Adding reduplication to finite-state machinery:

Finite-state buffered machines as recognizers

Yang Wang

Department of Linguistics
University of California, Los Angeles

yangwangx@g.ucla.edu

ESSLLI Student Session 2021

The puzzle of reduplication I

- Copying in natural language phonology and morphology

The puzzle of reduplication I

- Copying in natural language phonology and morphology
  - Total reduplication: Dyirbal plurals (Dixon, 1972, 242):

midi	*'little, small'*	midi-midi	*'lots of little ones'*
gulgiṛi	*'prettily painted men'*	gulgiṛi-gulgiṛi	*'lots of prettily painted men'*

[1](Rubino, 2013); (Dolatian and Heinz, 2020)

The puzzle of reduplication I

- Copying in natural language phonology and morphology
  - Total reduplication: Dyirbal plurals (Dixon, 1972, 242):

midi	*'little, small'*	midi-midi	*'lots of little ones'*
gulgiɽi	*'prettily painted men'*	gulgiɽi-gulgiɽi	*'lots of prettily painted men'*

  - Partial reduplication: Agta plurals (Healey, 1960,7):

labáng	*'patch'*	lab-labáng	*'patches'*
takki	*'leg'*	tak-takki	*'legs'*

[1](Rubino, 2013); (Dolatian and Heinz, 2020)

The puzzle of reduplication I

- Copying in natural language phonology and morphology
  - Total reduplication: Dyirbal plurals (Dixon, 1972, 242):

midi	*'little, small'*	midi-midi	*'lots of little ones'*
gulgiɽi	*'prettily painted men'*	gulgiɽi-gulgiɽi	*'lots of prettily painted men'*

  - Partial reduplication: Agta plurals (Healey, 1960,7):

labáng	*'patch'*	lab-labáng	*'patches'*
takki	*'leg'*	tak-takki	*'legs'*

- Reduplication is common cross-linguistically.

[1](Rubino, 2013); (Dolatian and Heinz, 2020)

The puzzle of reduplication I

- Copying in natural language phonology and morphology
  - Total reduplication: Dyirbal plurals (Dixon, 1972, 242):

midi	*'little, small'*	midi-midi	*'lots of little ones'*
gulgiɽi	*'prettily painted men'*	gulgiɽi-gulgiɽi	*'lots of prettily painted men'*

  - Partial reduplication: Agta plurals (Healey, 1960,7):

labáng	*'patch'*	lab-labáng	*'patches'*
takki	*'leg'*	tak-takki	*'legs'*

- Reduplication is common cross-linguistically.
  - 313 out of 368: productive reduplication ✓[1]

[1](Rubino, 2013); (Dolatian and Heinz, 2020)

The puzzle of reduplication I

- Copying in natural language phonology and morphology
  - Total reduplication: Dyirbal plurals (Dixon, 1972, 242):

midi	*'little, small'*	midi-midi	*'lots of little ones'*
gulgiɽi	*'prettily painted men'*	gulgiɽi-gulgiɽi	*'lots of prettily painted men'*

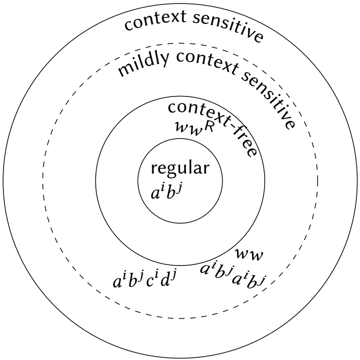
  - Partial reduplication: Agta plurals (Healey, 1960,7):

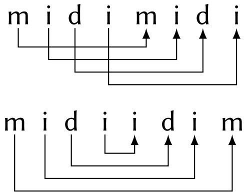
labáng	*'patch'*	lab-labáng	*'patches'*
takki	*'leg'*	tak-takki	*'legs'*

- Reduplication is common cross-linguistically.
  - 313 out of 368: productive reduplication ✓ [1]
- String reversals are rarely attested.
  - confined to language games (Bagemihl, 1989)

[1](Rubino, 2013); (Dolatian and Heinz, 2020)

The puzzle of reduplication II





- unbounded copying $\{ww \mid w \in \Sigma^*\}$
  - CH: mildly context-sensitive
  - NL: prevalent
- string reversal $\{ww^R \mid w \in \Sigma^*\}$
  - CH: context-free
  - NL: rare
- most phonology and morphology: regular

Main question

How can one fit in reduplicated *strings* without including some unattested context-free patterns, such as reversals?

Computing reduplication but not reversals

To exclude string reversals,

- Approaches that do not extend the expressivity and can only approximate productive total reduplication: Walther (2000),Cohen-Sygal and Wintner (2006), Hulden (2009)...

Computing reduplication but not reversals

To exclude string reversals,

- Approaches that do not extend the expressivity and can only approximate productive total reduplication: Walther (2000),Cohen-Sygal and Wintner (2006), Hulden (2009)...

  N.B.

  - Cannot really capture unbounded copying

Computing reduplication but not reversals

To exclude string reversals,

- Approaches that do not extend the expressivity and can only approximate productive total reduplication: Walther (2000),Cohen-Sygal and Wintner (2006), Hulden (2009)...

  N.B.

  - Cannot really capture unbounded copying

- A recent sequence of works (Dolatian and Heinz, 2018a; Dolatian and Heinz, 2018b; Dolatian and Heinz, 2019; Dolatian and Heinz, 2020) proposes 2-way finite-state **transducers** to model unbounded copying, and further developed sub-classes to exclude mirror image relations.

Computing reduplication but not reversals

To exclude string reversals,

- Approaches that do not extend the expressivity and can only approximate productive total reduplication: Walther (2000),Cohen-Sygal and Wintner (2006), Hulden (2009)...

  N.B.

