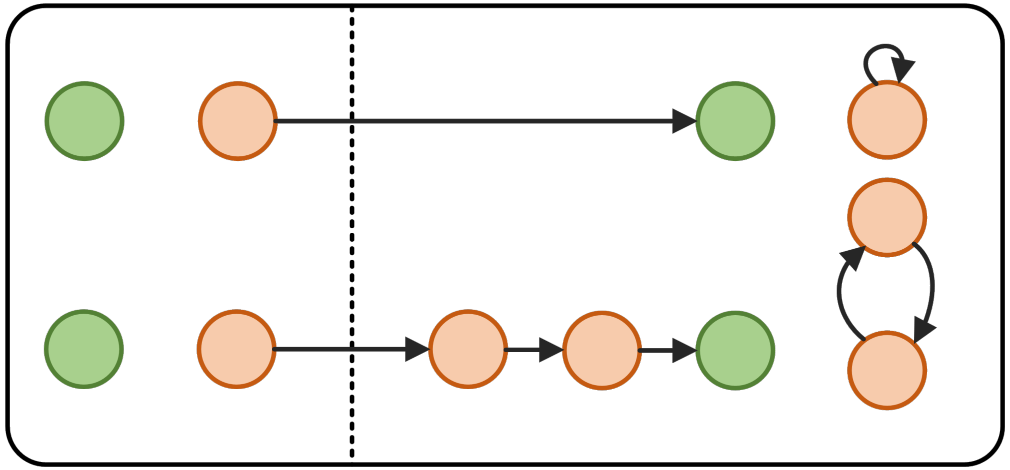


Codes for Constrained Periodicity

Adir Kobovich*, Orian Leitersdorf*, Daniella Bar-Lev, and Eitan Yaakobi

Technion – Israel Institute of Technology

** Equal Contribution*

Overview

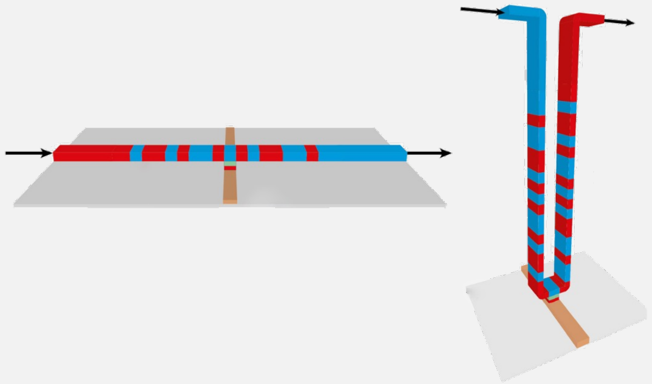
Constrained Periodicity

$n + r = 20$

010100100100111100010

$\ell = 9$

(period 3)

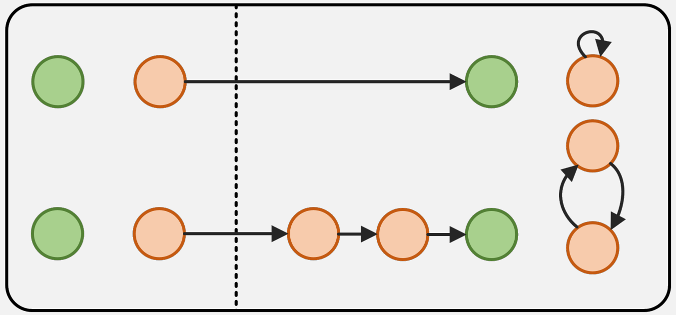


Iterative Construction

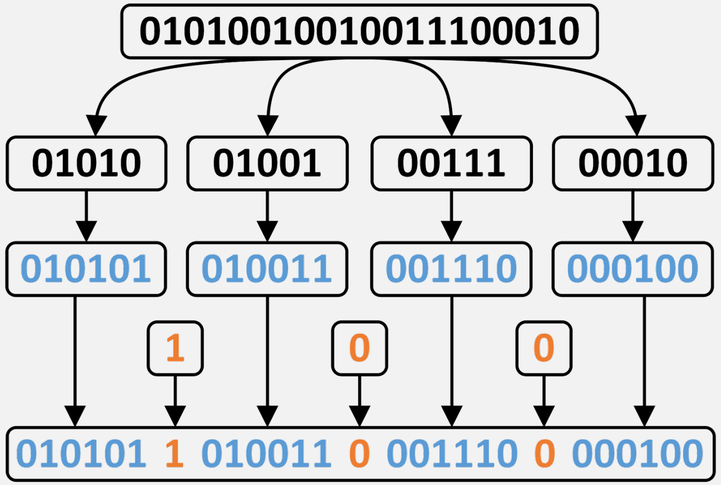
Near-Optimal Alg:

Algorithm 1 Encoder

1: **while** exists periodic window **do**
2: Fix identified window.
3: **end while**



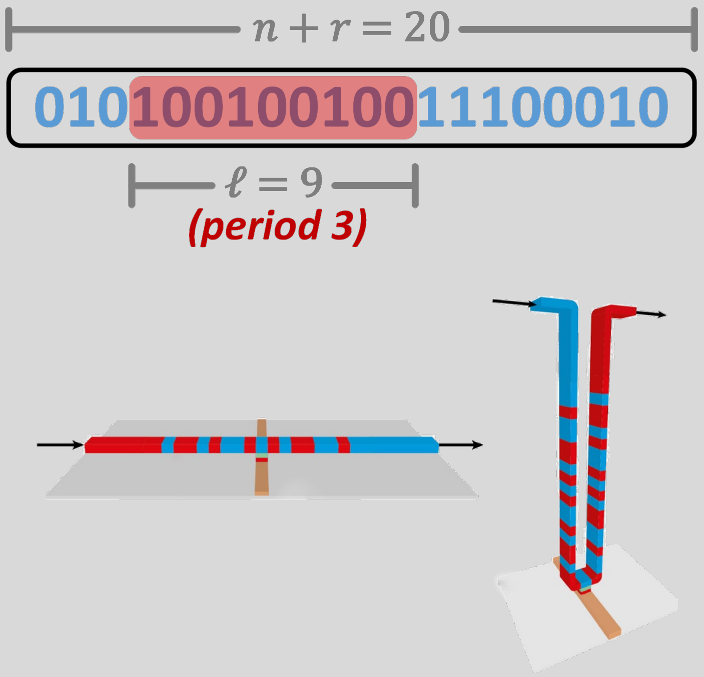
Extentions



Generalizing Beyond Periodicity

Overview

Constrained Periodicity

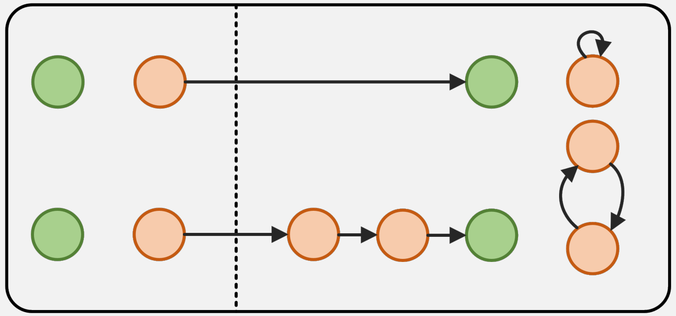


Iterative Construction

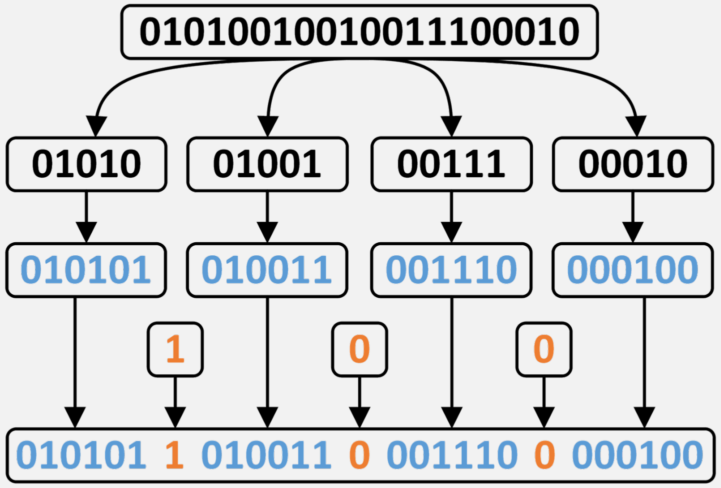
Near-Optimal Alg:

Algorithm 1 Encoder

1: **while** exists periodic window **do**
2: Fix identified window.
3: **end while**



