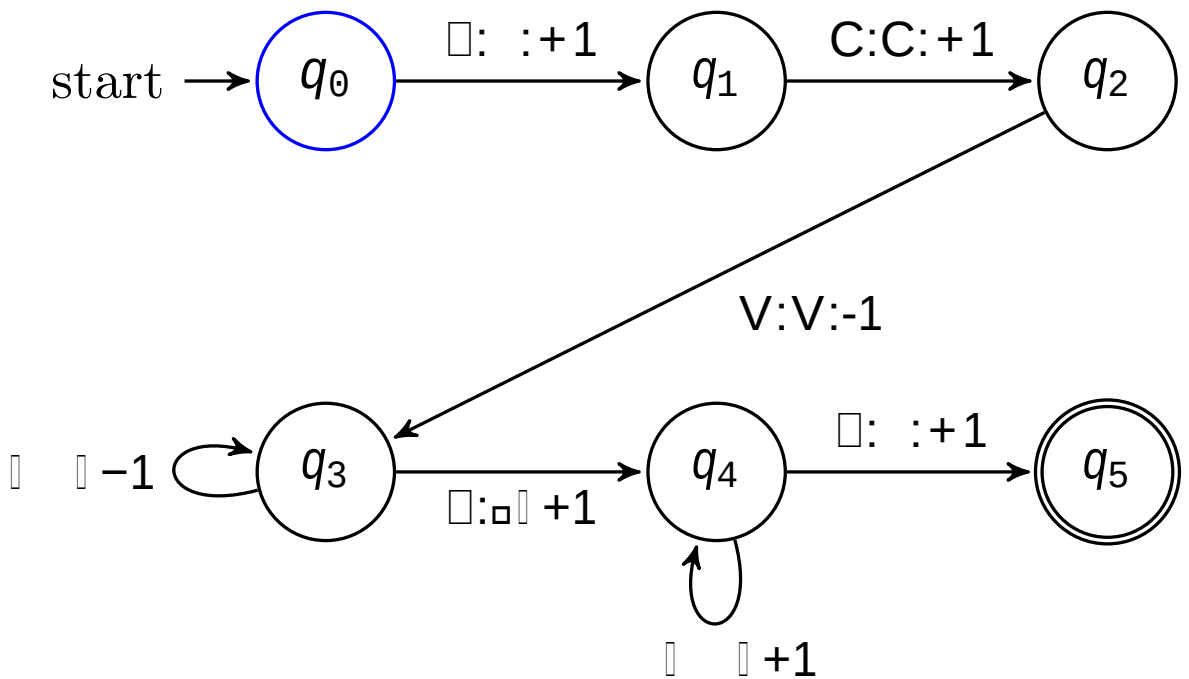
1. State explosion:
   - scaling problems as size of reduplicant and alphabet increases
   - unwieldy machines (Roark and Sproat, 2007:54)
2. Limited expressivity:
   - can do partial reduplication but not total reduplication
   - No bound on how big the copies are
3. Segment alignment:
   - Memorizes, doesn't 'copy'

Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

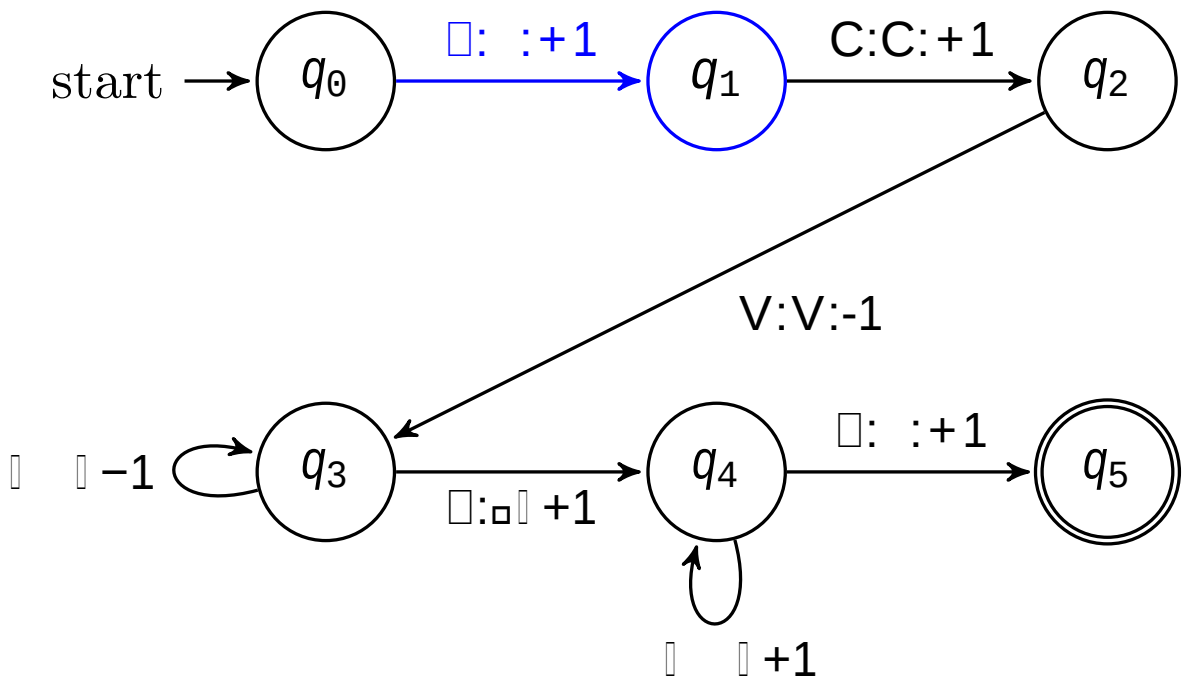
Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]
  Input: □ p a t □
  Output:



Partial reduplication with 2-way FSTs

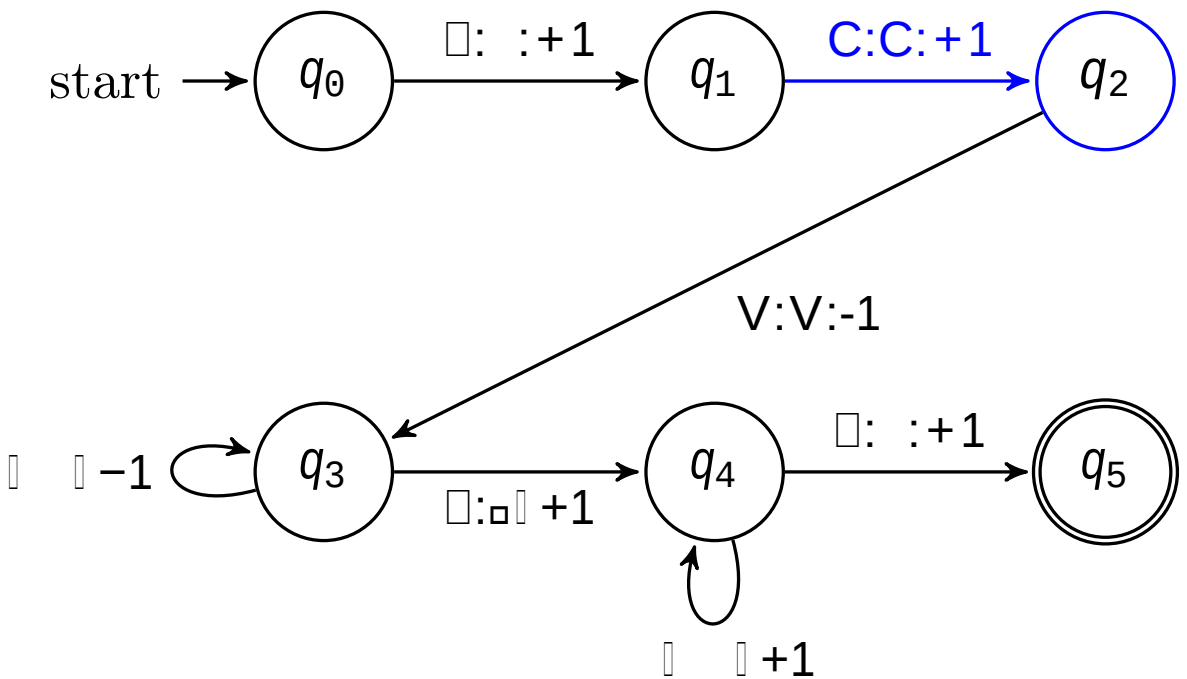
- Working example: pat→[pa□pat]
  Input: □ p a t □
  Output:



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

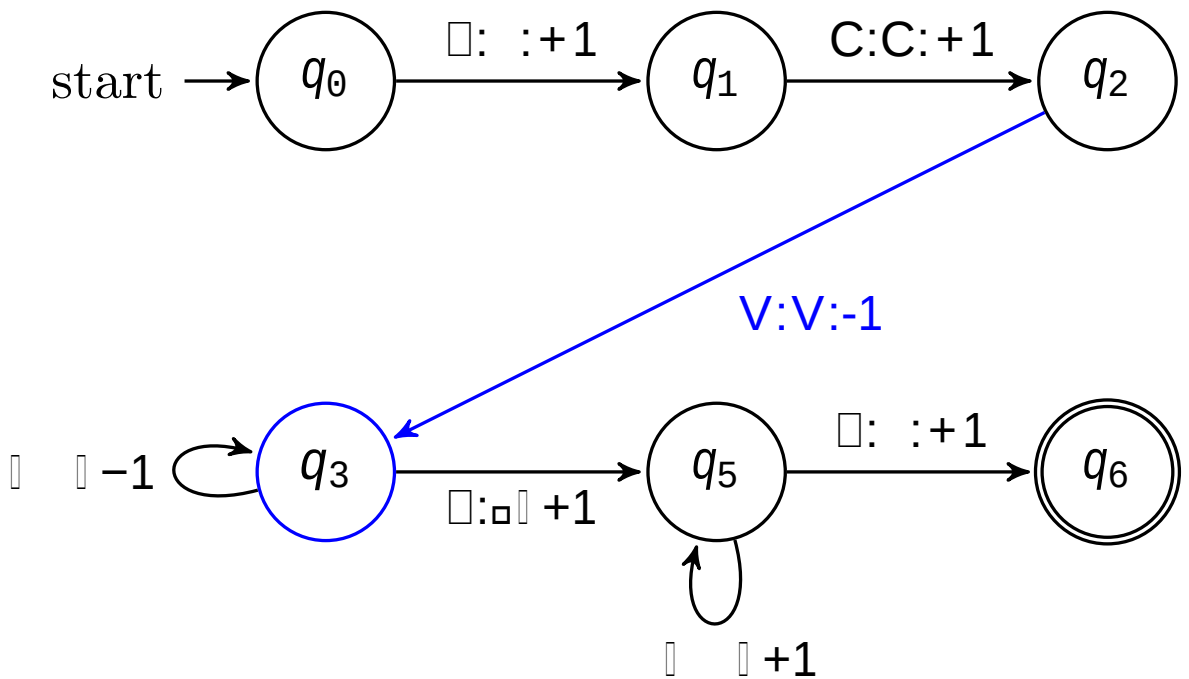
Input:	□	p	a	t	□
Output:		p			



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

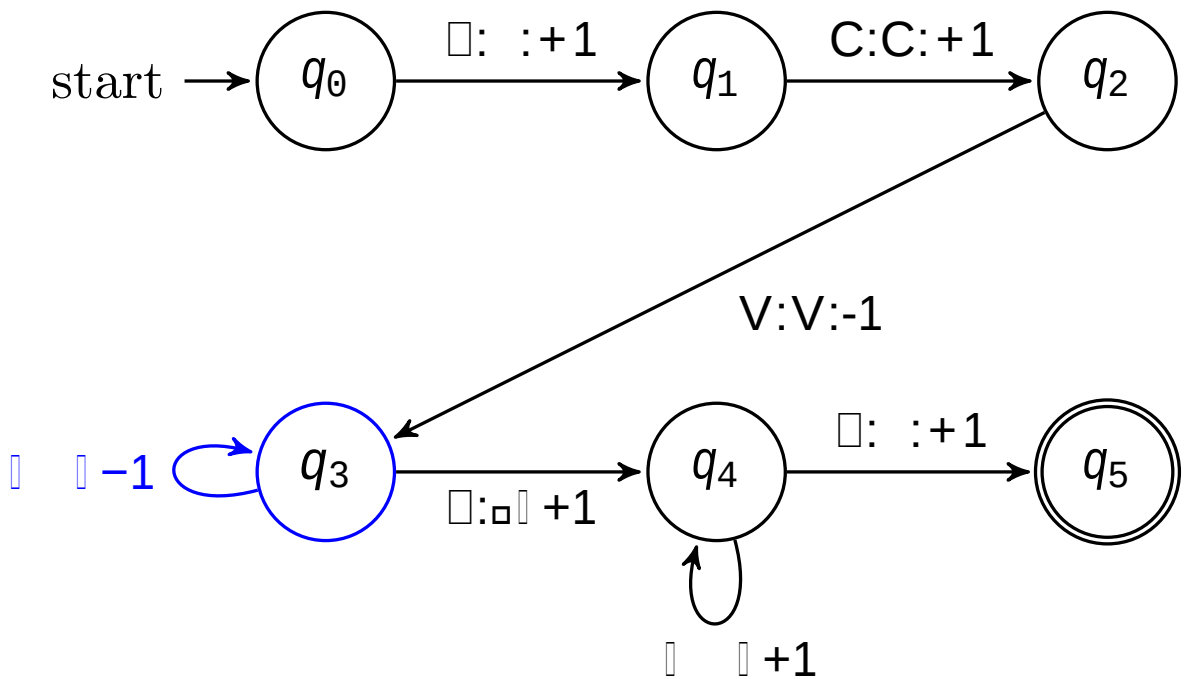
Input:	□	p	a	t	□
Output:		p	a		



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

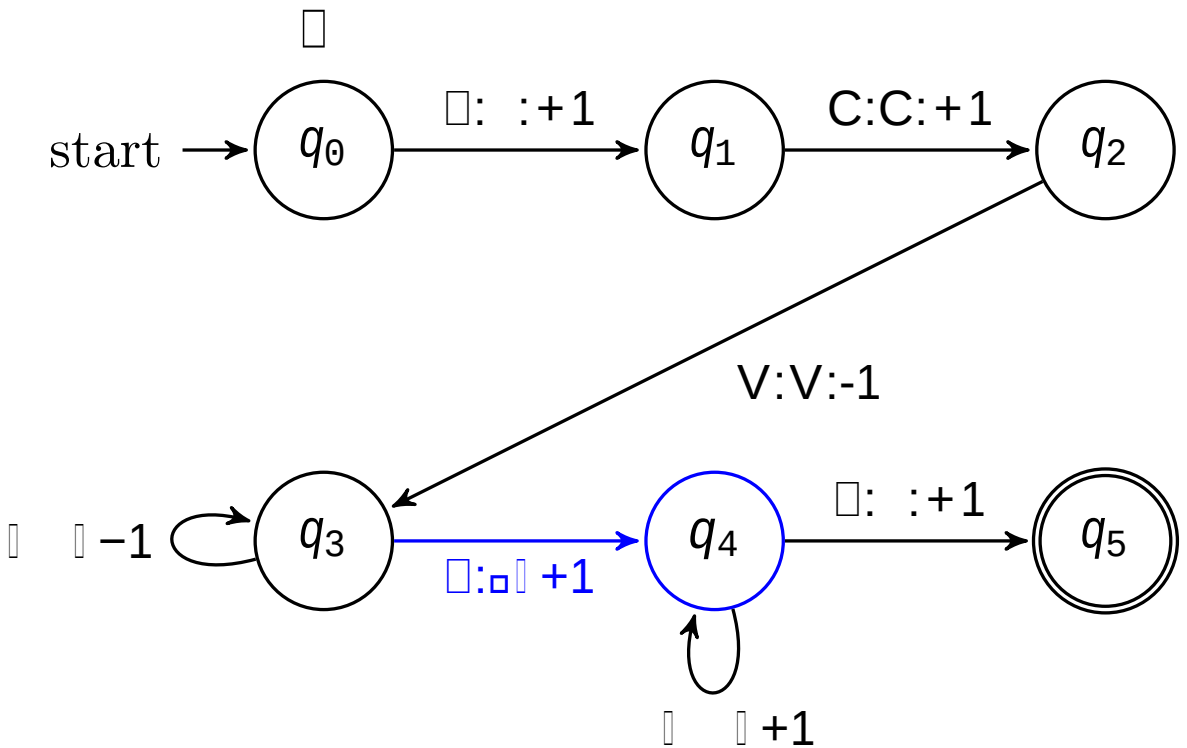
Input:	□	p	a	t	□
Output:		p	a		



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

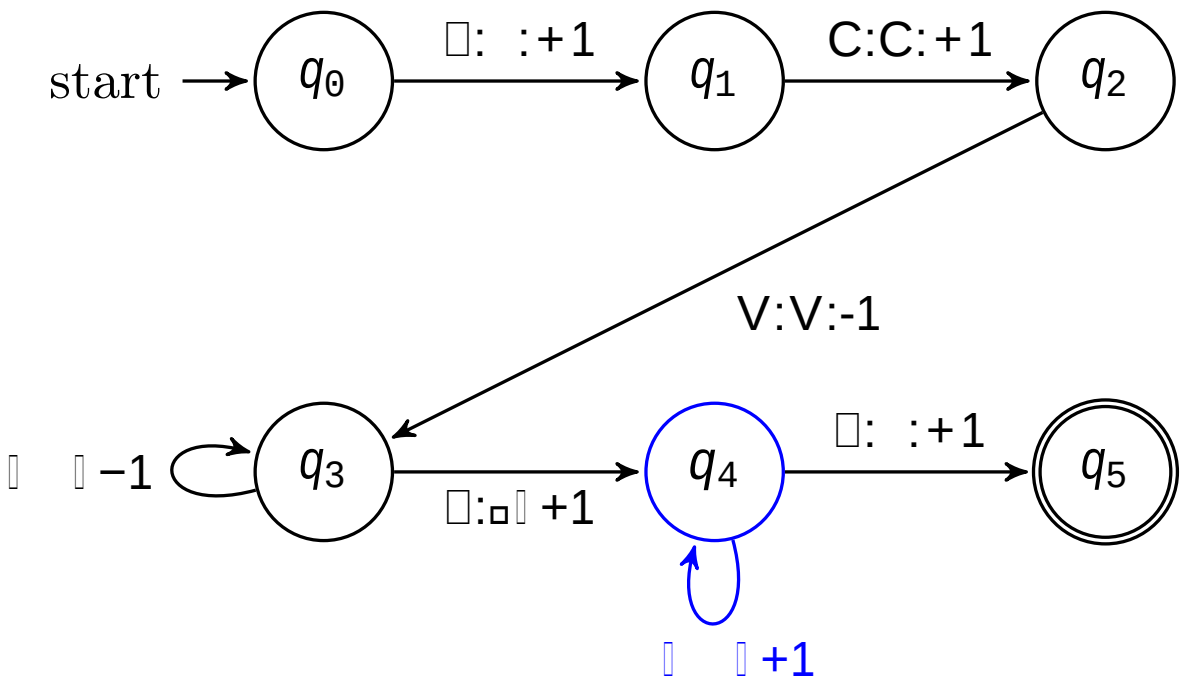
Input:	□	p	a	t	□
Output:		p	a		



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

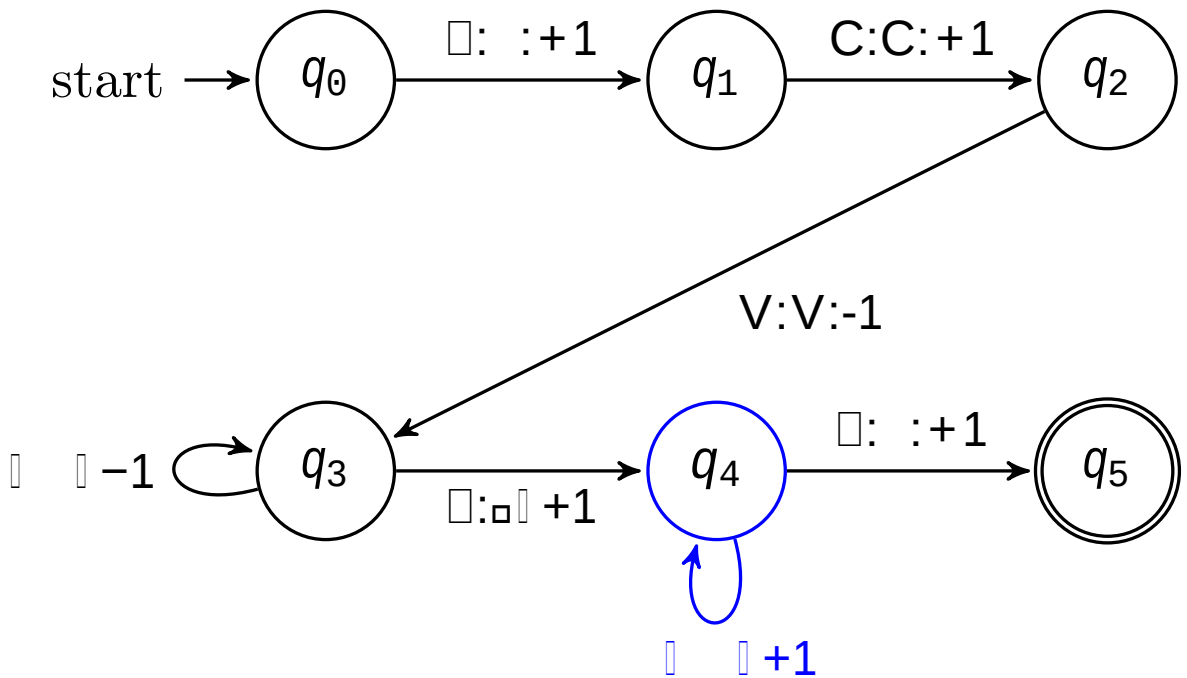
Input:	□	p	a	t	□
Output:		p	a		
	□	p			



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

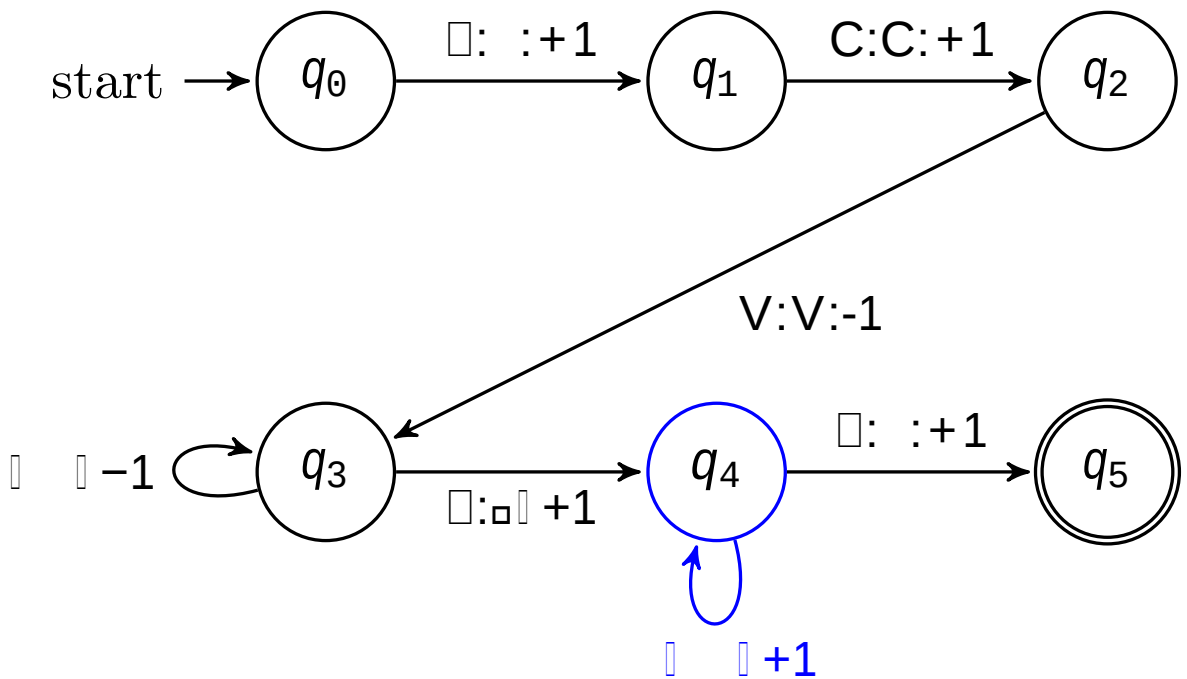
Input:	□	p	a	t	□
Output:		p	a		
	□	p	a		



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

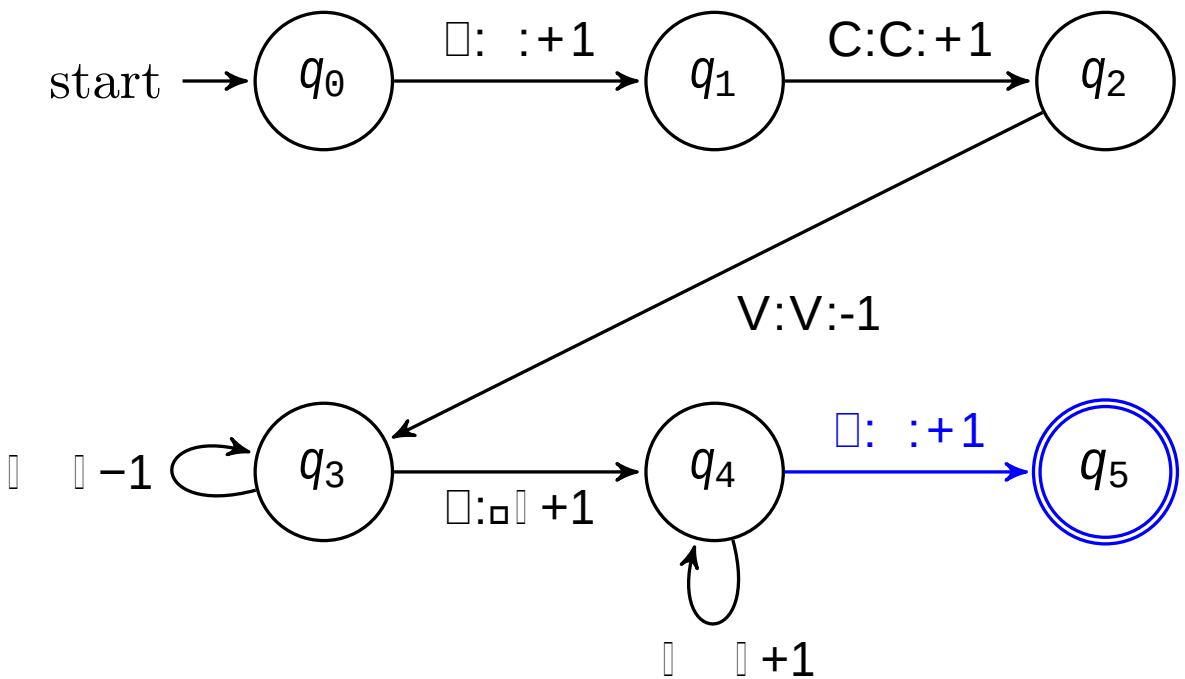
Input:	□	p	a	t	□
Output:		p	a		
	□	p	a	t	



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

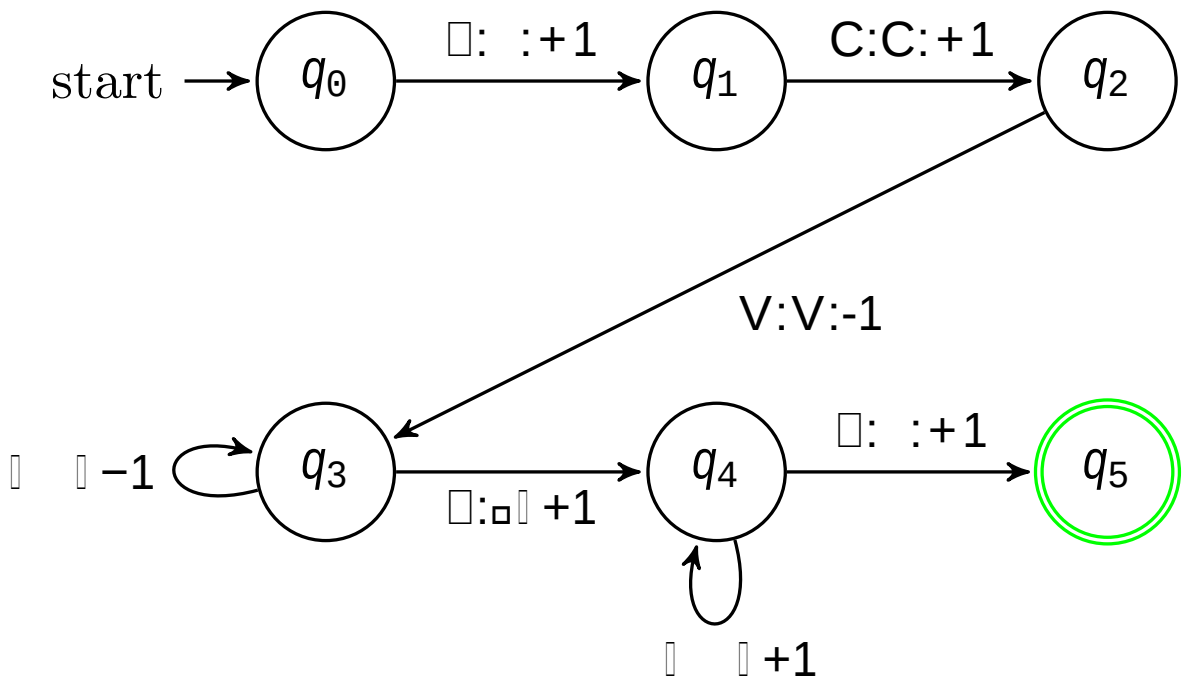
Input:	□	p	a	t	□
Output:		p	a		
	□	p	a	t	



Partial reduplication with 2-way FSTs

- Working example: pat→[pa□pat]

Input:	□	p	a	t	□
Output:		p	a		
	□	p	a	t	



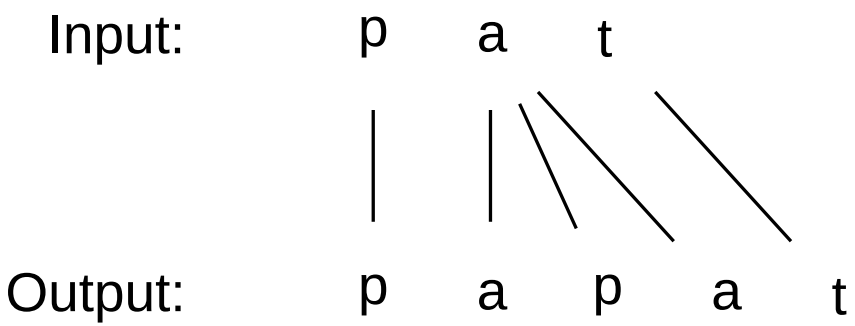
Reduplication with 2-way FSTs

- How does 2-way FST handle reduplication?
  - look back at the input to generate copies
- Increased expressivity, removes limitations...

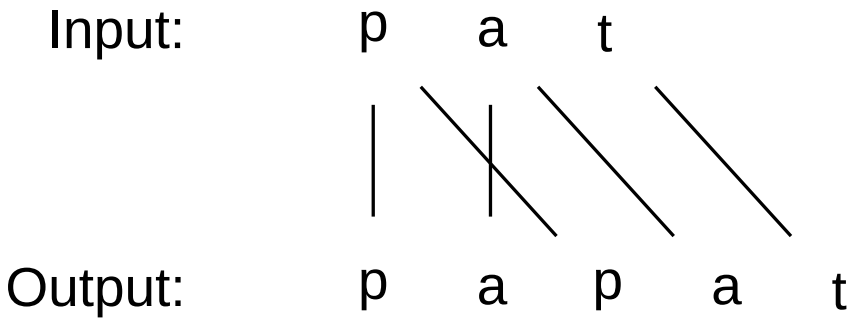
1. Compact :
   - no state explosion
2. Expressive :
   - can do partial and total reduplication
3. Segment alignment :
   - Output segments are aligned with the `right' input segments
   - Formally, look at origin semantics of how input-output segments align (Boja«czyk, 2014)

Segment alignment with FSTs

- Origin information : origin of output symbols in the input
- 1-way FSTs remember what to repeat, they don't actively copy



- But linguistic theory says copy like a 2-way FST!



Learning Reduplication

Reduplication is provably learnable in polynomial time and data (Chandlee et al., 2015; Dolatian and Heinz, 2018)

RNNs with segmental inputs cannot be trained as reduplication acceptors (Gasser, 1993; Marcus et al., 1999)

- Y Recognizing reduplication requires the comparison of static subsequences - di cult for an RNN to store

Encoder-Decoders learn reduplication with a xed-size reduplicant in a small toy language (Prickett et al., 2018)

- Y Generalizable to novel segments and sequences
- Y Generalization to novel lengths not tested, computable by 1-way FST that uses featural representations

Recurrence

- Recurrence relation: The function relating hidden states in the encoder and decoder RNNs - a ects practical expressivity of network
- Two types of recurrence tested:
  - sRNN - t^{th} state is a nonlinear function of the t^{th} input and state t 1 (Elman, 1990)
  - GRU - t^{th} state is a linear function of three functions (gates) of the t^{th} input and state t 1 (Cho et al., 2014)
- Saturating nonlinearities (tanh) - sRNNs and GRUs cannot count with nite precision (Weiss et al., 2018)
- LSTM is supra-regular, we are testing necessary properties of RNN and GRU, which are nite-state (Merrill, 2019)

Attention