  - Cannot really capture unbounded copying

- A recent sequence of works (Dolatian and Heinz, 2018a; Dolatian and Heinz, 2018b; Dolatian and Heinz, 2019; Dolatian and Heinz, 2020) proposes 2-way finite-state **transducers** to model unbounded copying, and further developed sub-classes to exclude mirror image relations.

  N.B.

  - reduplication is modeled as *functions*
    midi → midi-midi
  - the morphological analysis problem ☹
    midi-midi → midi

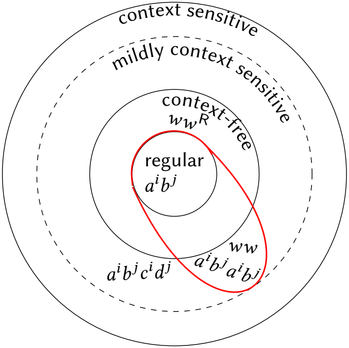
Main question

How can one fit in reduplicated *strings* without including some unattested context-free patterns, such as reversals?

Gazdar and Pullum (1985, p. 278)

> *We do not know whether there exists an independent characterization of the class of languages that includes the regular sets and languages derivable from them through reduplication,…this class might be relevant to the characterization of NL word-sets.*

Goals of this talk



- give a formal characterization of regular languages + the copying-derived languages
  - unbounded copying ✓
  - string reversal ✗
  - Swiss-German crossing dependencies ✗
- analyze the closure properties of this class of languages

Outline

Finite-state buffered machines

Finite-state automata + a copying mechanism

Finite-state buffered machines

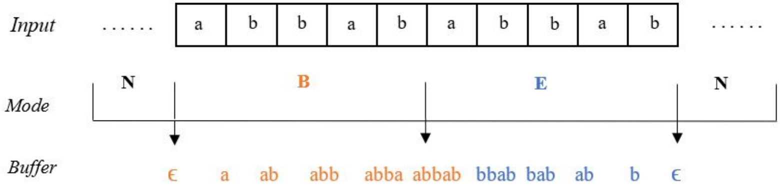
Finite-state automata + a copying mechanism

- The copying mechanism
  - an unbounded memory buffer, with queue storage

Finite-state buffered machines

Finite-state automata + a copying mechanism

- The copying mechanism
  - an unbounded memory buffer, with queue storage
  - three different modes
    - normal (N) mode: similar to a normal FSA
    - buffering (B) mode: copying input symbols to the buffer
    - emptying (E) mode: matching symbols in memory against the input



When to switch to which mode?

When to switch to which mode?

Based on different states...

When to switch to which mode?

Based on different states...

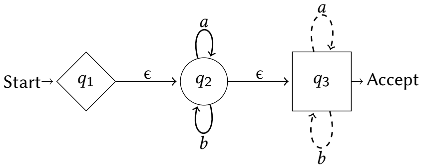


Figure: $L(\mathsf{M}_1) = \{ww \mid w \in \{a, b\}^*\}$

When to switch to which mode?

Based on different states...

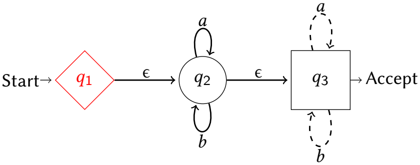


Figure: $L(\mathsf{M}_1) = \{ww|w \in \{a,b\}^*\}$

- q_1: "switch to buffering mode, please"

When to switch to which mode?

Based on different states...

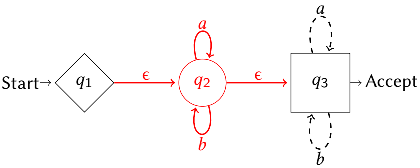


Figure: $L(M_1) = \{ww | w \in \{a, b\}^*\}$

- q_1: “switch to buffering mode, please”
- keeps in buffering mode and stores symbols in the buffer (queue)

When to switch to which mode?

Based on different states...

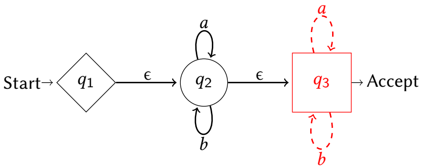


Figure: $L(\mathsf{M}_1) = \{ww | w \in \{a, b\}^*\}$

- q_1: "switch to buffering mode, please"
- keeps in buffering mode and storing symbols in the buffer (queue)
- q_3: "stop buffering and let's empty the buffer if strings match up "

Finite-state buffered machines: formal definition

A **Finite-State Buffered Machine (FSBM)** is a 7-tuple $\langle \Sigma, Q, I, F, \mathbf{G}, \mathbf{H}, \delta \rangle$ where

- Q: a finite set of states
- $I \subseteq Q$: initial states
- $F \subseteq Q$: final states
- $G \subseteq Q$: **states where the machine must enter buffering (B) mode**
- $H \subseteq Q$: **states visited while the machine is emptying the buffer**
- $G \cap H = \varnothing$
- δ: $Q \times (\Sigma \cup \{\epsilon\}) \times Q$: the state transitions according to a specific symbol

Finite-state buffered machines: configuration

A configuration of an FSBM D = $(u, q, \mathbf{v}, \mathbf{t}) \in \Sigma^* \times Q \times \Sigma^* \times \{\text{N}, \text{B}, \text{E}\}$

- u: the input string
- q: the state the machine is currently in
- v: **the string in the buffer**
- t: **the mode the machine is currently in**

Finite-state buffered machines: configuration transition

Given an FSBM M and $x \in (\Sigma \cup \{\epsilon\})$, $u, w, v \in \Sigma^*$, we define a configuration D_1 **yields** a configuration D_2 in M ($\mathsf{D}_1 \vdash_{\mathsf{M}} \mathsf{D}_2$) as the smallest relation such that:

Finite-state buffered machines: configuration transition

Given an FSBM M and $x \in (\Sigma \cup \{\epsilon\})$, $u, w, v \in \Sigma^*$, we define a configuration D_1 **yields** a configuration D_2 in M ($D_1 \vdash_M D_2$) as the smallest relation such that:

- For every transition (q_1, x, q_2) with at least one state of $q_1, q_2 \notin H$
  $(xu, q_1, \epsilon, \text{N}) \vdash_M (u, q_2, \epsilon, \text{N})$ with $q_1 \notin G$ "normal" actions
  $(xu, q_1, v, \text{B}) \vdash_M (u, q_2, vx, \text{B})$ with $q_2 \notin G$ "buffering" actions
- For every transition (q_1, x, q_2) and $q_1, q_2 \in H$
  $(xu, q_1, xv, \text{E}) \vdash_M (u, q_2, v, \text{E})$ "emptying" actions

Finite-state buffered machines: configuration transition

Given an FSBM M and $x \in (\Sigma \cup \{\epsilon\})$, $u, w, v \in \Sigma^*$, we define a configuration D_1 **yields** a configuration D_2 in M ($D_1 \vdash_M D_2$) as the smallest relation such that:

- For every transition (q_1, x, q_2) with at least one state of $q_1, q_2 \notin H$
  $(xu, q_1, \epsilon, \text{N}) \vdash_M (u, q_2, \epsilon, \text{N})$ with $q_1 \notin G$ "normal" actions
  $(xu, q_1, v, \text{B}) \vdash_M (u, q_2, vx, \text{B})$ with $q_2 \notin G$ "buffering" actions
- For every transition (q_1, x, q_2) and $q_1, q_2 \in H$
  $(xu, q_1, xv, \text{E}) \vdash_M (u, q_2, v, \text{E})$ "emptying" actions
- For every $q \in G$
  $(u, q, \epsilon, \text{N}) \vdash_M (u, q, \epsilon, \text{B})$ *mode-changing actions*
- For every $q \in H$
  $(u, q, v, \text{B}) \vdash_M (u, q, v, \text{E})$ *mode-changing actions*
  $(u, q, \epsilon, \text{E}) \vdash_M (u, q, \epsilon, \text{N})$ *mode-changing actions*

Outline

An example run: abbabb

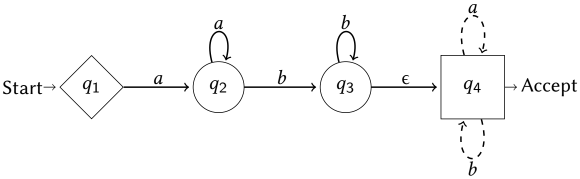


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

An example run: abbabb

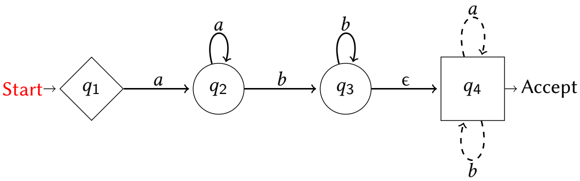


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

An example run: abbabb

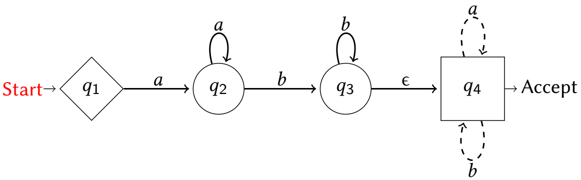


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N

An example run: abbabb

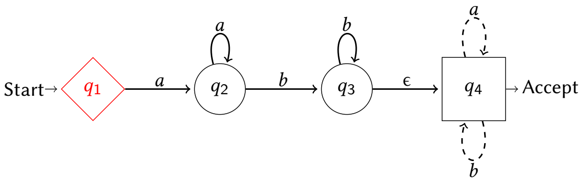


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B

An example run: abbabb

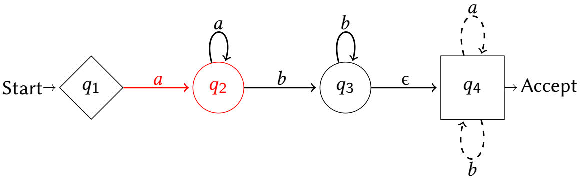


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B

An example run: abbabb

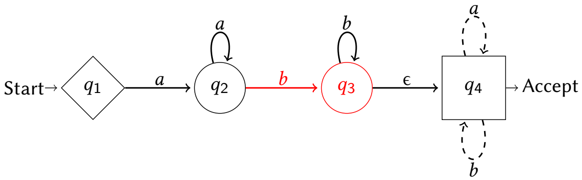


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B

An example run: abbabb

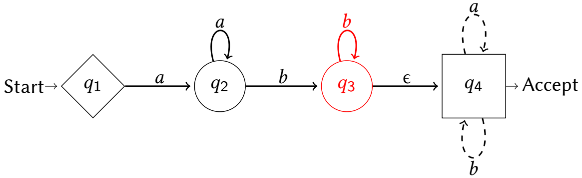


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	b
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B

An example run: abbabb

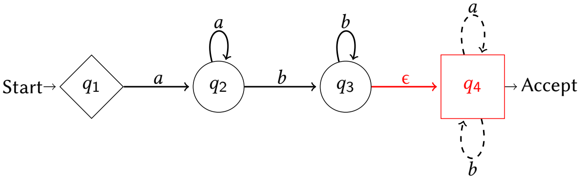


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$*.* $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B
abb	q_4	abb	B