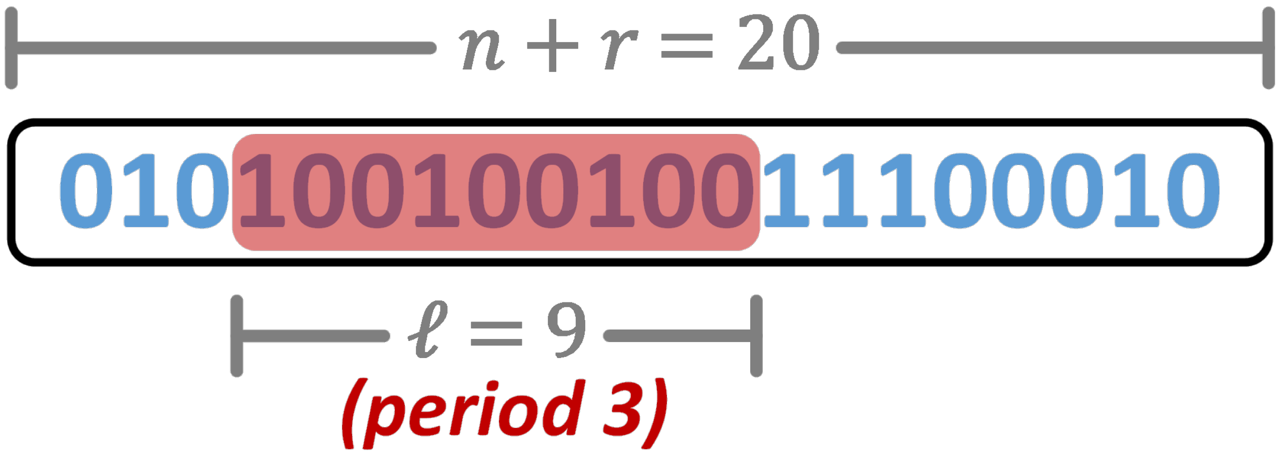
Extentions



Constrained Periodicity

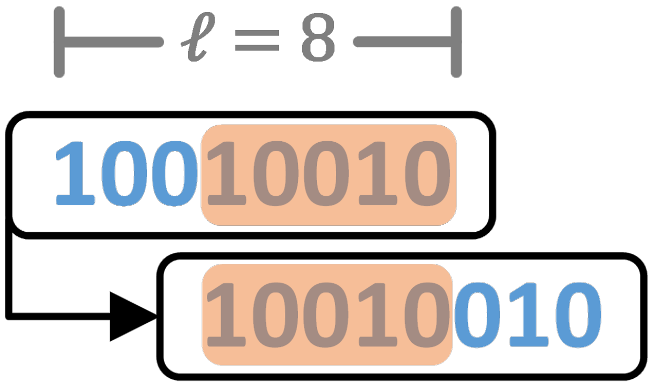
Problem:

- *Input:* $x \in \Sigma^n$
- *Output:* $y \in \Sigma^{n+r}$ **without** periodicity in all windows
  $(r = \#\, redundancy\ symbols)$



Periodicity

- $x \in \Sigma^{\ell}$ is p-periodic if $\forall i \leq \ell - p : x_i = x_{i+p}$



(period 3)

- Kernel: repetitive portion (e.g., 100)

Least-Period-Avoiding (LPA) Constraint

- $x \in \Sigma^{n+r}$ is ℓ-window p-least-period avoiding (LPA) if all windows do **not** have periodicity $< \boldsymbol{p}$ (where ℓ, p may depend on n)

$n + r = 20$

$01010010010011100010 \notin \boldsymbol{LPA}(\ell = \mathbf{9}, \boldsymbol{p} = \mathbf{4})$

$\ell = 9$

(period 3)

$n + r = 20$

$11001101001000011000 \in \boldsymbol{LPA}(\ell = \mathbf{9}, \boldsymbol{p} = \mathbf{4})$

LPA Lower Bound

- Let $a(n, \ell, p)$ be the number of words that satisfy the constraint,

$$a(n, \ell, p) \geq 2^n \cdot \left(1 - \frac{n}{2^{\ell - p}}\right)$$

- **Notice:** $\ell = \lceil \log n \rceil + p + 1 \Longrightarrow a(n, \ell, p) \geq 2^{n-1}$
- **Corollary:** there exists a code with a single bit of redundancy ($r = 1$) for $\ell = \lceil \log n \rceil + p + 1$

Y. M. Chee *et al.*, "Coding for racetrack memories," *IEEE TIT*, 2018.

Least-Period-Avoiding (LPA) Constraint

- $x \in \Sigma^{n+r}$ is ℓ-window p-least-period avoiding (LPA) if all windows do **not** have periodicity $< \boldsymbol{p}$ (where ℓ, p may depend on n)

Work	Minimal ℓ	# Redundancy Symbols	Time Complexity
Chee *et al.*	$\log n + p + 1$	1	Existence Proof

Y. M. Chee *et al.*, "Coding for racetrack memories," *IEEE TIT*, 2018.

Least-Period-Avoiding (LPA) Constraint

- $x \in \Sigma^{n+r}$ is ℓ-window p-least-period avoiding (LPA) if all windows do **not** have periodicity $< \boldsymbol{p}$ (where ℓ, p may depend on n)

Work	Minimal ℓ	# Redundancy Symbols	Time Complexity
Chee *et al.*	$\log n + p + 1$	1	Existence Proof
Sima and Bruck	$\log n + 3p - 2$	$p + 1$	$O(n^2 p \log n)$

Y. M. Chee *et al.*, “Coding for racetrack memories,” *IEEE TIT*, 2018.
J. Sima and J. Bruck, “Correcting deletions in multiple-heads racetrack memories,” *in IEEE ISIT*, 2019.

Least-Period-Avoiding (LPA) Constraint

- $x \in \Sigma^{n+r}$ is ℓ-window p-least-period avoiding (LPA) if all windows do **not** have periodicity $< \boldsymbol{p}$ (where ℓ, p may depend on n)

Work	Minimal ℓ	# Redundancy Symbols	Time Complexity
Chee *et al.*	$\log n + p + 1$	1	Existence Proof
Sima and Bruck	$\log n + 3p - 2$	$p + 1$	$O(n^2 p \log n)$
This Work	$\log n + p + 1$	1	$O(n)$ average

Y. M. Chee *et al.*, "Coding for racetrack memories," *IEEE TIT*, 2018.
J. Sima and J. Bruck, "Correcting deletions in multiple-heads racetrack memories," *in IEEE ISIT*, 2019.

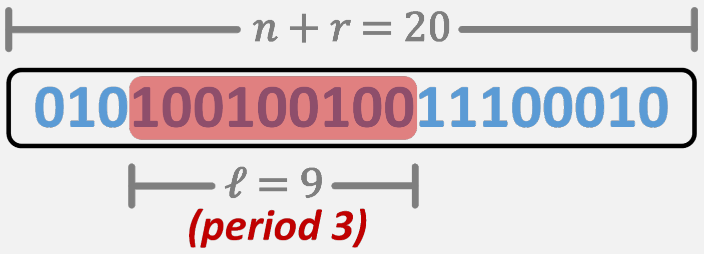
Related Works: RLL Constraint

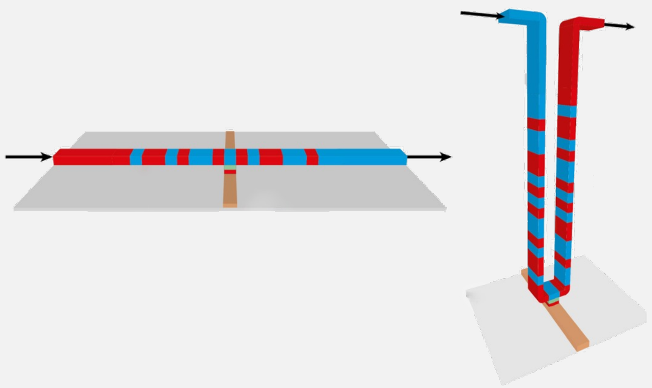
- $x \in \Sigma^{n+r}$ satisfies the k-**run-length-limited (RLL)** constraint if all k-windows are **non-zero** (where k may depend on n)
- Applicative to many other codes, such as *non-overlapping codes* (codes with disjoint non-trivial prefixes and suffixes)
- **Construction:** iteratively removes zero windows, monotonically progressing through the message

M. Levy and E. Yaakobi, "Mutually uncorrelated codes for DNA storage," *IEEE TIT*, 2019.
S. R. Blackburn, "Non-Overlapping Codes," *IEEE TIT*, 2015.

Overview

Constrained Periodicity

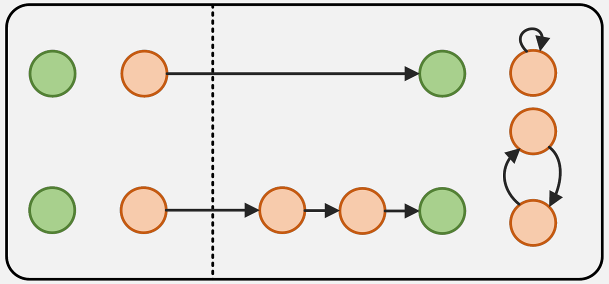




Iterative Construction

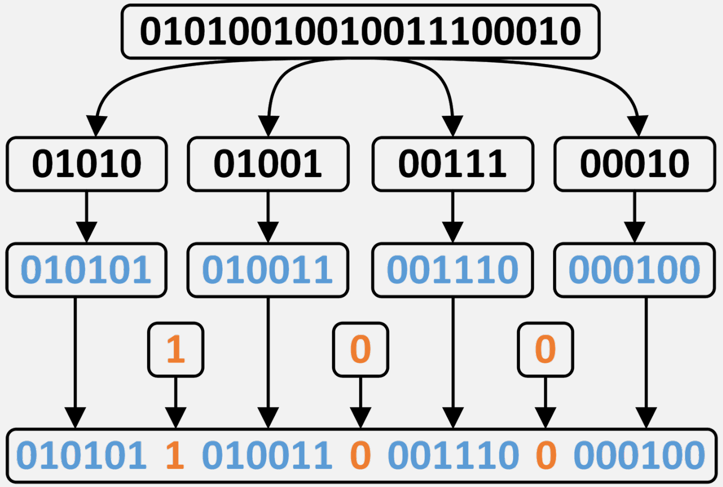
Algorithm 1 Encoder

1: **while** exists periodic window **do**
2: Fix identified window.
3: **end while**