- In standard ED, the encoded representation is the only link between the encoder and decoder
- Global attention allows the decoder to selectively pull information from hidden states of the encoder (Bahdanau et al., 2014)
- FLT Analog : 2-way FST has full access to the input by moving back and forth

Test data

- Input-output mappings generated with 2-way FSTs from RedTyp database[2]
  1. Initial-CV — tasgati ta tasgati
     Fixed-size reduplicant
  2. Initial two-syllable (C*VC*V) — tasgati tasga tasgati
     Onset maximizing, xed over vowels
  3. Total — tasgati tasgati tasgati
     Variably sized reduplicant
- 10,000 generated for each language, 70/30 train/test split
- Minimum string length 3 - maximum string length varied
- Alphabet of 10, 16, or 26 characters
- Boundary symbols ( ) are not present

[2] Dolatian and Heinz (2019); also available on GitHub

Experiment 1

- Interaction between reduplication type, recurrence, and attention
  - Total and partial (two-syllable) reduplication
  - sRNN and GRU with and without attention
- Max string length: 9
- 10 symbols alphabet

Attention should improve function generalization across reduplication types and recurrence relations

Experiment 1

Experiment 2

- Y E ects of alphabet size and range of permitted string lengths
- Y CV reduplication only
- Y sRNN/GRU attention/non-attention 3 alphabet sizes 7 length ranges

Network generalization while learning a general reduplication function should be invariant to language composition

Experiment 2

Experiment 2

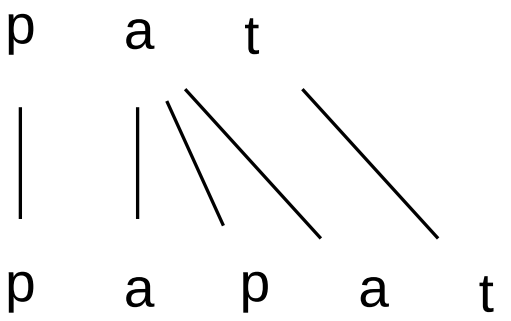
Discussion

- Networks with global attention learn and generalize all types of reduplication and seem robust to string length and alphabet size
- sRNNs without attention show slightly better generalization of partial reduplication than total reduplication
  - Confound with less attested reduplicant lengths or a bias preferring the regular pattern?
- GRUs perform better than sRNNs across all conditions
  - Without attention not robust to length/alphabet - likely learning heuristics that capture most data rather than a general function

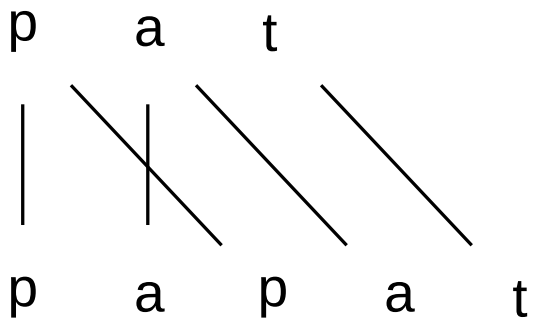
Networks that cannot see material in the input multiple times cannot learn generalizable reduplication

Attention and Origin Semantics

1-Way:



2-Way:



Summary

1. Why use reduplication functions?
   - properties de ne ne-grained subregular function classes
   - Allows us to test the generalization capacity of neural nets
2. Expressivity of attention
   - Attention is necessary and su cient for robustly learning and generalizing reduplication functions using Encoder-Decoders
3. FST approximations
   - Non-attention networks are limited to a single input pass, approximating 1-way FST
   - Attention networks can read the input again during decoding, approximating 2-way FST,
4. Attention weights and origin information
   - Evidence for approximation comes from attention weights
   - IO correspondence relations mirror origin semantics of 2-way FST
5. Next step : trying more copying and non-copying functions

Albro, D. M. (2005). Studies in Computational Optimality Theory, with Special Reference to the Phonological System of Malagasy. Ph. D. thesis, University of California, Los Angeles, Los Angeles.