An example run: abbabb

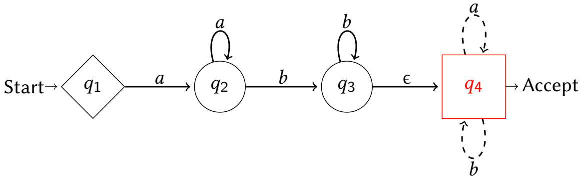


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B
abb	q_4	abb	B

input	state	buffer	mode
abb	q_4	abb	E

An example run: abbabb

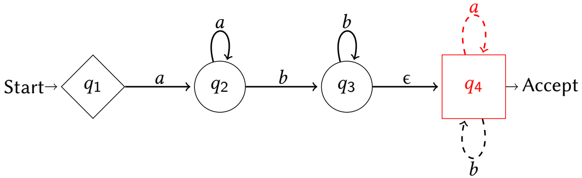


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B
abb	q_4	abb	B

input	state	buffer	mode
abb	q_4	abb	E
bb	q_4	bb	E

An example run: abbabb

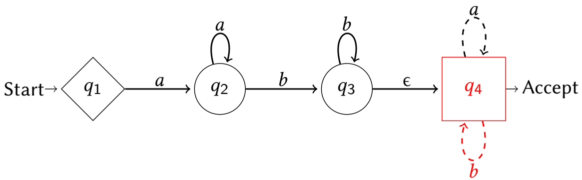


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B
abb	q_4	abb	B

input	state	buffer	mode
abb	q_4	abb	E
bb	q_4	bb	E
b	q_4	b	E

An example run: abbabb

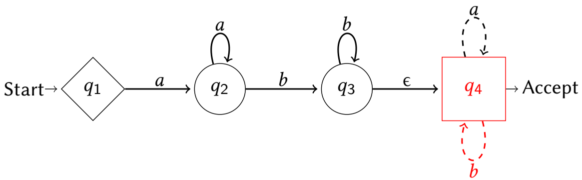


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathrm{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B
abb	q_4	abb	B

input	state	buffer	mode
abb	q_4	abb	E
bb	q_4	bb	E
b	q_4	b	E
ϵ	q_4	ϵ	E

An example run: abbabb

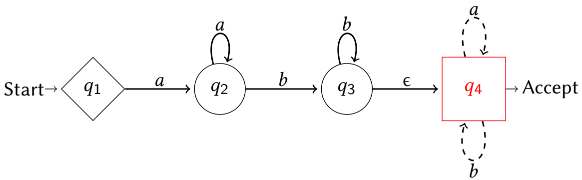


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B
abb	q_4	abb	B

input	state	buffer	mode
abb	q_4	abb	E
bb	q_4	bb	E
b	q_4	b	E
ϵ	q_4	ϵ	E
ϵ	q_4	ϵ	N

An example run: abbabb

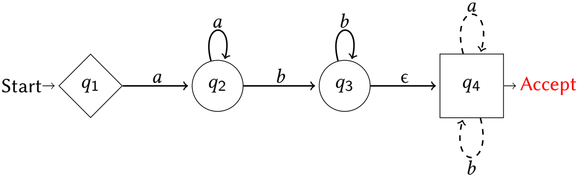


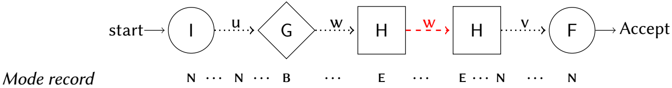
Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
abbabb	q_1	ϵ	N
abbabb	q_1	ϵ	B
bbabb	q_2	a	B
babb	q_3	ab	B
abb	q_3	abb	B
abb	q_4	abb	B

input	state	buffer	mode
abb	q_4	abb	E
bb	q_4	bb	E
b	q_4	b	E
ϵ	q_4	ϵ	E
ϵ	q_4	ϵ	N
ACCEPT			

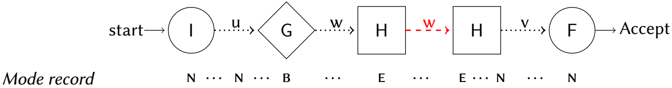
Outline

Different arrangements of G & H states

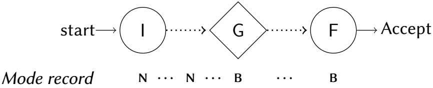


(a) *Useful arrangements of G and H states*

Different arrangements of G & H states

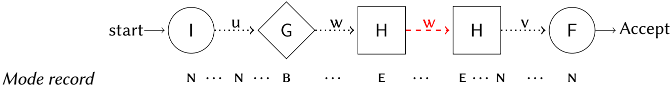


(a) *Useful arrangements of G and H states*

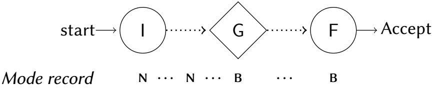


(b) *Non-Useful arrangements: No H states*

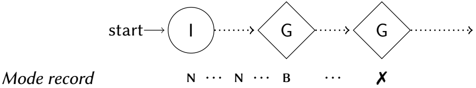
Different arrangements of G & H states



(a) *Useful arrangements of G and H states*

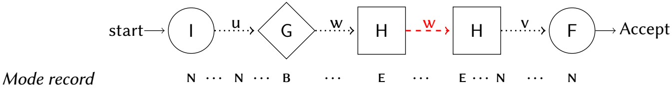


(b) *Non-Useful arrangements: No H states*

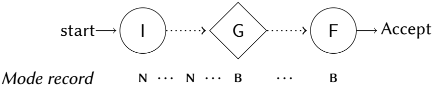


(c) *Non-Useful arrangements: No H states in between two G states along a path*

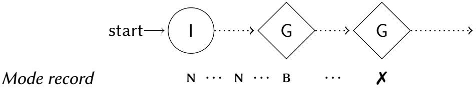
Different arrangements of G & H states



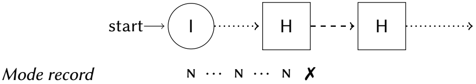
(a) *Useful arrangements of G and H states*



(b) *Non-Useful arrangements: No H states*



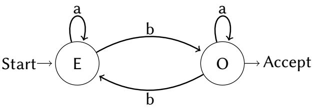
(c) *Non-Useful arrangements: No H states in between two G states along a path*