*Near-optimal ℓ using **1** redundancy symbol*

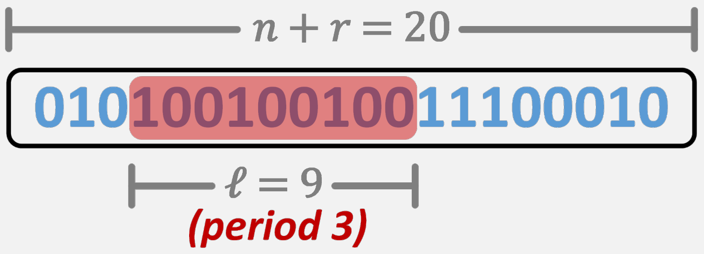
Extentions

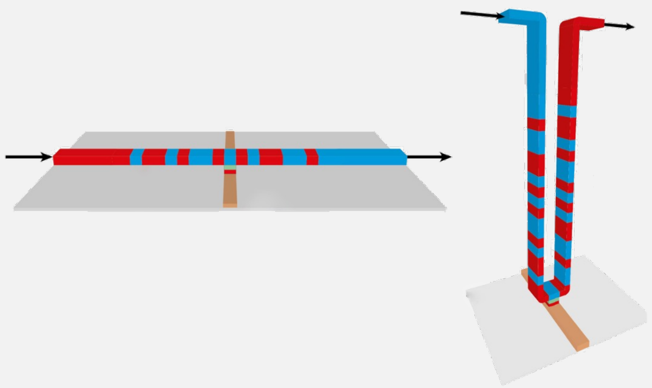


Given ℓ, use minimal redundancy symbols

Overview

Constrained Periodicity

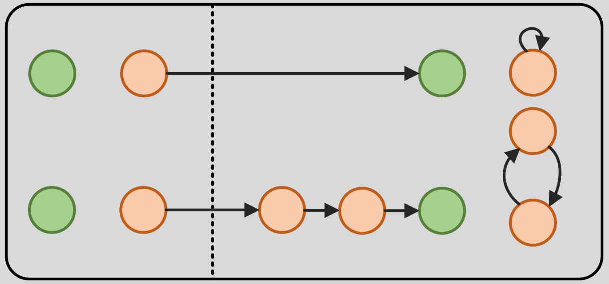




Iterative Construction

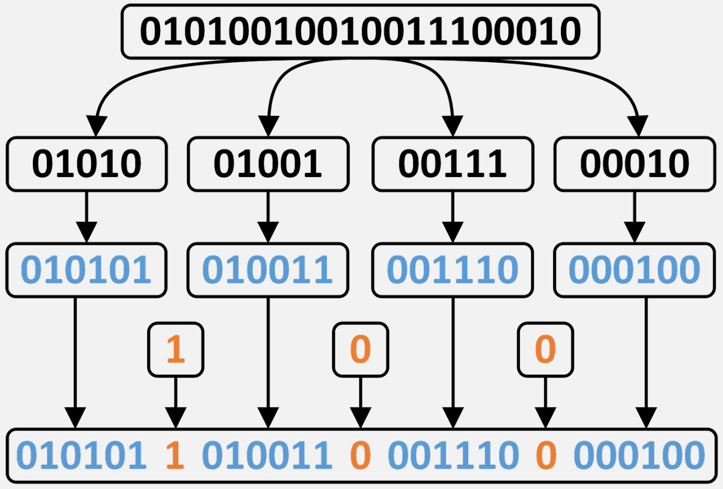
Algorithm 1 Encoder

1: **while** exists periodic window **do**
2: Fix identified window.
3: **end while**



Near-optimal ℓ using 1 redundancy symbol

Extentions



Given ℓ, use minimal redundancy symbols

Iterative Construction

$n = 14$

10001010101100

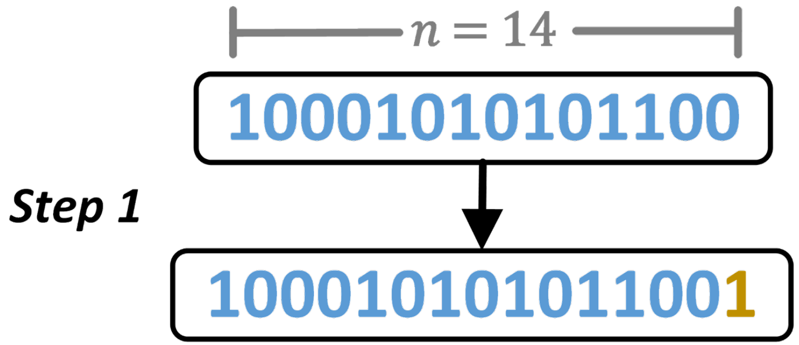
- "As long as there exists a periodic window, fix it"

Algorithm 1 Encoder

Append 1 to $\mathbf{x}$
while exists periodic window in $\mathbf{x}$ **do**
    Encode window and index, append to end
    Append 0 to $\mathbf{x}$
end while

Iterative Construction

- "As long as there exists a periodic window, fix it"



Algorithm 1 Encoder

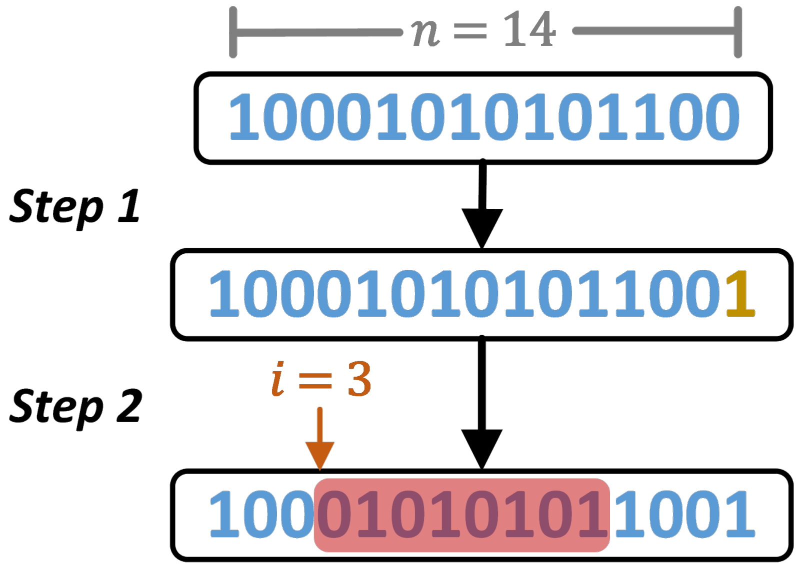
```
Append 1 to x
while exists periodic window in x do
    Encode window and index, append to end
    Append 0 to x
end while
```

Iterative Construction

- "As long as there exists a periodic window, fix it"

Algorithm 1 Encoder

Append 1 to $\mathbf{x}$
while exists periodic window in $\mathbf{x}$ **do**
    Encode window and index, append to end
    Append 0 to $\mathbf{x}$
end while

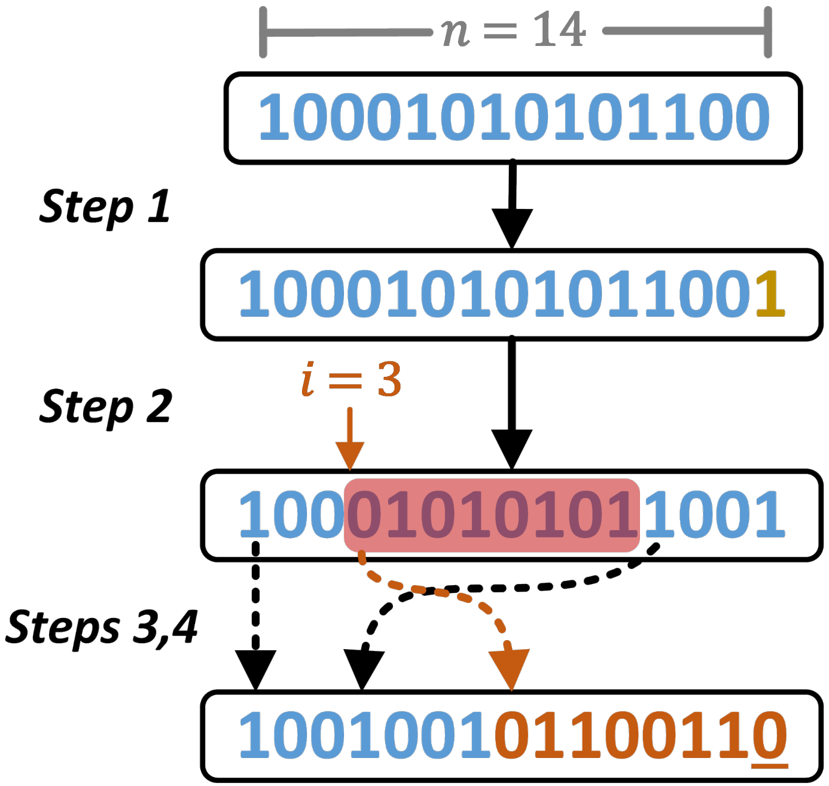


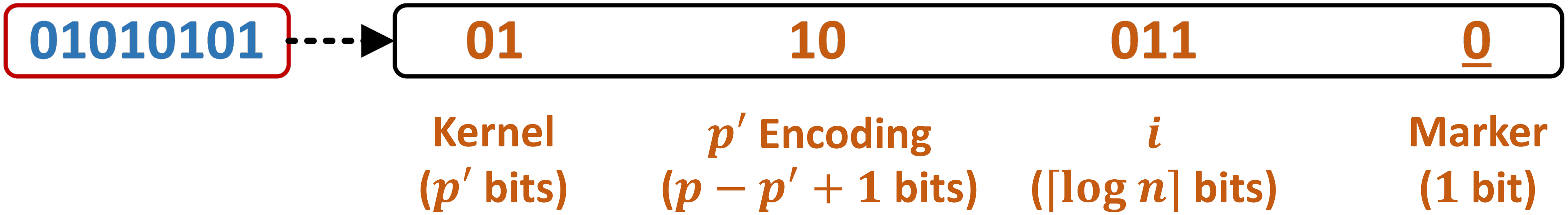
Iterative Construction

- "As long as there exists a periodic window, fix it"