Avcu, E., C. Shibata, and J. Heinz (2017). Subregular complexity and deep learning. In S. Dobnik and S. Lappin (Eds.), CLASP Papers in Computational Linguistics: Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017), Gothenburg, 12 13 June, pp. 20 33.

Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Beesley, K. and L. Karttunen (2003). Finite-state morphology: Xerox tools and techniques. Stanford, CA: CSLI Publications.

Boja«czyk, M. (2014). Transducers with origin information. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias (Eds.), Automata, Languages, and Programming, Berlin, Heidelberg, pp. 26 37. Springer.

Chandlee, J., R. Eyraud, and J. Heinz (2015, July). Output strictly local functions. In Proceedingsof the 14th Meeting on the Mathematics of Language(MoL 2015), Chicago, USA, pp. 112 125.

Cho, K., B. Van Merriënboer, D. Bahdanau, and Y. Bengio (2014). On the properties of neural machine translation: Encoder-decoder approaches.arXiv preprint arXiv:1409.1259.

Crysmann, B. (2017). Reduplication in a computational HPSG of Hausa. Morphology 27(4), 527 561.

Dolatian, H. and J. Heinz (2018, September). Learning reduplication with 2-way nite-state transducers. In O. Unold, W. Dyrka, , and W. Wieczorek (Eds.), Proceedingsof Machine Learning Research: International Conferenceon Grammatical Inference, Volume 93 of Proceedingsof Machine Learning Research, Wroclaw, Poland, pp. 67 80.

Dolatian, H. and J. Heinz (2019). Redtyp: A database of reduplication with computational models. In Proceedingsof the Society for Computation in Linguistics, Volume 2. Article 3.

Elman, J. L. (1990). Finding structure in time. Cognitive science14(2), 179 211.

Gasser, M. (1993). Learning words in time: Towards a modular connectionist account of the acquisition of receptive morphology. Indiana University, Department of Computer Science.

Heinz, J. and R. Lai (2013). Vowel harmony and subsequentiality. In A. Kornai and M. Kuhlmann (Eds.), Proceedingsof the 13th Meeting on the Mathematics of Language(MoL 13), So a, Bulgaria, pp. 52 63. Association for Computational Linguistics.

Hulden, M. (2009). Finite-state machine construction methods and algorithms for phonology and morphology. Ph. D. thesis, The University of Arizona, Tucson, AZ.

Marcus, G. F., S. Vijayan, S. B. Rao, and P. M. Vishton (1999). Rule learning by seven-month-old infants. Science283(5398), 77 80.

Merrill, W. (2019). Sequential neural networks as automata. In Proceedingsof the Deep Learning and Formal Languagesworkshop at ACL 2019.

Moravcsik, E. (1978). Reduplicative constructions. In J. Greenberg (Ed.), Universals of Human Language, Volume 1, pp. 297 334. Stanford, California: Stanford University Press.

Prickett, B., A. Traylor, and J. Pater (2018). Seq2seq models with dropout can learn generalizable reduplication. In Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology, pp. 93 100.

Rabusseau, G., T. Li, and D. Precup (2019). Connecting weighted automata and recurrent neural networks through spectral learning. In AISTATS.

Roark, B. and R. Sproat (2007). Computational Approaches to Morphology and Syntax. Oxford: Oxford University Press.

Rubino, C. (2013). Reduplication. Leipzig: Max Planck Institute for Evolutionary Anthropology.

Savitch, W. J. (1989). A formal model for context-free languages augmented with reduplication. Computational Linguistics 15(4), 250 261.

Siegelmann, H. T. (2012). Neural networks and analog computation: beyond the Turing limit. Springer Science & Business Media.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. CoRR abs/1409.3215.

Walther, M. (2000). Finite-state reduplication in one-level prosodic morphology. In Proceedingsof the 1st North American chapter of the Association for Computational Linguistics conference, NAACL 2000, Seattle, Washington, pp. 296 302. Association for Computational Linguistics.

Weiss, G., Y. Goldberg, and E. Yahav (2018). On the practical computational power of nite precision rnns for language recognition. In Proceedingsof the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 740 745.

Guide to appendix

- Reduplication across FSTs and RNNs [25]
- Harmony Extensions [26]
- Finite-State Automata & Representation Learning [27]
- Learning Reduplication [28]
- Problems with 1-way FSTs for Total Reduplication [29]
- Total reduplication with 2-way FSTs [31]

Reduplication across FSTs and RNNs

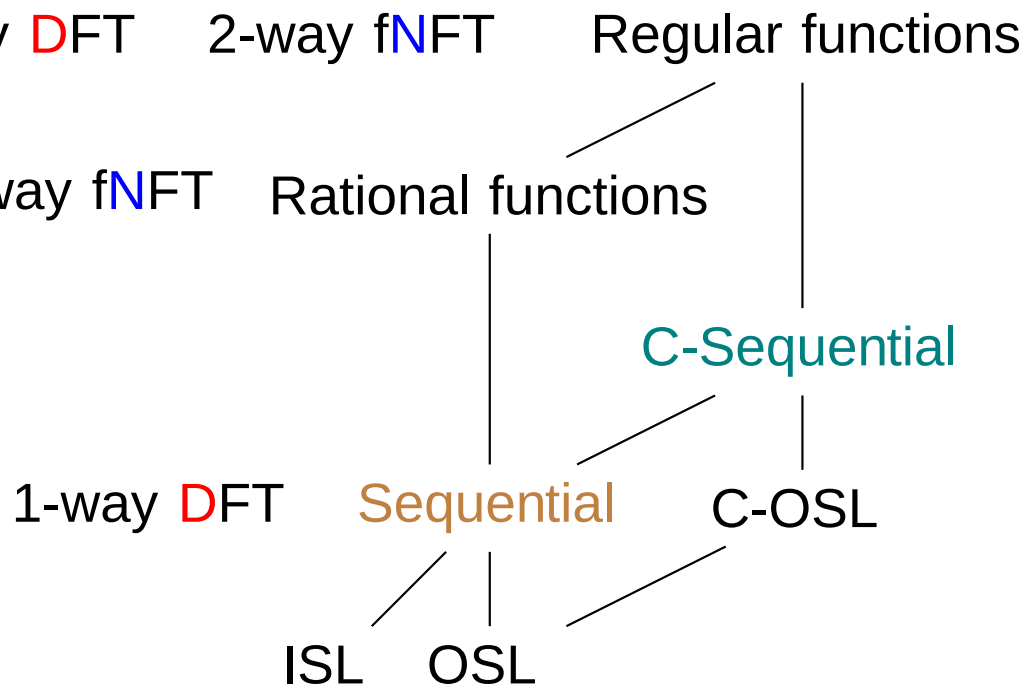
- 1-way and 2-way FSTs compute reduplicative functions di erently

	1-way	2-way
Strategy ?		
How does it reduplicate?	Memorize	Look back
Scaling?		
Is there state explosion	3 /	7 ,
Expressive ?		
Can it do total reduplication?	7 /	3 ,
Alignment ?		
Does origin information match theory?	7 /	3 ,

- Strategy creates all additional properties
- Link to RNNs :
  - attention-less EDs compute like 1-way FSTs!
  - attention-based EDs compute like 2-way FSTs

Next: Attention, 2-way, and Determinism

The subregular hierarchy is more subtle



- Does attention enable non-regularity? Non-determinism?
  - What about $w \to w^3$, $w \to ww^r$, $w \to w^w$, ...
- Idea: Use Harmony processes (Heinz and Lai, 2013)
  - harmony spans subregular hierarchy
  - unattested non-regular harmony (ex. Majority Rules)

Finite-State Automata & Representation Learning

- An FSA induces a mapping ‡ R
- The mapping is compositional
- The output f_A ˆx• ˆx•;! is linear in ˆx•

Pic credit: Guillaume Rabusseau

Learning Reduplication

- Reduplication is provably learnable in polynomial time and data (Chandlee et al., 2015; Dolatian and Heinz, 2018)
- RNNs with segmental inputs cannot be trained as reduplication acceptors (Gasser, 1993; Marcus et al., 1999)
  - Recognizing reduplication requires the comparison of static subsequences - difficult for an RNN to store
- Encoder-Decoders learn reduplication with a fixed-size reduplicant in a small toy language (Prickett et al., 2018)
  - Generalizable to novel segments and sequences
  - Generalization to novel lengths not tested, computable by 1-way FST that uses featural representations

Problems with 1-way FSTs for Total

- 1-way FSTs can do Partial Red inelegantly
- Total reduplication cannot be modeled at all.
- Why?
  - copied portion has unbounded size
  - 1-way FST can't do that!
  - needs an in nite # of states

Problems with 1-way FSTs for Total

- Total reduplication cannot be modeled at all.
- Can you approximate?
  - some nite-state approximations exist... [3]
  - But : they impose un-linguistic restrictions (e.g. a nite bound on word size,...) so don't directly capture reduplication
- Give up on nite-state?
  - MCFGs, HPSG, pushdown accepters with queues[4]
  - But... those are recognizers not transducers

[3] Hulden (2009); Beesley and Karttunen (2003); Walther (2000)
[4] Albro (2005); Crysmann (2017); Savitch (1989)