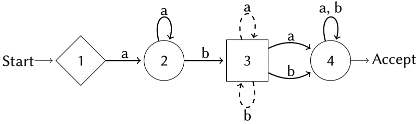
(d) *Non-Useful arrangements: H states before G states*

Outline

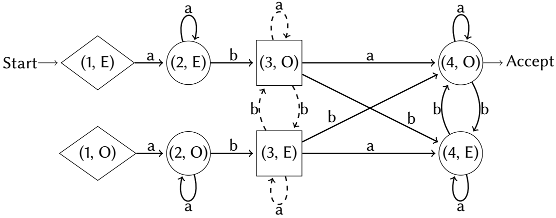
Intersection with regular languages



(a) *an FSA enforcing odd number of bs in a string. State E and State O represent even, odd number of bs in the prefix respectively*



(b) *a machine recognizing initial 'aa*b'-identity.* G = {1}, H = {3}

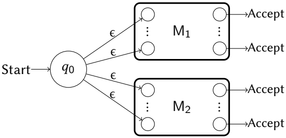


(c) *the intersection FSBM after the construction:* G = {(1, E), (1, O)}, H = {(3, 0), (3, E)}.

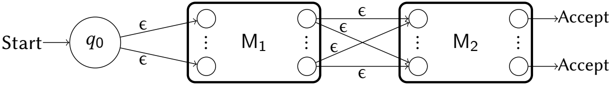
Outline

Regular operations [2]

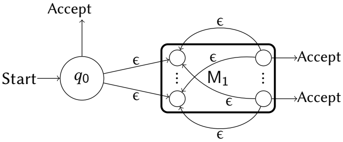
- union



- concatenation:



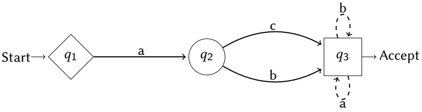
- Kleene Star



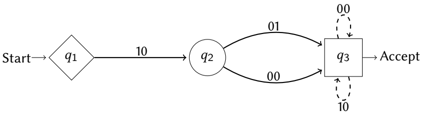
[2] That FSBM languages are closed under regular operations hints that the set of languages recognized by the new automata might be equivalent to the set of languages denoted by a version of regular expression with copying added.

Outline

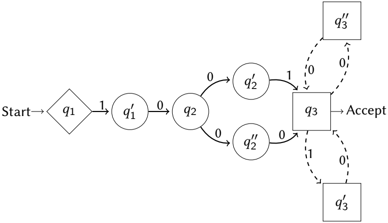
Homomorphism



(a) $\mathsf{L}(\mathsf{M}) = \{abab\}$

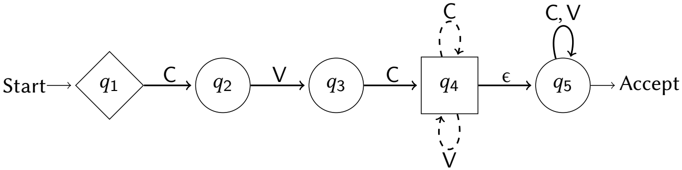


(b) $h(a) = 10, h(b) = 00, h(c) = 01$. *The intermediate step when the arcs are relabeled with mapped strings*

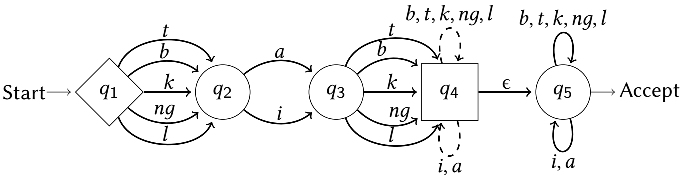


(c) *States* $q_1', q_2', q_2'', q_3', q_3''$ *are added to split the arcs.* $\mathsf{L}(\mathsf{M}_h) = \{10001000\}$

Inverse homomorphism?



(a) *An FSBM on the alphabet* $\{\mathsf{C}, \mathsf{V}\}$*: Agta initial CVC-reduplication. Assume* $\Sigma = \{t, b, k, ng, l, i, a\}$*. Consider a homomorphism* $\Sigma \to \{\mathsf{C}, \mathsf{V}\}$ *maps each consonant to C and each vowel to V.*



(b) *Under-generation of the conventional construction of the inverse homomorphic image: only bakbak-; never baktal-*

Inverse homomorphism?

- Consider $\mathsf{L} = \{a^i b^j a^i b^j \mid i, j \geq 1\}$.
- An alphabetic homomorphism $h : \{0, 1, 2\} \rightarrow \{a, b\}^*$ with $h(0) = a$, $h(1) = a$ and $h(2) = b$.

Inverse homomorphism?

- Consider $\mathsf{L} = \{a^i b^j a^i b^j \mid i, j \geq 1\}$.
- An alphabetic homomorphism $h : \{0, 1, 2\} \to \{a, b\}^*$ with $h(0) = a$, $h(1) = a$ and $h(2) = b$.
- $h^{-1}(\mathsf{L}) = \{(0|1)^i 2^j (0|1)^i 2^j \mid i, j \geq 1\}$
  - The language $\{w2^j w2^j \mid w \in \{0, 1\}^*, j \geq 1\} \subset h^{-1}(\mathsf{L})$.
  - including 1202, 11020002, 1012201022.

Inverse homomorphism?

- Consider $L = \{a^i b^j a^i b^j \mid i, j \geq 1\}$.
- An alphabetic homomorphism $h : \{0, 1, 2\} \to \{a, b\}^*$ with $h(0) = a$, $h(1) = a$ and $h(2) = b$.
- $h^{-1}(L) = \{(0|1)^i 2^j (0|1)^i 2^j \mid i, j \geq 1\}$
  - The language $\{w2^j w2^j \mid w \in \{0, 1\}^*, j \geq 1\} \subset h^{-1}(L)$.
  - including 1202, 11020002, 1012201022.