Algorithm 1 Encoder

Append 1 to $\mathbf{x}$
while exists periodic window in $\mathbf{x}$ **do**
  Encode window and index, append to end
  Append 0 to $\mathbf{x}$
end while



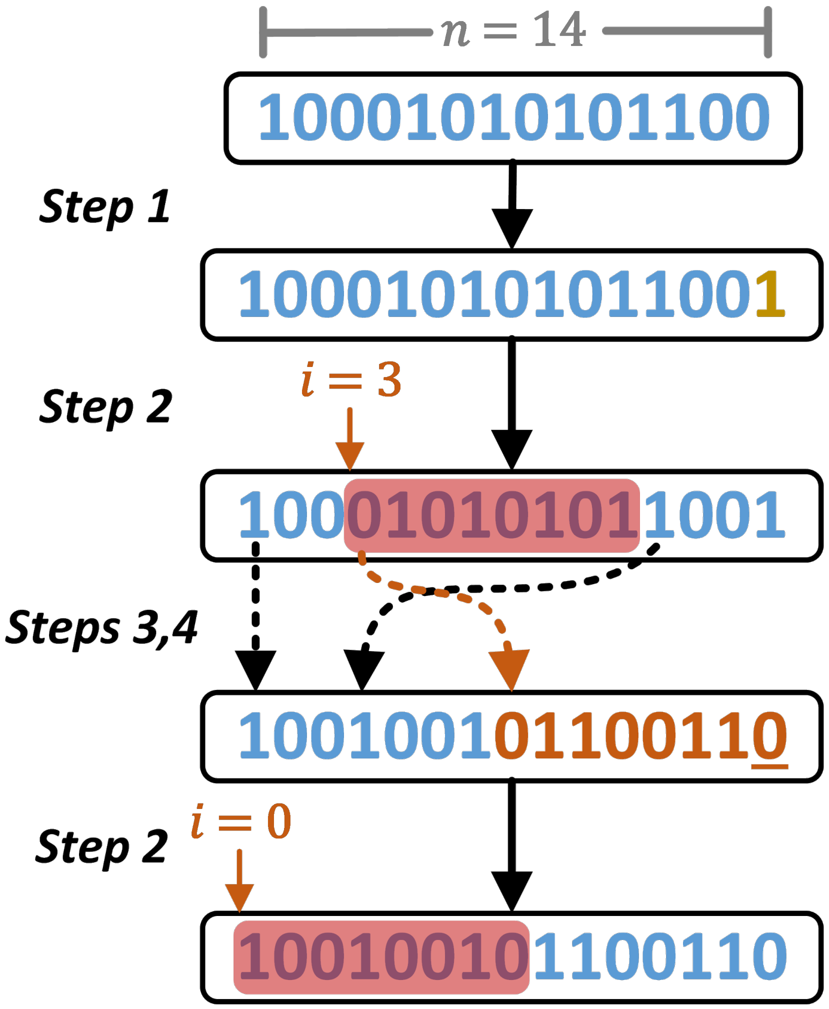


Iterative Construction

- "As long as there exists a periodic window, fix it"

Algorithm 1 Encoder

```
1: Append 1 to x
2: while exists periodic window in x do
3:     Encode window and index, append to end
4:     Append 0 to x
5: end while
```

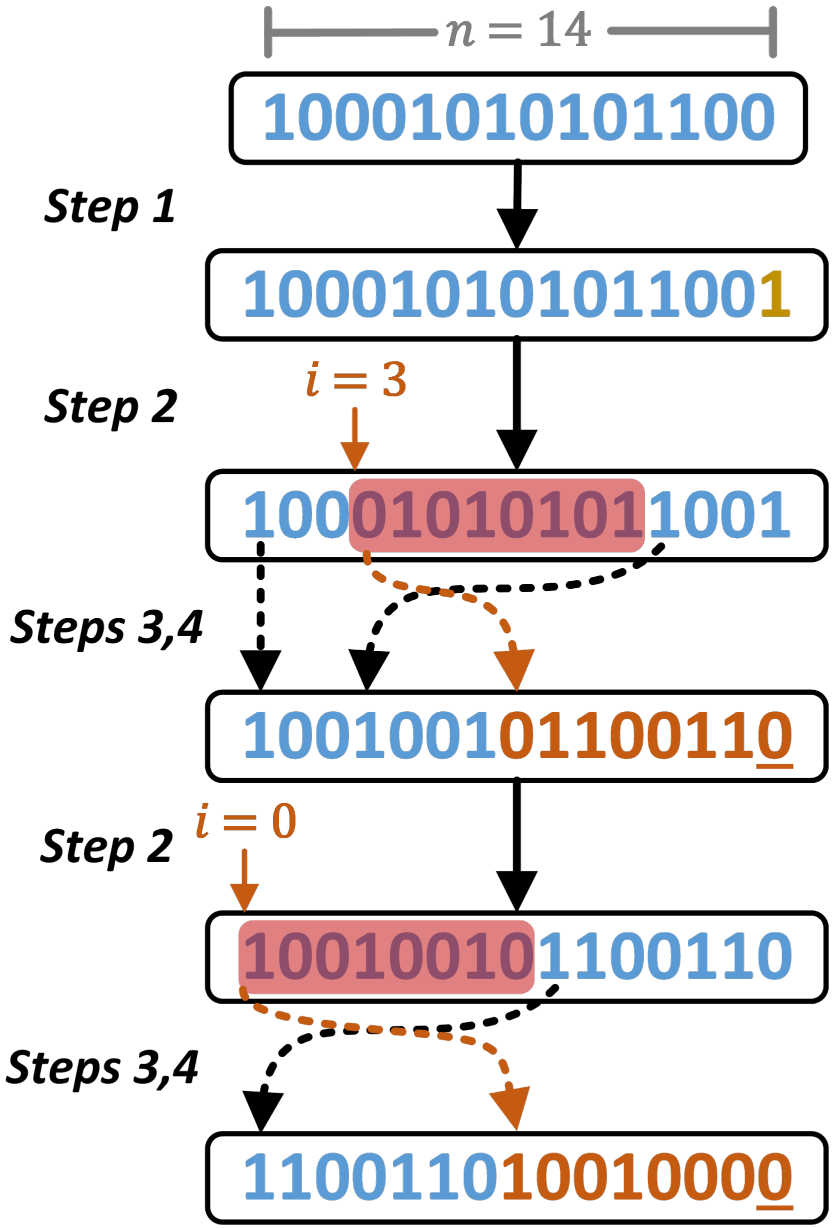


Iterative Construction

- “As long as there exists a periodic window, fix it”

Algorithm 1 Encoder

1: Append 1 to $\mathbf{x}$
2: **while** exists periodic window in $\mathbf{x}$ **do**
3: Encode window and index, append to end
4: Append 0 to $\mathbf{x}$
5: **end while**