Total reduplication with 2-way FSTs

- Total reduplication copies an unbounded size

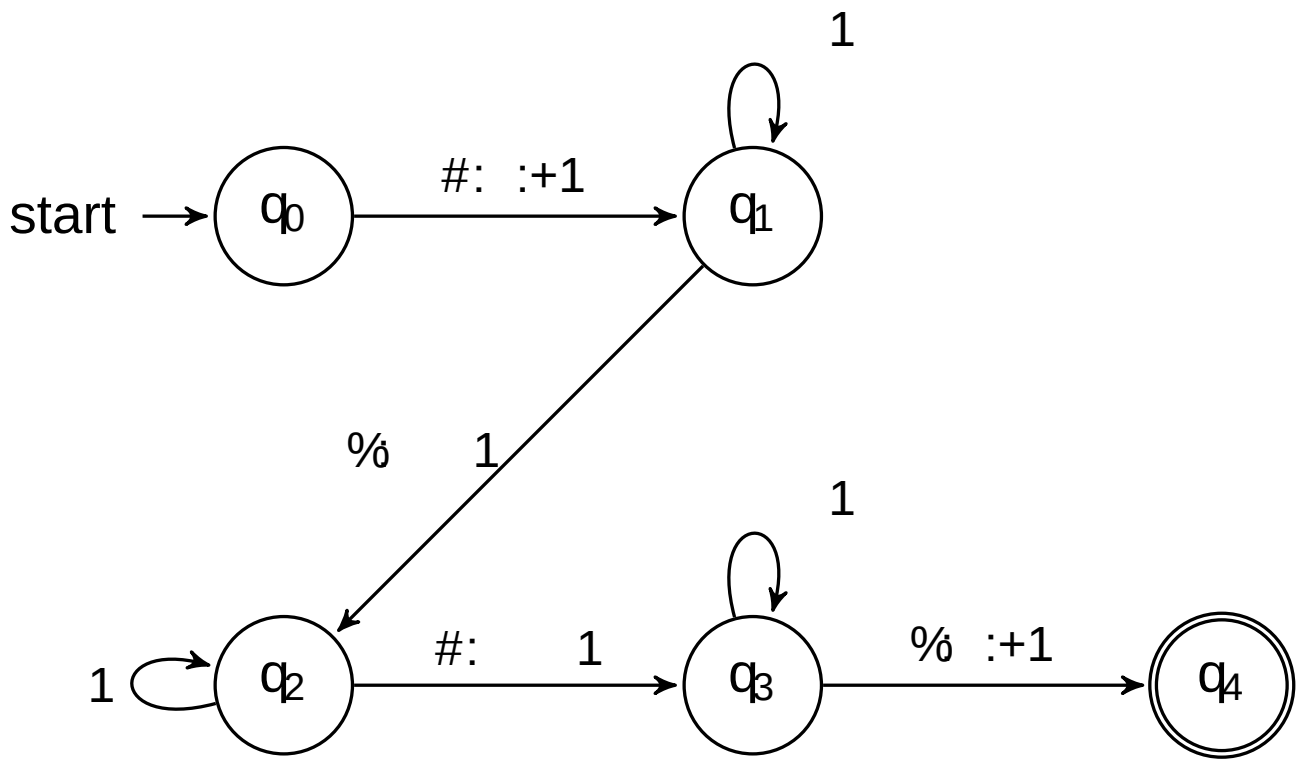
(3) wanita wanita wanita `woman' `women' (Indo.)

Total reduplication with 2-way FSTs

- Total reduplication copies an unbounded size

  (4) wanita wanita wanita `woman' `women' (Indo.)

- This 2-way FST reads the input left to right (+1), goes back (-1), and reads the input again (+1)



Total reduplication with 2-way FSTs

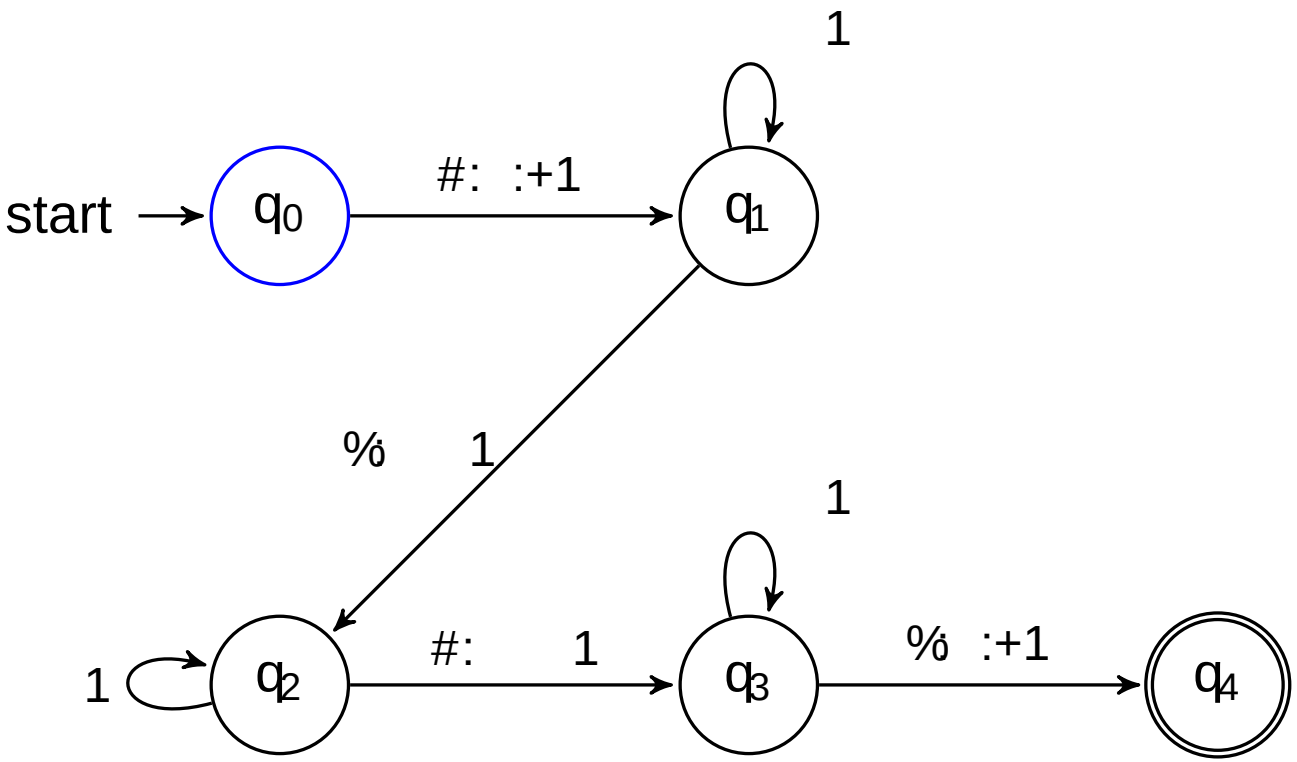
- Indonesian example: wanita wanita wanita
- Working example: bye ?

Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

Input: # b y e %

Output:

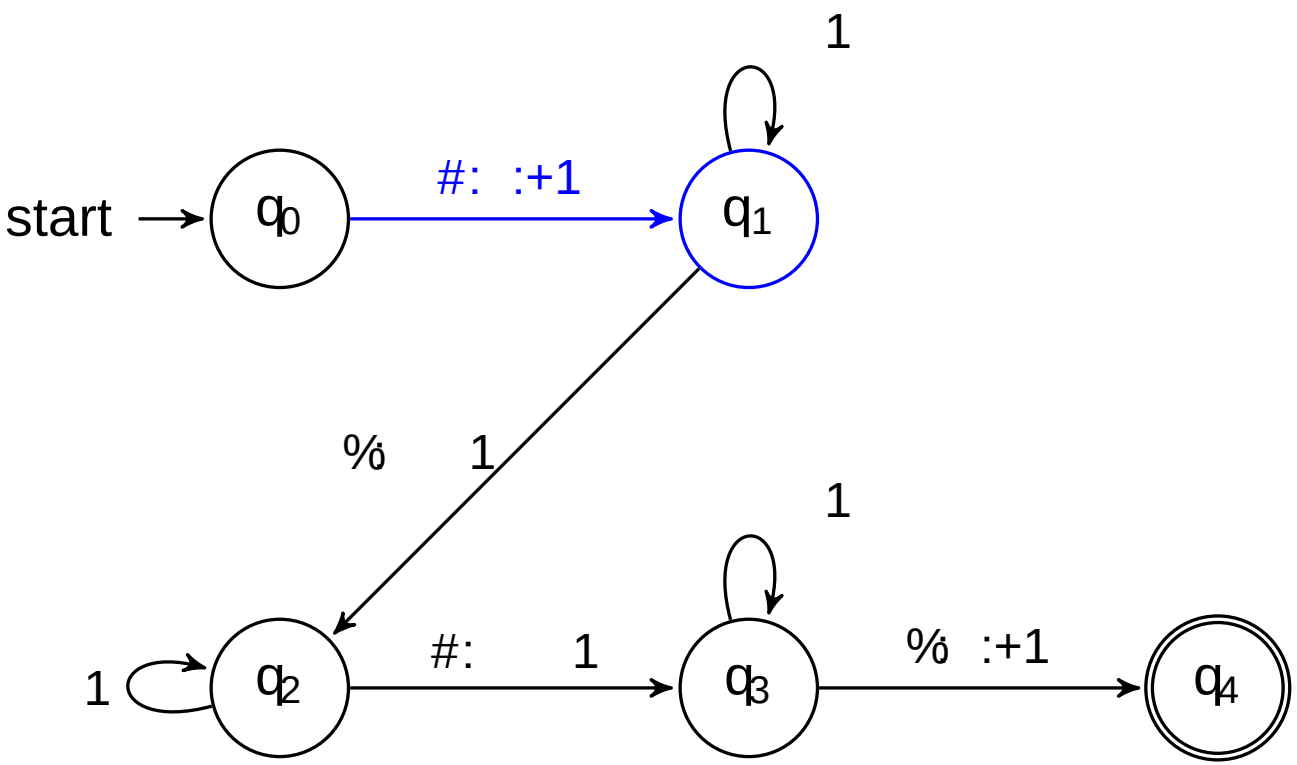


Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

Input: # b y e %

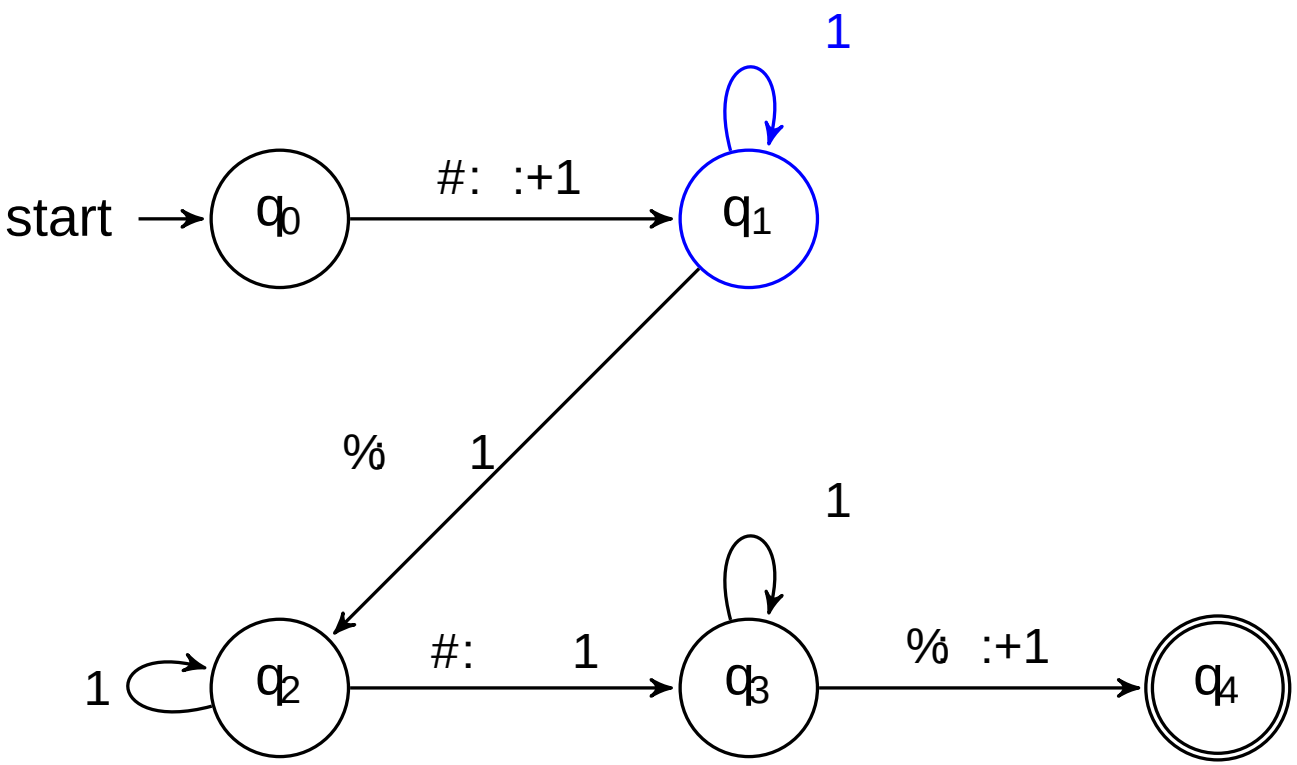
Output:



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

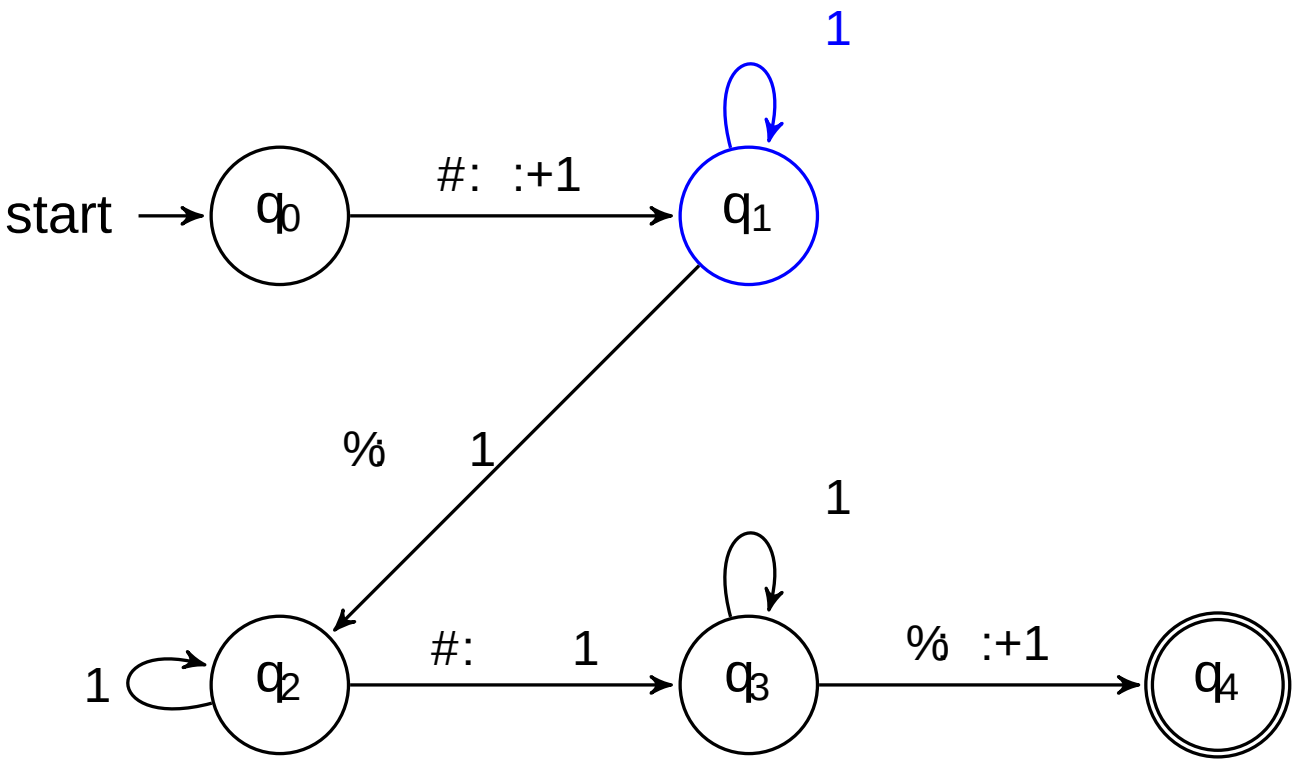
Input:	#	b	y	e	%
Output:		b			



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

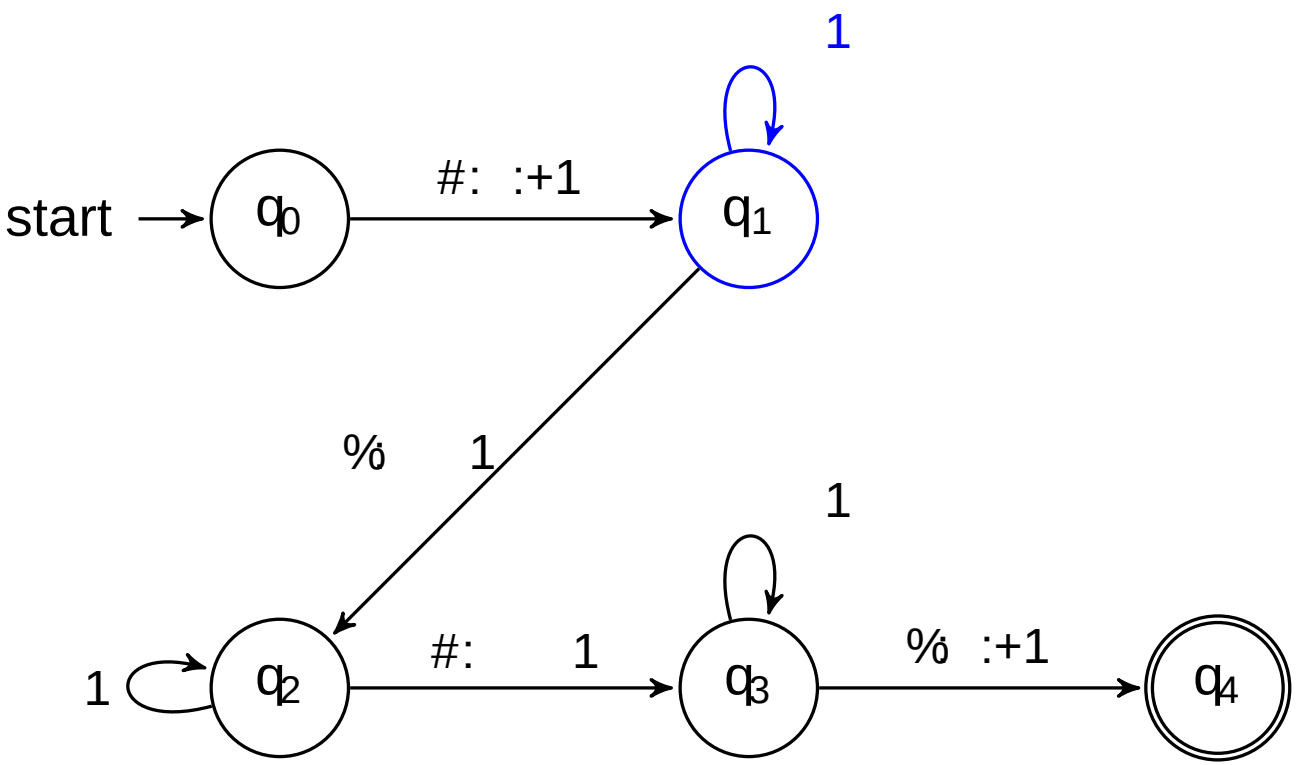
Input:	#	b	y	e	%
Output:		b	y		



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

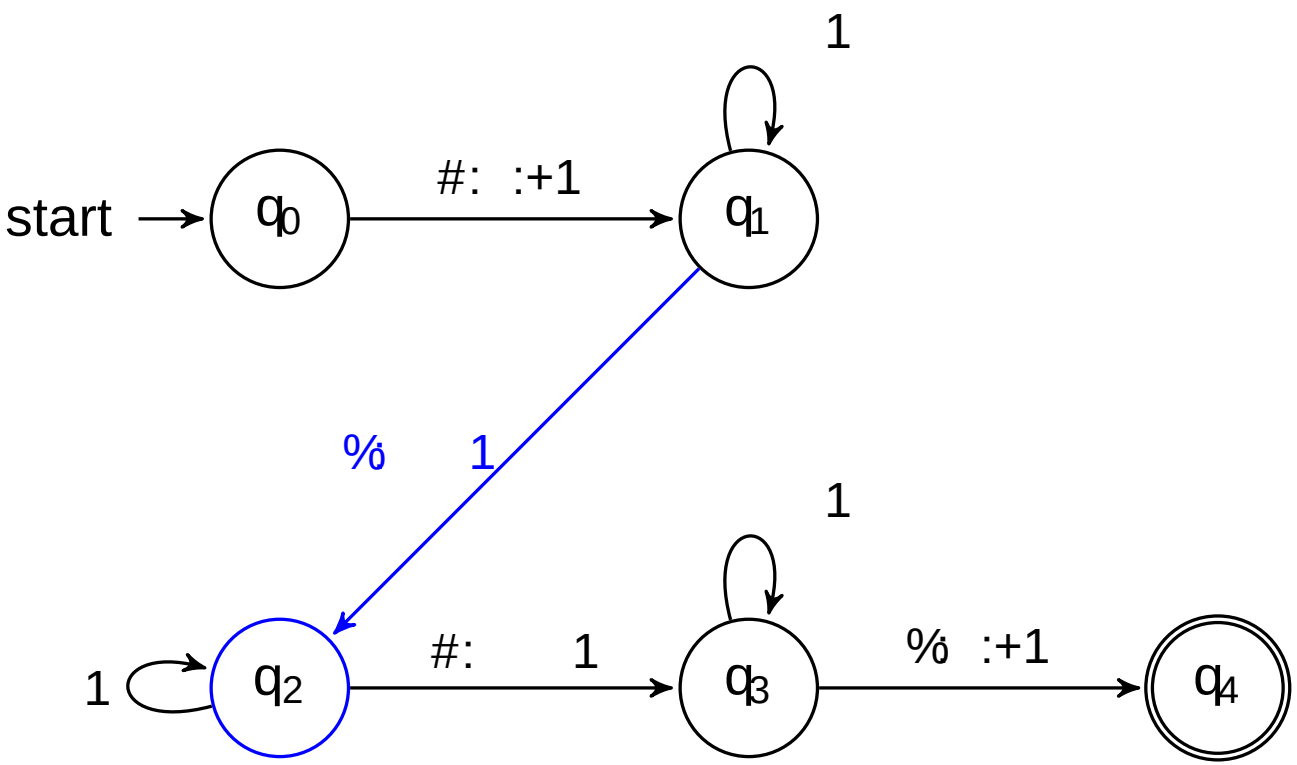
Input:	#	b	y	e	%
Output:		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

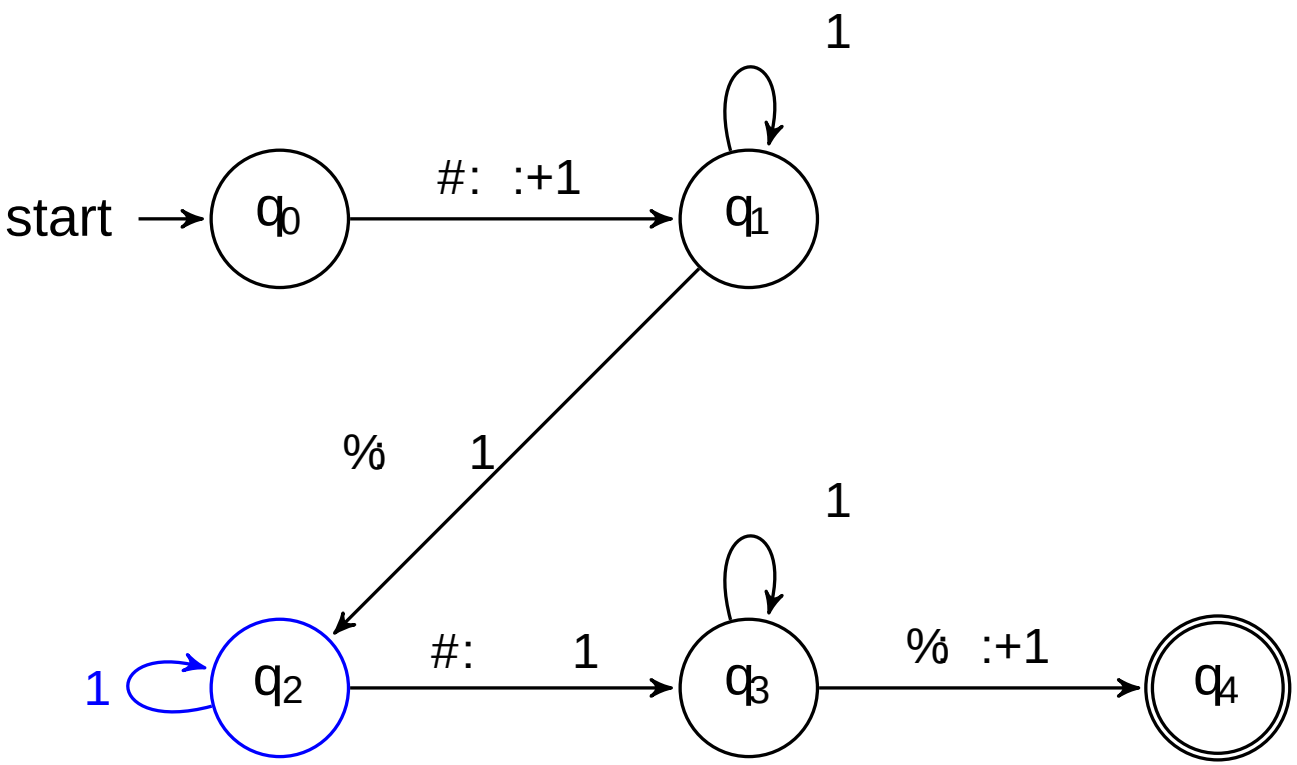
Input:	#	b	y	e	%
Output:		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