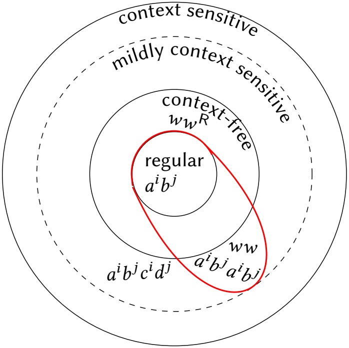
$\boxed{\text{FSBM:}}$

- FSA: the incurred crossing dependencies: ☹
- the augmented copying mechanism: ☹
  - only *identical* copies
  - not general cases of symbol correspondence.

Closure properties: a summary

Operations	Closed or not
union	✓
concatenation	✓
Kleene star	✓
homomorphism	✓
intersection with regular languages	✓
inverse homomorphism	✗?

Conclusion



- **Finite-state buffered machines**
  - FSA + a copying mechanism
  - compute productive total reduplication on any regular languages
- introduce a new class of languages incomparable to CFLs.
  - string reversal ✗
    queue-like buffer
  - Swiss-German crossing dependencies ✗

Thank you!

References I

Bagemihl, Bruce (1989). “The crossing constraint and ‘backwards languages’”. In: *Natural language & linguistic Theory* 7.4, pp. 481–549.

Cohen-Sygal, Yael and Shuly Wintner (2006). “Finite-state registered automata for non-concatenative morphology”. In: *Computational Linguistics* 32.1, pp. 49–82.

Dixon, Robert M. W. (1972). *The Dyirbal Language of North Queensland*. Vol. 9. Cambridge Studies in Linguistics. Cambridge: Cambridge University Press.

Dolatian, Hossep and Jeffrey Heinz (2018a). “Learning reduplication with 2-way finite-state transducers”. In: *Proceedings of the 14th International Conference on Grammatical Inference*. Ed. by Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek. Vol. 93. Proceedings of Machine Learning Research. PMLR, pp. 67–80.

— (Oct. 2018b). “Modeling Reduplication with 2-way Finite-State Transducers”. In: *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Brussels, Belgium: Association for Computational Linguistics, pp. 66–77. DOI: 10.18653/v1/W18-5807.

— (2019). “RedTyp: A Database of Reduplication with Computational Models”. In: *Proceedings of the Society for Computation in Linguistics*. Vol. 2. Article 3.

— (2020). “Computing and classifying reduplication with 2-way finite-state transducers”. In: *Journal of Language Modelling* 8.1, pp. 179–250.

Gazdar, Gerald and Geoffrey K Pullum (1985). “Computationally relevant properties of natural languages and their grammars”. In: *New generation computing* 3.3, pp. 273–306.

Healey, Phyllis M. (1960). *An Agta Grammar*. Manila: Bureau of Printing.

References II

Hulden, Mans (2009). "Finite-state Machine Construction Methods and Algorithms for Phonology and Morphology". PhD thesis. University of Arizona, Tucson, USA. URL: http://hdl.handle.net/10150/196112.

Rubino, Carl (2013). "Reduplication". In: *The World Atlas of Language Structures Online*. Ed. by Matthew S. Dryer and Martin Haspelmath. Leipzig: Max Planck Institute for Evolutionary Anthropology. URL: https://wals.info/chapter/27.

Walther, Markus (2000). "Finite-State Reduplication in One-Level Prosodic Morphology". In: *1st Meeting of the North American Chapter of the Association for Computational Linguistics*. URL: https://www.aclweb.org/anthology/A00-2039.

Acknowledgments

I would like to express gratitude to Tim Hunter, Bruce Hayes, Dylan Bumford, Kie Zuraw, Ed Keenan and the members of the UCLA Phonology Seminar for their comments, feedback and insights. Special thanks to the anonymous reviewers for their constructive suggestions and discussions. All errors remain my own.

Appendix: Typology of reduplication

- current machinery: local reduplication with *two* adjacent, completely identical copies.

Appendix: Typology of reduplication

- current machinery: local reduplication with *two* adjacent, completely identical copies.
  - non-local copies ☹

    Chukchee absolute singular

    'voice' quli quli-qul
  - multiple copies ☹

    Mokilese

    'give a shudder' roar roar-roar-roar *'continue to shudder'*
  - non-identical copies ☹

    Javanese Habitual Repetitive

    'remember' eliŋ elaŋ-eliŋ

Appendix: definition of homomorphism and inverse homomorphism

Definition

A (string) homomorphism is a function mapping one alphabet to strings of another alphabet, written $h : \Sigma \to \Delta^*$. We can extend h to operate on strings over Σ^* such that

1. $h(\epsilon_\Sigma) = \epsilon_\Delta$
2. for $w = a_1 a_2 \dots a_n \in \Sigma^*$, $h(w) = h(a_1)h(a_2)\dots h(a_n)$ where each $a_i \in \Sigma$

Definition

An alphabetic homomorphism h_0 is a special homomorphism with h_0 maps each symbol in previous alphabet to another symbol in the new alphabet.

Appendix: definition of homomorphism and inverse homomorphism

Definition

A (string) homomorphism is a function mapping one alphabet to strings of another alphabet, written $h : \Sigma \to \Delta^*$. We can extend h to operate on strings over Σ^* such that

1. $h(\epsilon_\Sigma) = \epsilon_\Delta$
2. for $w = a_1 a_2 \ldots a_n \in \Sigma^*$, $h(w) = h(a_1)h(a_2)\ldots h(a_n)$ where each $a_i \in \Sigma$

Definition

An alphabetic homomorphism h_0 is a special homomorphism with h_0 maps each symbol in previous alphabet to another symbol in the new alphabet.