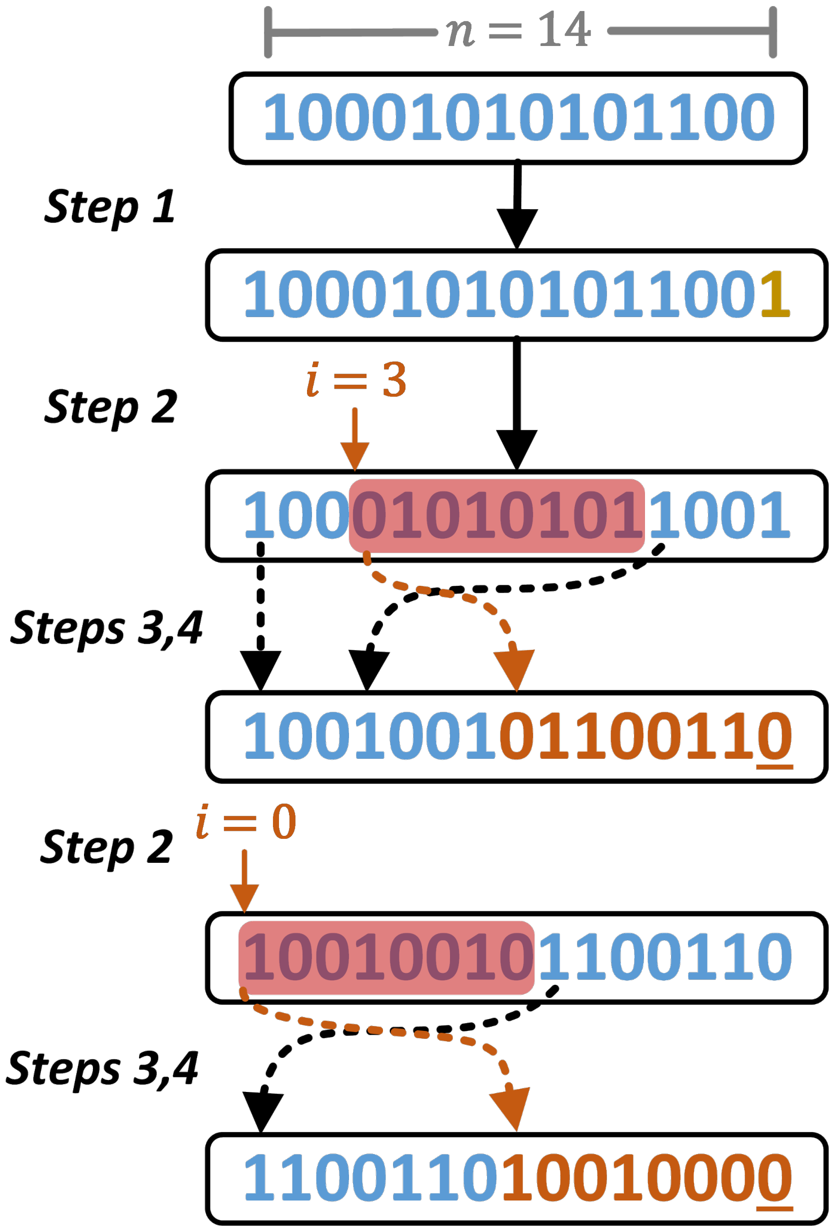


Convergence?

- Why should it converge?

Convergence?

- Why should it converge?
  - Non-monotonic progression!



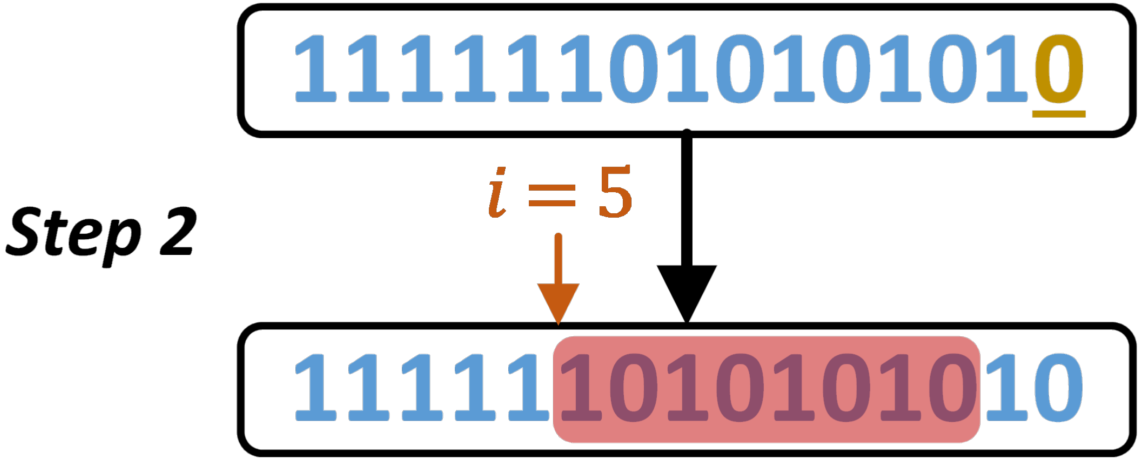
Convergence?

- Why should it converge?
  - Non-monotonic progression!
  - Self-loops!

111111010101010<u>0</u>

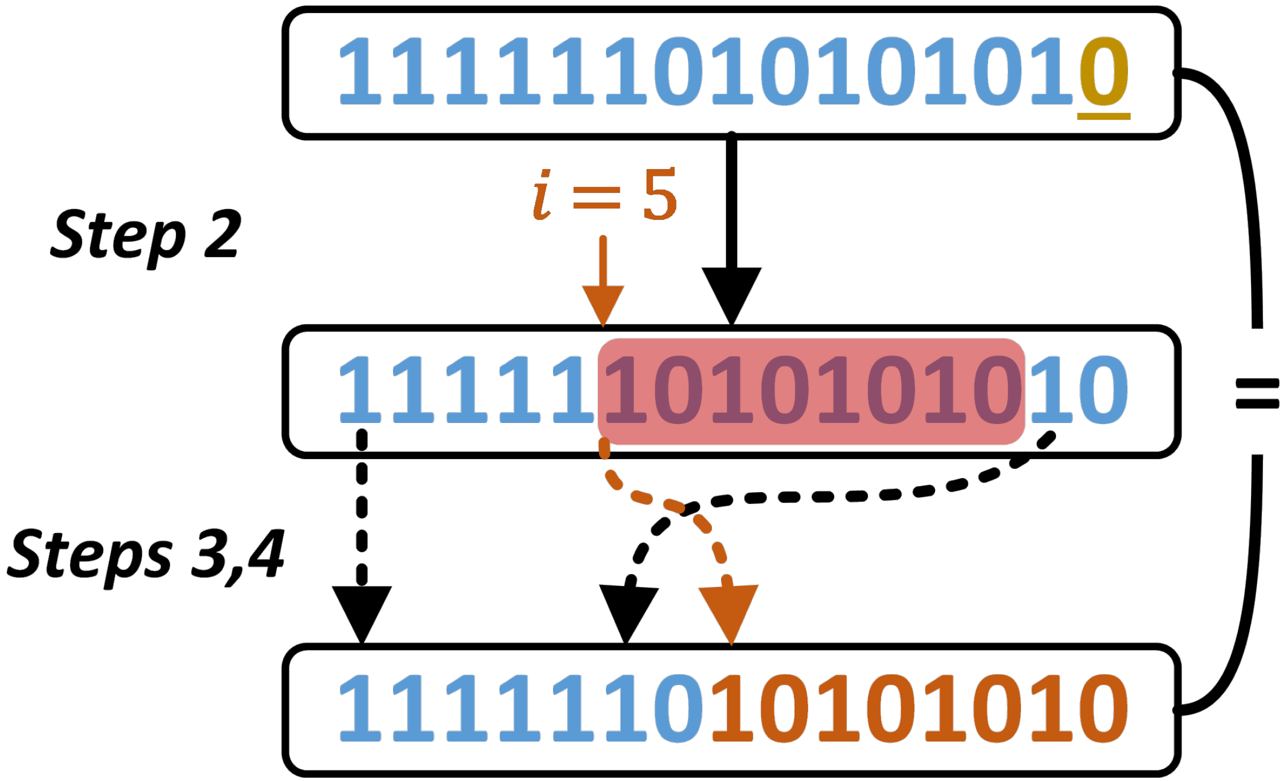
Convergence?

- Why should it converge?
  - Non-monotonic progression!
  - Self-loops!



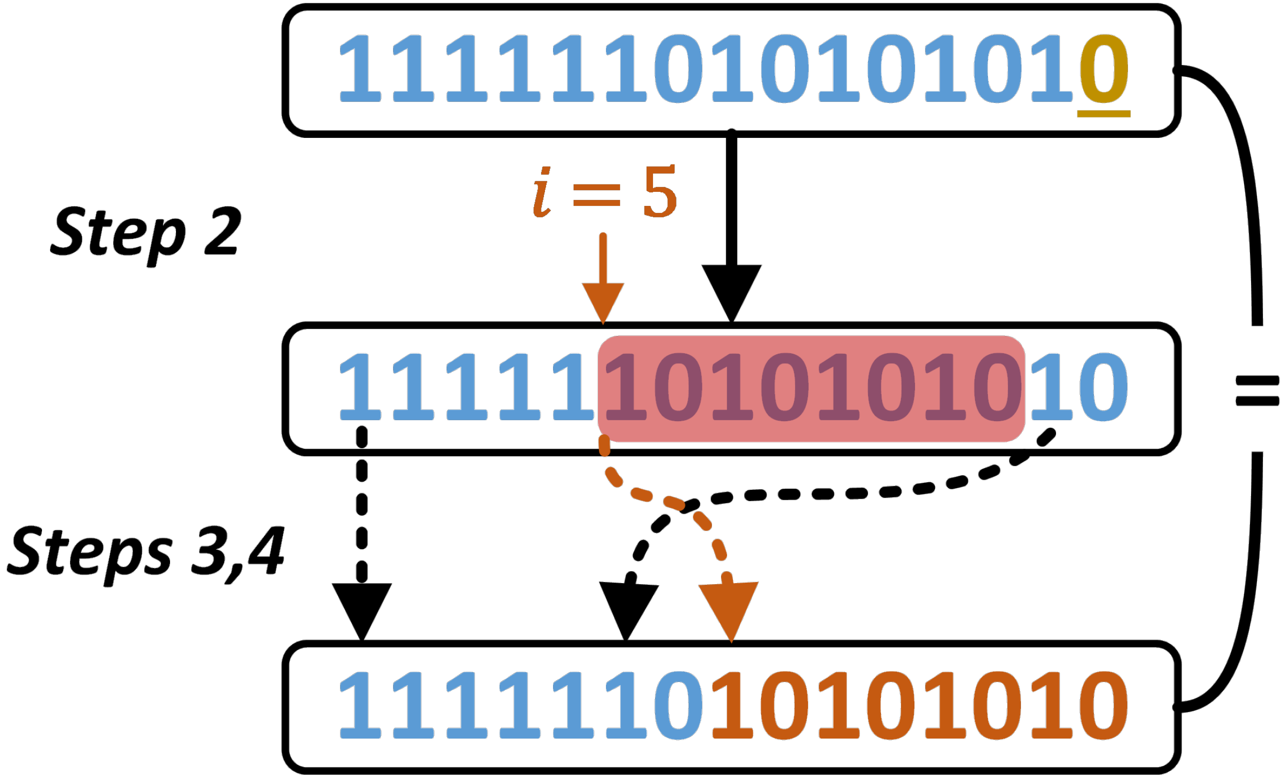
Convergence?