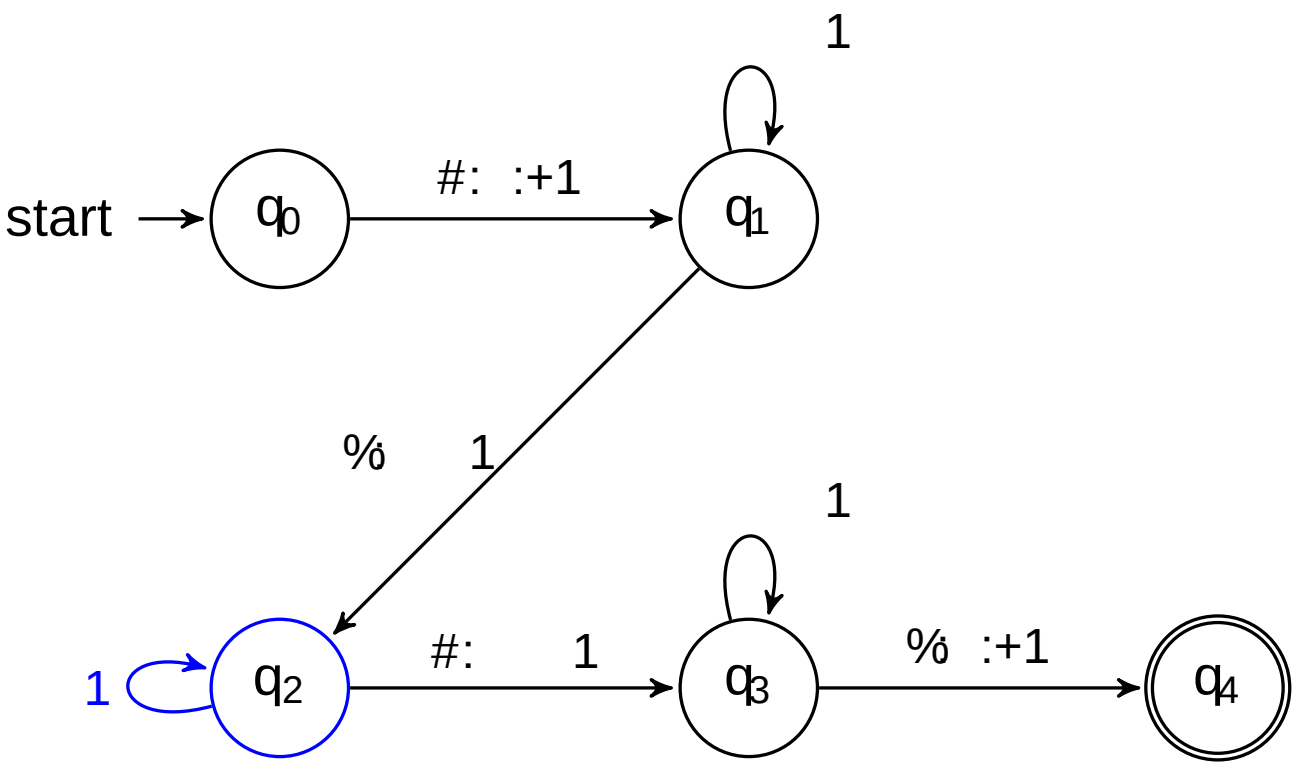
Input:	#	b	y	e	%
Output:		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

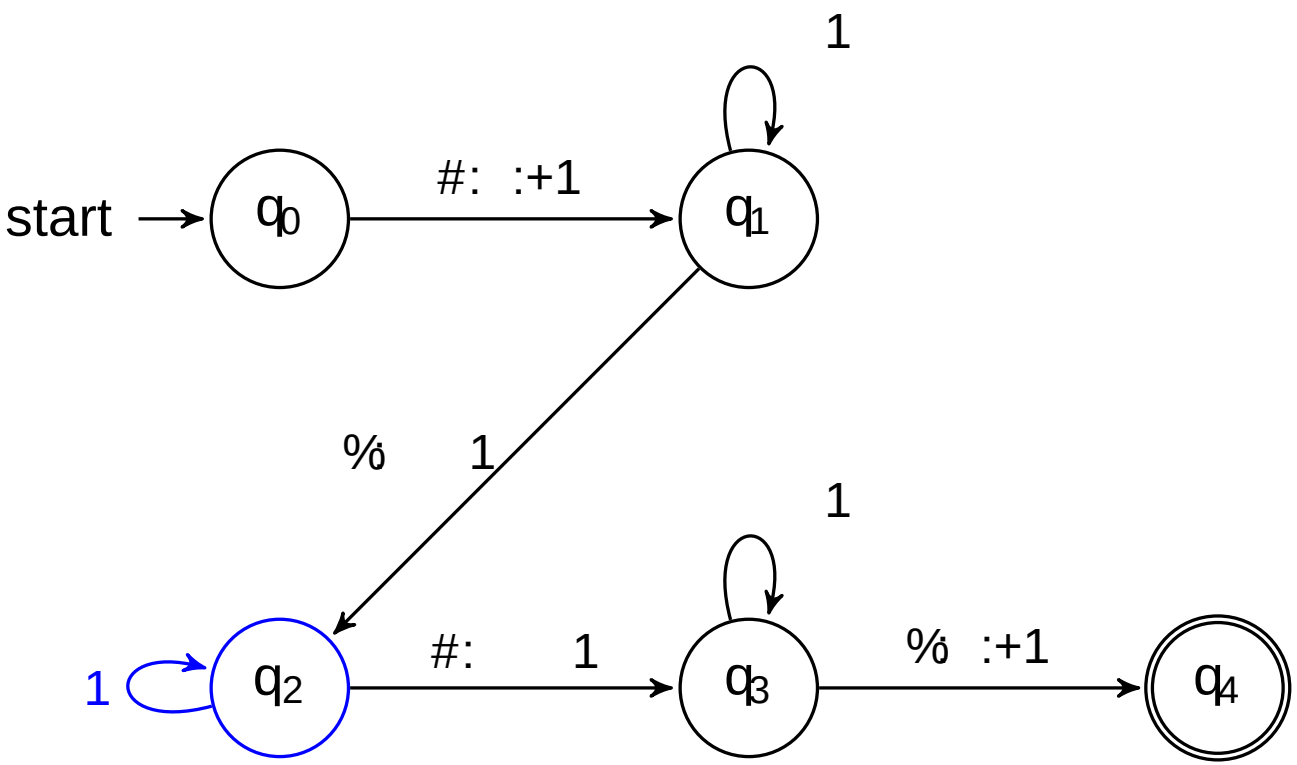
Input:	#	b	y	e	%
Output:		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

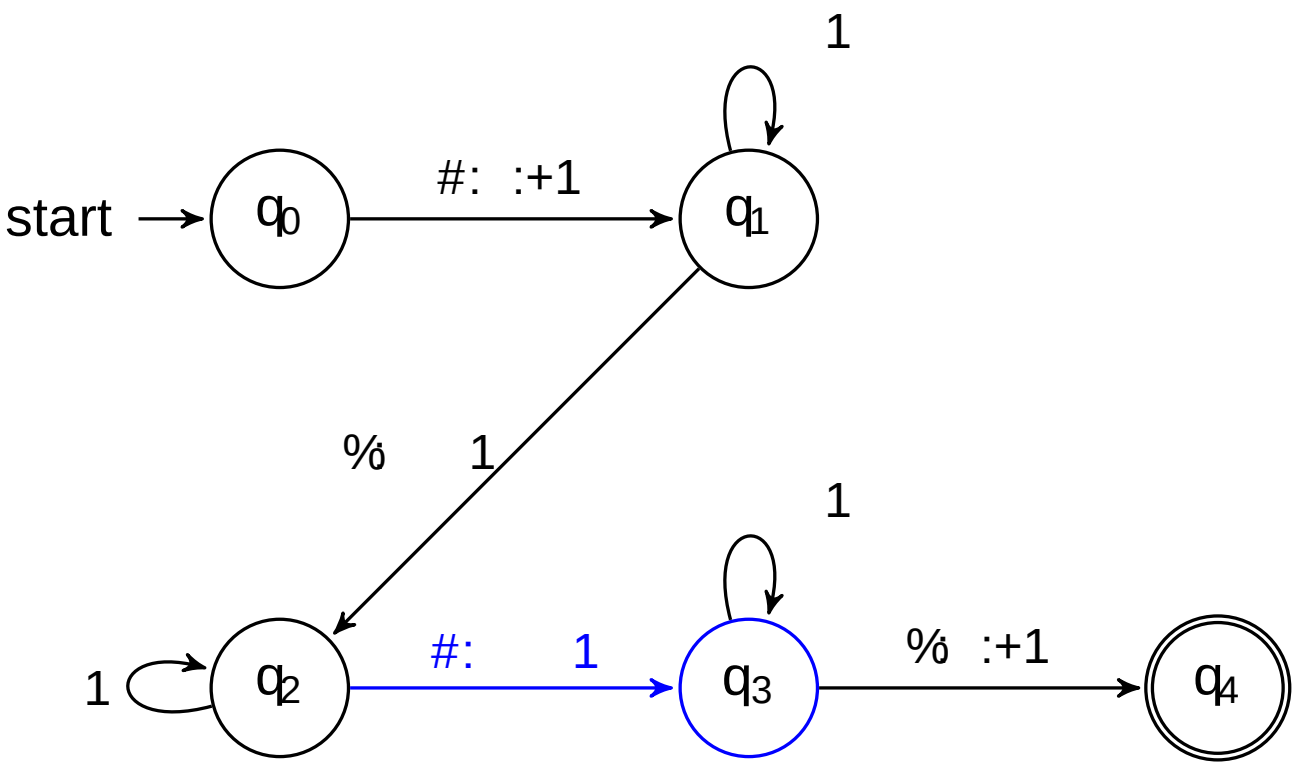
Input:	#	b	y	e	%
Output:		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