$\boxed{\text{Example:}}$ $h : \{\mathrm{H}, \mathrm{L}\} \to \{\mathrm{C}, \mathrm{V}\}^*$ with $h(\mathrm{H}) = \mathrm{CVC}$ and $h(\mathrm{L}) = \mathrm{CV}$.

Appendix: definition of homomorphism and inverse homomorphism

Definition

A (string) homomorphism is a function mapping one alphabet to strings of another alphabet, written $h : \Sigma \to \Delta^*$. We can extend h to operate on strings over Σ^* such that

1. $h(\epsilon_\Sigma) = \epsilon_\Delta$
2. for $w = a_1 a_2 \dots a_n \in \Sigma^*$, $h(w) = h(a_1)h(a_2)\dots h(a_n)$ where each $a_i \in \Sigma$

Definition

An alphabetic homomorphism h_0 is a special homomorphism with h_0 maps each symbol in previous alphabet to another symbol in the new alphabet.

Example: $h : \{\mathrm{H}, \mathrm{L}\} \to \{\mathrm{C}, \mathrm{V}\}^*$ with $h(\mathrm{H}) = \mathrm{CVC}$ and $h(\mathrm{L}) = \mathrm{CV}$.
$h(\mathrm{LLH}) = h(\mathrm{L})h(\mathrm{L})h(\mathrm{H}) = \mathrm{CVCVCVC}$.

Appendix: definition of homomorphism and inverse homomorphism

Definition

Given a homomorphism $h\colon \Sigma \to \Delta^*$ and $\mathsf{L}_1 \subseteq \Sigma^*$, $\mathsf{L}_2 \subseteq \Delta^*$, define $h(\mathsf{L}_1) = \{h(w) \mid w \in \mathsf{L}_1\} \subseteq \Delta^*$ and $h^{-1}(\mathsf{L}_2) = \{w \mid h(w) = v \in \mathsf{L}_2\} \subseteq \Sigma^*$.

Appendix: definition of homomorphism and inverse homomorphism

Definition

Given a homomorphism $h \colon \Sigma \to \Delta^*$ and $\mathsf{L}_1 \subseteq \Sigma^*$, $\mathsf{L}_2 \subseteq \Delta^*$, define $h(\mathsf{L}_1) = \{h(w) \mid w \in \mathsf{L}_1\} \subseteq \Delta^*$ and $h^{-1}(\mathsf{L}_2) = \{w \mid h(w) = v \in \mathsf{L}_2\} \subseteq \Sigma^*$.

Example: $h : \{\mathsf{H}, \mathsf{L}\} \to \{\mathsf{C}, \mathsf{V}\}^*$ with $h(\mathsf{H}) = \mathsf{CVC}$ and $h(\mathsf{L}) = \mathsf{CV}$.

Appendix: definition of homomorphism and inverse homomorphism

Definition

Given a homomorphism $h: \Sigma \to \Delta^*$ and $L_1 \subseteq \Sigma^*$, $L_2 \subseteq \Delta^*$, define $h(L_1) = \{h(w) \mid w \in L_1\} \subseteq \Delta^*$ and $h^{-1}(L_2) = \{w \mid h(w) = v \in L_2\} \subseteq \Sigma^*$.

$\boxed{\text{Example:}}$ $h : \{H, L\} \to \{C, V\}^*$ with $h(H) = CVC$ and $h(L) = CV$.

$L_1 = \{(LH)^n \mid n \in \mathbb{N}\}$

Appendix: definition of homomorphism and inverse homomorphism

Definition

Given a homomorphism h: $\Sigma \to \Delta^*$ and $\mathsf{L}_1 \subseteq \Sigma^*$, $\mathsf{L}_2 \subseteq \Delta^*$, define $h(\mathsf{L}_1) = \{h(w) \,|\, w \in \mathsf{L}_1\} \subseteq \Delta^*$ and $h^{-1}(\mathsf{L}_2) = \{w \,|\, h(w) = v \in \mathsf{L}_2\} \subseteq \Sigma^*$.

$\boxed{\text{Example:}}$ $h : \{\mathsf{H}, \mathsf{L}\} \to \{\mathsf{C}, \mathsf{V}\}^*$ with $h(\mathsf{H}) = \mathsf{CVC}$ and $h(\mathsf{L}) = \mathsf{CV}$.

$$\mathsf{L}_1 = \{(\mathsf{LH})^n \,|\, n \in \mathbb{N}\} \to h(\mathsf{L}_1) = \{(\mathsf{CVCVC})^n \,|\, n \in \mathbb{N}\}.$$

Appendix: definition of homomorphism and inverse homomorphism

Definition

Given a homomorphism h: $\Sigma \to \Delta^*$ and $L_1 \subseteq \Sigma^*$, $L_2 \subseteq \Delta^*$, define $h(L_1) = \{h(w) \mid w \in L_1\} \subseteq \Delta^*$ and $h^{-1}(L_2) = \{w \mid h(w) = v \in L_2\} \subseteq \Sigma^*$.

$\boxed{\text{Example:}}$ $h : \{H, L\} \to \{C, V\}^*$ with $h(H) = CVC$ and $h(L) = CV$.

$$L_1 = \{(LH)^n \mid n \in \mathbb{N}\} \to h(L_1) = \{(CVCVC)^n \mid n \in \mathbb{N}\}.$$

$$L_2 = \{(CV)^n (CVC)^n\}$$

Appendix: definition of homomorphism and inverse homomorphism

Definition

Given a homomorphism $h \colon \Sigma \to \Delta^*$ and $L_1 \subseteq \Sigma^*$, $L_2 \subseteq \Delta^*$, define $h(L_1) = \{h(w) \mid w \in L_1\} \subseteq \Delta^*$ and $h^{-1}(L_2) = \{w \mid h(w) = v \in L_2\} \subseteq \Sigma^*$.