- Why should it converge?
  - Non-monotonic progression!
  - Self-loops!



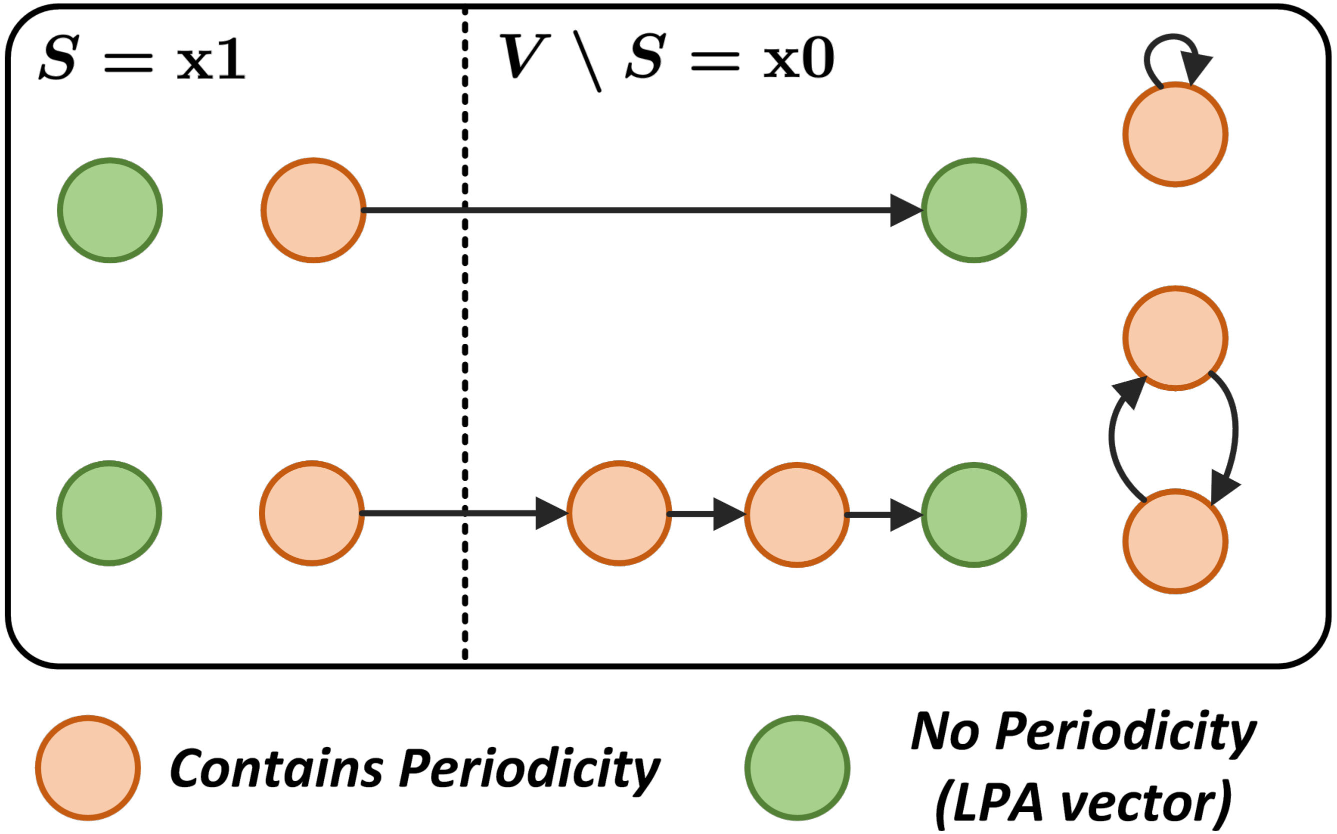
Convergence?

- Why should it converge?
  - Non-monotonic progression!
  - Self-loops!
- **Answer: invertability**



Graph Interpretation

- **Nodes:** $V = \Sigma^{n+1}$
  - $\boldsymbol{S}$: ends with 1
  - **Green:** satisfy LPA
- **Edges:** one iteration of the while loop = edge



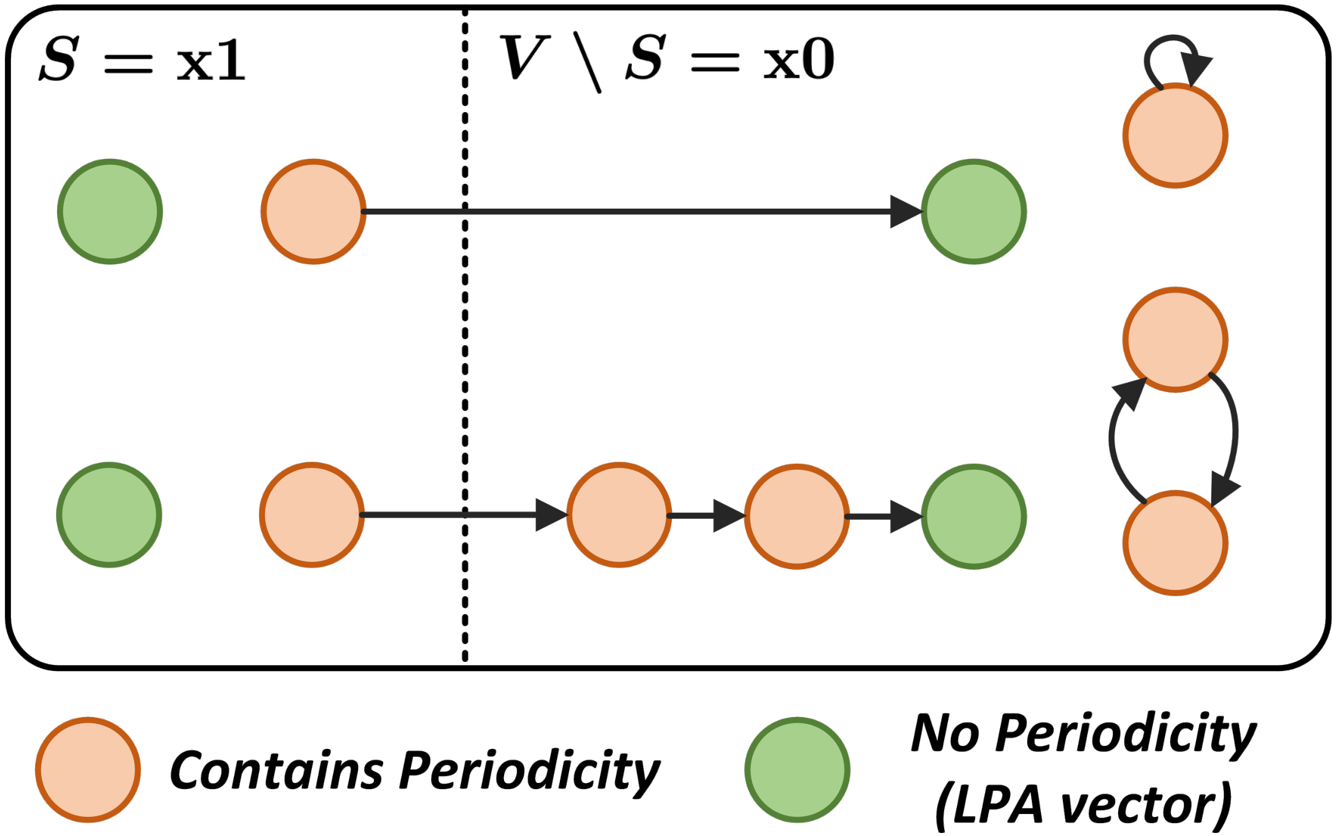
Iterative Construction – Convergence

“Starts in S and continues until no periodicity”

Algorithm 1 Encoder

```
Append 1 to x
while exists periodic window in x do
    Encode window and index, append to end
    Append 0 to x
end while
```

- **Observation 1:** In-degree of all nodes ≤ 1
- **Observation 2:** All loops must be in $V \setminus S$



Iterative Construction – Time Complexity

- Encoding requires $O(1)$ iterations on average:
  - **Observation:** paths taken by different inputs are disjoint
  - Number of nodes in graph is bounded by 2^{n+1}
  - Number of different inputs is 2^n

$$\frac{1}{2^n} \cdot \sum_{\boldsymbol{x} \in \Sigma^n} t(\boldsymbol{x}) \le \frac{1}{2^n} \cdot (2^{n+1}) = 2 = O(1)$$

- Overall, $O(n)$ average time encoding/decoding

Iterative Construction – Summary

- Attains theoretical results of Chee *et al.*

Work	Minimal ℓ	# Redundancy Symbols	Time Complexity
Chee *et al.*	$\log n + p + 1$	1	Existence Proof
Sima and Bruck	$\log n + 3p - 2$	$p + 1$	$O(n^2 p \log n)$
This Work	$\log n + p + 1$	1	$O(n)$ average

Y. M. Chee *et al.*, "Coding for racetrack memories," *IEEE TIT*, 2018.
J. Sima and J. Bruck, "Correcting deletions in multiple-heads racetrack memories," *in IEEE ISIT*, 2019.