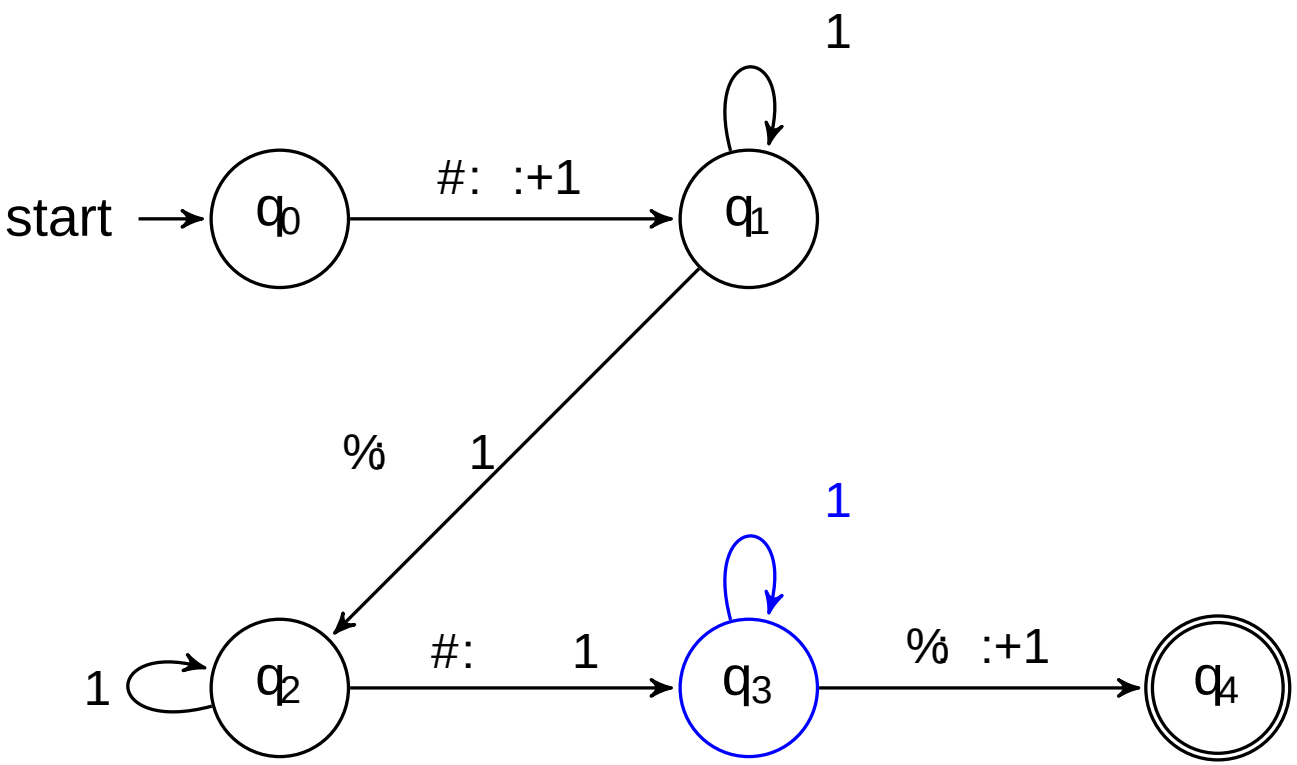
Input:	#	b	y	e	%
Output:		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

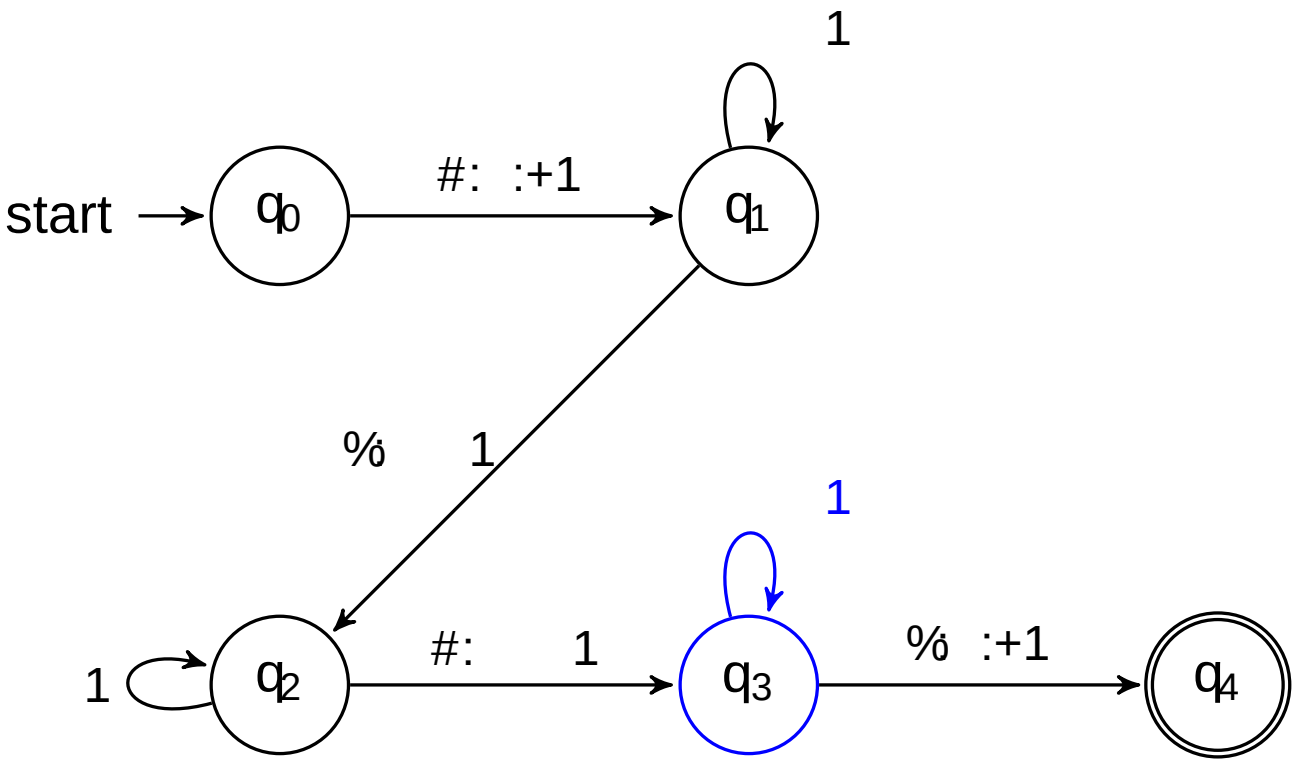
Input:	#	b	y	e	%
Output:		b	y	e	
		b			



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

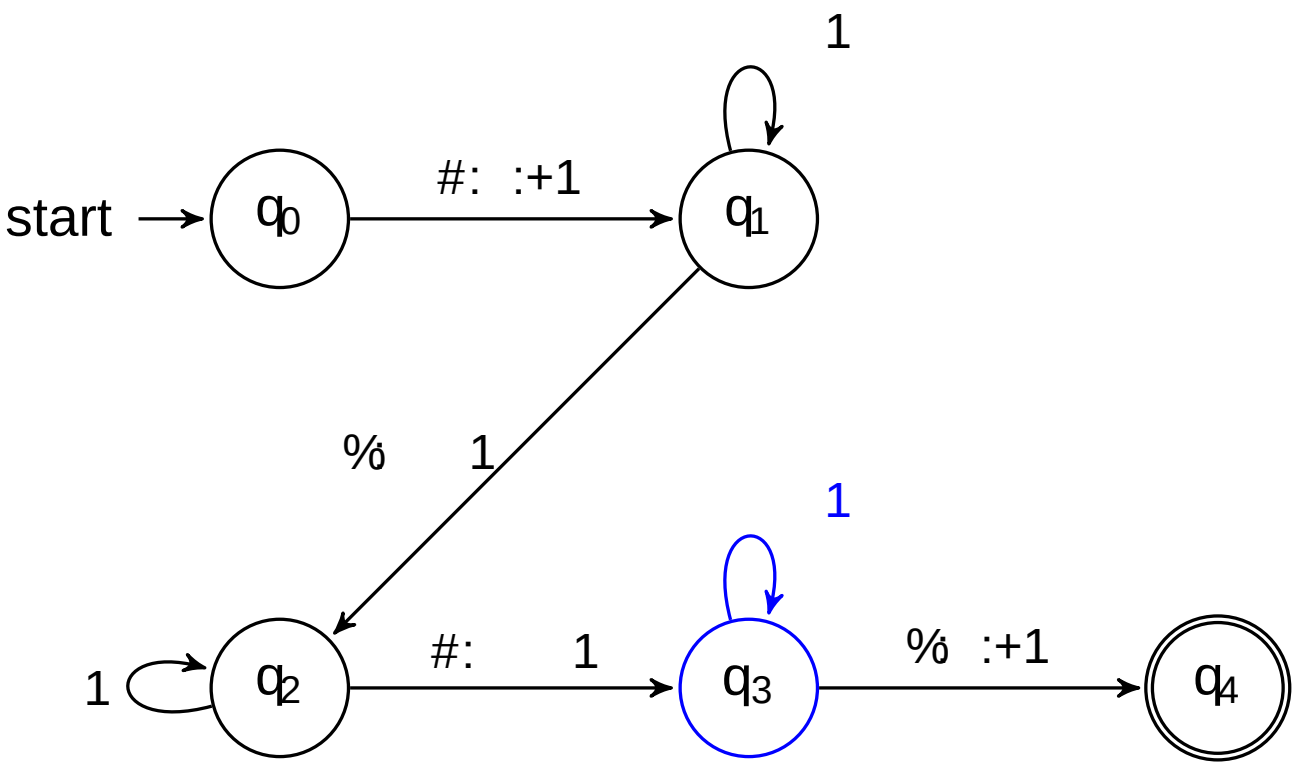
Input:	#	b	y	e	%
Output:		b	y	e	
		b	y		



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

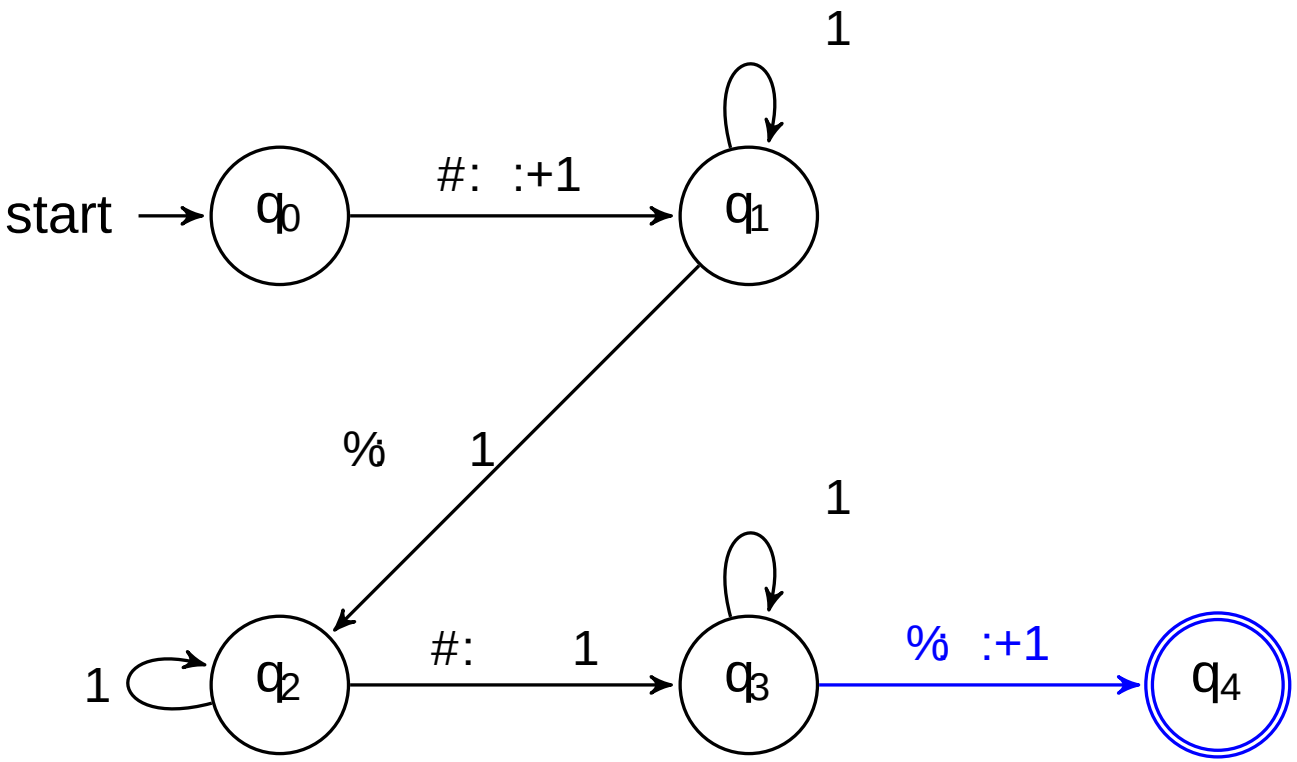
Input:	#	b	y	e	%
Output:		b	y	e	
		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita wanita wanita
- Working example: bye bye bye

Input:	#	b	y	e	%
Output:		b	y	e	
		b	y	e	



Total reduplication with 2-way FSTs

- Indonesian example: wanita→wanita□wanita
- Working example: bye→bye□bye

Input:	□	b	y	e	□
Output:		b	y	e	□
		b	y	e