$\boxed{\text{Example:}}$ $h : \{H, L\} \to \{C, V\}^*$ with $h(H) = CVC$ and $h(L) = CV$.

$$L_1 = \{(LH)^n \mid n \in \mathbb{N}\} \to h(L_1) = \{(CVCVC)^n \mid n \in \mathbb{N}\}.$$

$$L_2 = \{(CV)^n(CVC)^n\} \to h^{-1}(L_2) = \{L^n H^n\}.$$

Appendix: An example run: ababb

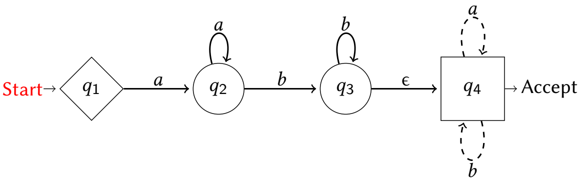


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

Appendix: An example run: ababb

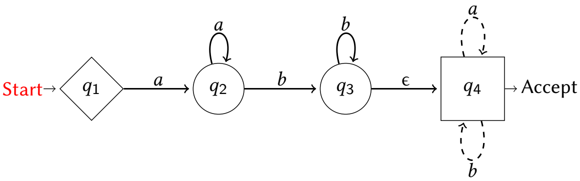


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N

Appendix: An example run: ababb

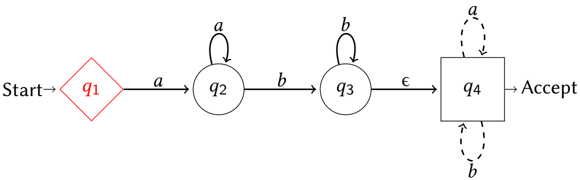


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B

Appendix: An example run: ababb

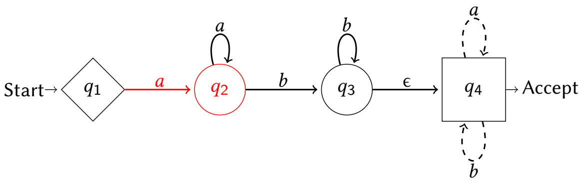


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B
babb	q_2	a	B

Appendix: An example run: ababb

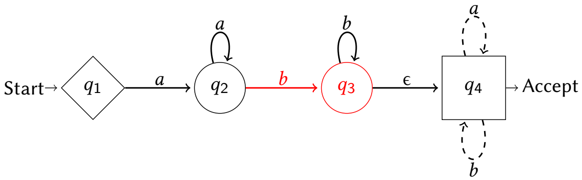


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B
babb	q_2	a	B
abb	q_3	ab	B

Appendix: An example run: ababb

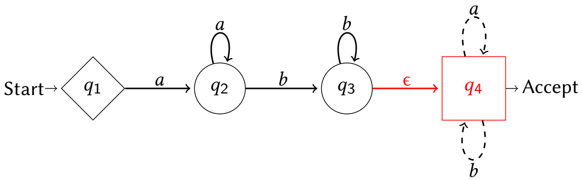


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathrm{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B
babb	q_2	a	B
abb	q_3	ab	B
abb	q_4	ab	B

Appendix: An example run: ababb

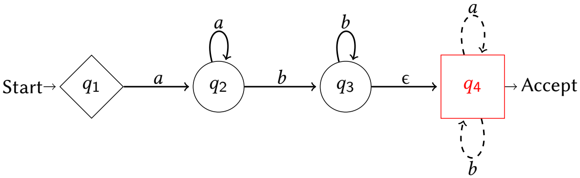


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B
babb	q_2	a	B
abb	q_3	ab	B
abb	q_4	ab	B

input	state	buffer	mode
abb	q_4	ab	E

Appendix: An example run: ababb

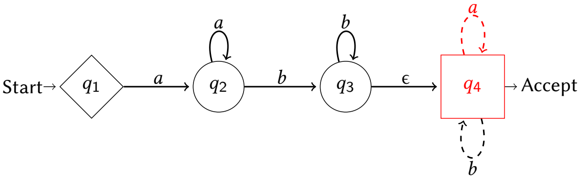


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathrm{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B
babb	q_2	a	B
abb	q_3	ab	B
abb	q_4	ab	B

input	state	buffer	mode
abb	q_4	ab	E
bb	q_4	b	E

Appendix: An example run: ababb

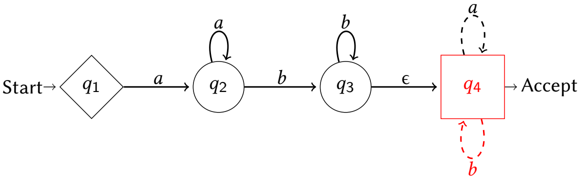


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B
babb	q_2	a	B
abb	q_3	ab	B
abb	q_4	ab	B

input	state	buffer	mode
abb	q_4	ab	E
bb	q_4	b	E
b	q_4	ϵ	E

Appendix: An example run: ababb

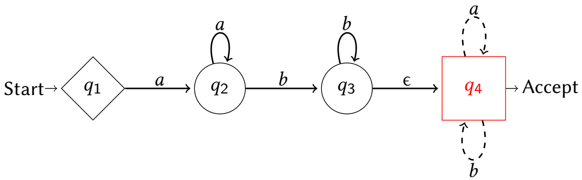


Figure: *An FSBM* M_2 *with* $G = \{q_1\}$ *and* $H = \{q_4\}$. $L(\mathsf{M}_2) = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

input	state	buffer	mode
ababb	q_1	ϵ	N
ababb	q_1	ϵ	B
babb	q_2	a	B
abb	q_3	ab	B
abb	q_4	ab	B

input	state	buffer	mode
abb	q_4	ab	E
bb	q_4	b	E
b	q_4	ϵ	E
b	q_4	ϵ	N