Is the Minimal ℓ Optimal?

- Reduction from PA constraint to run-length-limited (RLL) constraint:

$$x_i \Longrightarrow x_i \oplus x_{i+p}$$

as p-periodic windows become zero runs.

- Using bound on RLL, we get an **upper** bound on the LPA constraint:

$$a(n,\ell,p) \leq 2^{n-c\cdot\frac{n-2\ell+p-1}{2^{\ell-p+1}}}, c = \frac{\log e}{8}$$

- **Corollary:** $\ell \geq \log(n) + p - 4.5$ for a single redundancy bit

M. Levy and E. Yaakobi, "Mutually uncorrelated codes for DNA storage," IEEE TIT, 2019.

Iterative Construction – Summary

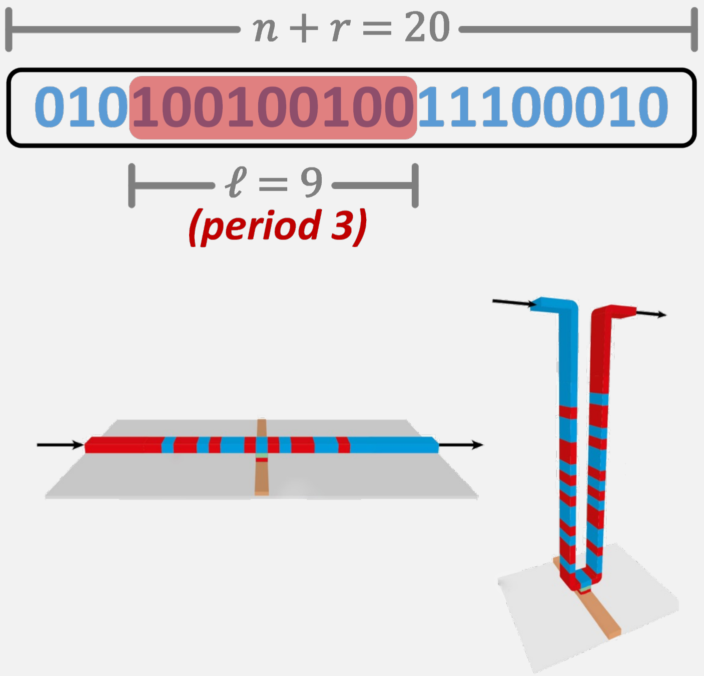
- Near-optimal ℓ for single-symbol redundancy:

Work	Minimal ℓ	# Redundancy Symbols	Time Complexity
Chee *et al.*	$\log n + p + 1$	1	Existence Proof
Sima and Bruck	$\log n + 3p - 2$	$p + 1$	$O(n^2 p \log n)$
This Work	$\log n + p + 1$	1	$O(n)$ average
Lower Bound	$\log n + p - 4.5$	1	—

Y. M. Chee *et al.*, "Coding for racetrack memories," *IEEE TIT*, 2018.
J. Sima and J. Bruck, "Correcting deletions in multiple-heads racetrack memories," *in IEEE ISIT*, 2019.

Overview

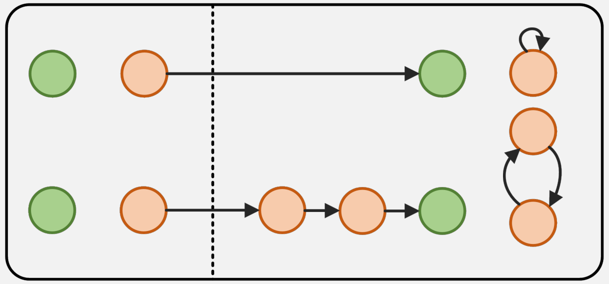
Constrained Periodicity



Iterative Construction

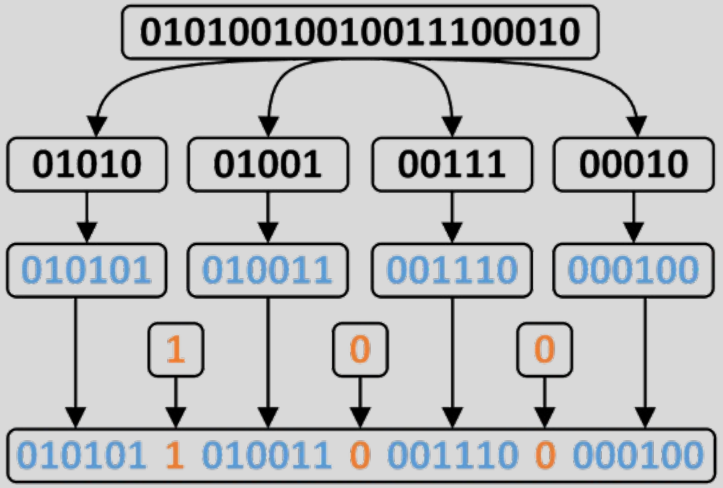
Algorithm 1 Encoder

1: **while** exists periodic window **do**
2: Fix identified window.
3: **end while**



*Near-optimal ℓ using **1** redundancy symbol*

Extentions



Given ℓ, use minimal redundancy symbols

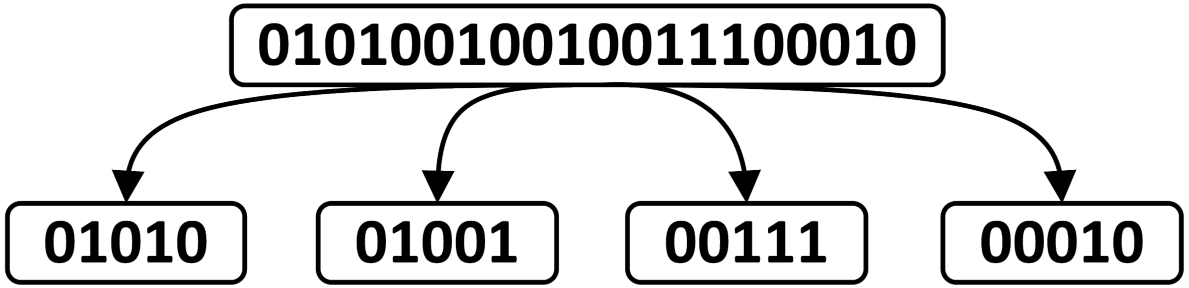
Block Constructions

- Seek to support smaller ℓ with additional redundancy
- **Solution:** split into k blocks to "reduce $\log n$ to $\log \frac{n}{k}$"

0101001001001110001 0

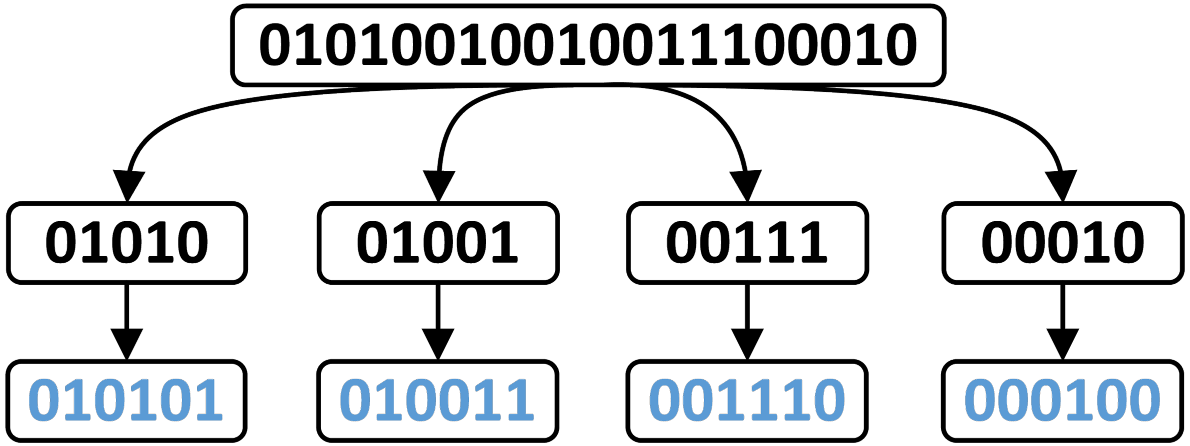
Block Constructions

- Seek to support smaller ℓ with additional redundancy
- **Solution:** split into k blocks to “reduce $\log n$ to $\log \frac{n}{k}$”



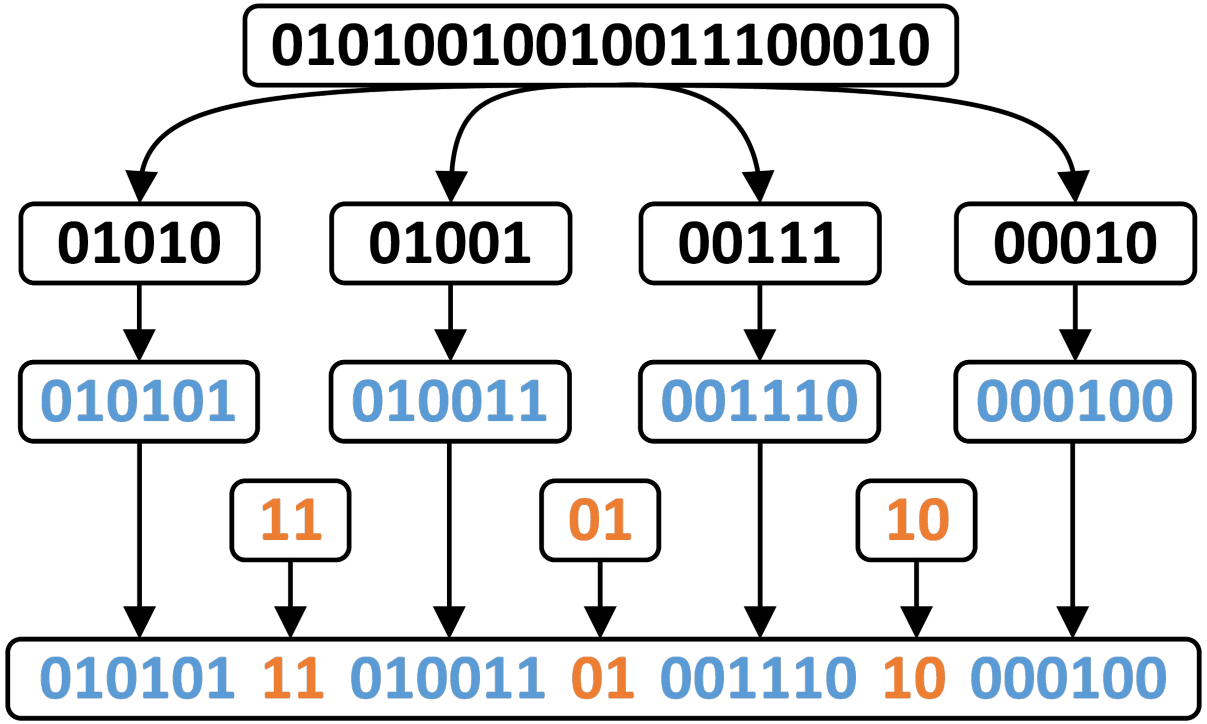
Block Constructions

- Seek to support smaller ℓ with additional redundancy
- **Solution:** split into k blocks to "reduce $\log n$ to $\log \frac{n}{k}$"



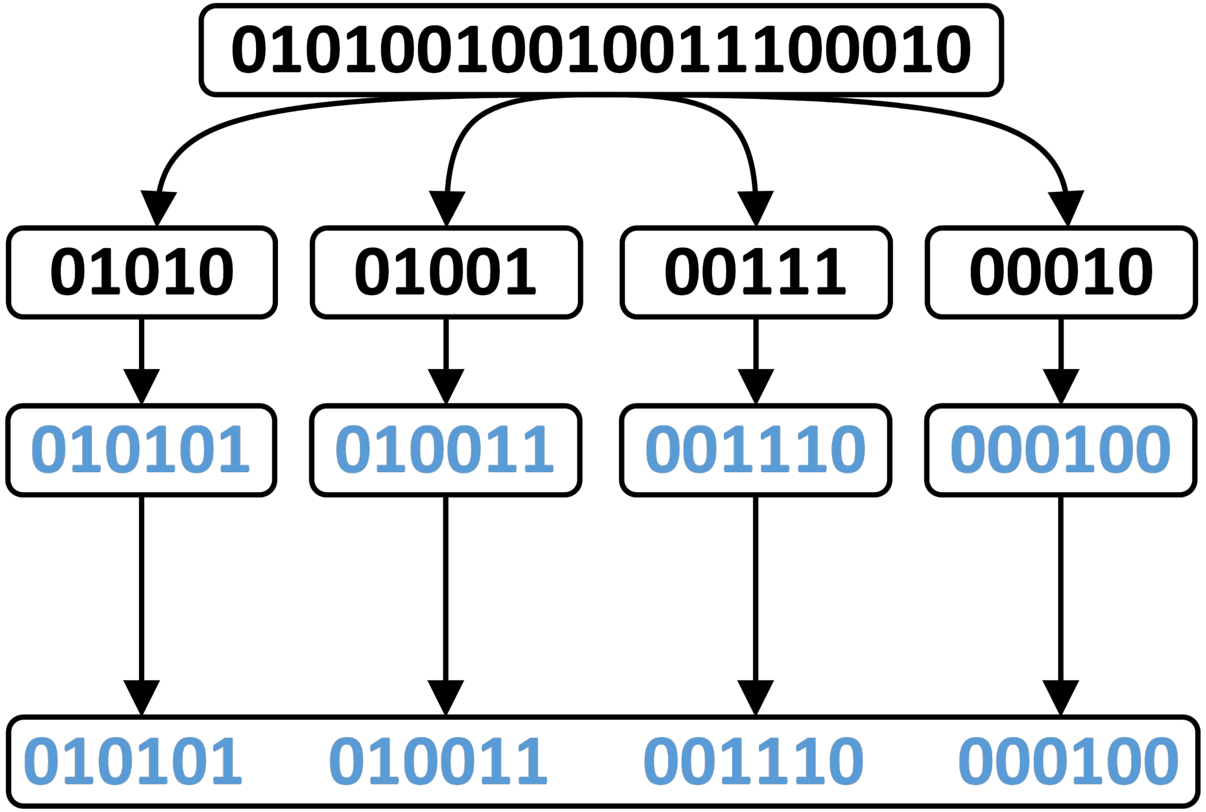
Block Constructions

- Seek to support smaller ℓ with additional redundancy
- **Solution:** split into k blocks to "reduce $\log n$ to $\log \frac{n}{k}$"



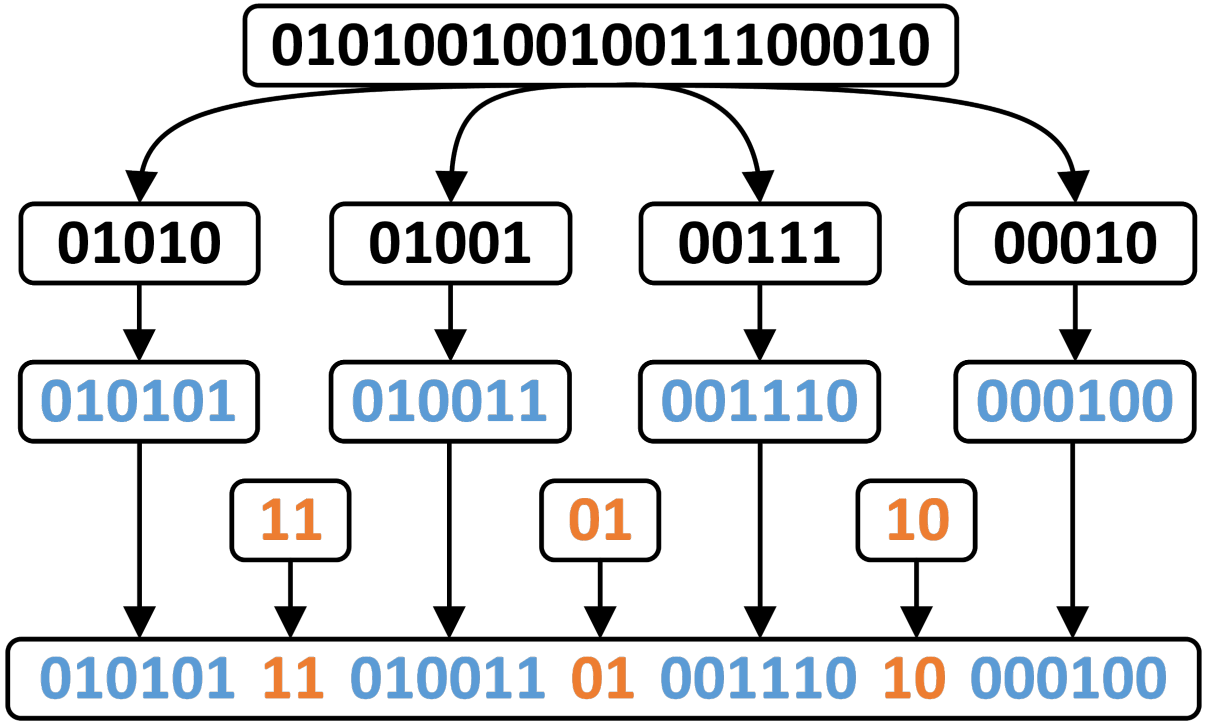
Block Construction 1

- **Observation:** concatenation of ℓ-LPA codes is a 2ℓ-LPA code
  - Any 2ℓ window contains at least ℓ bits from a single block
  - Substring of periodic string is also periodic
- **Construction:** concat. blocks of the iterative construction
- **Redundancy:** k bits



Block Construction 2

- **Lemma:** for any $\boldsymbol{s} \in \Sigma^{\ell}$, there exists $a \in \Sigma$ such that $\boldsymbol{s}a$ contanis no periods $\leq \lfloor \ell/2 \rfloor + 2$
- **Construction:** Concatenate blocks with 2 bits in between
- **Redundancy:** $3k - 2$ bits



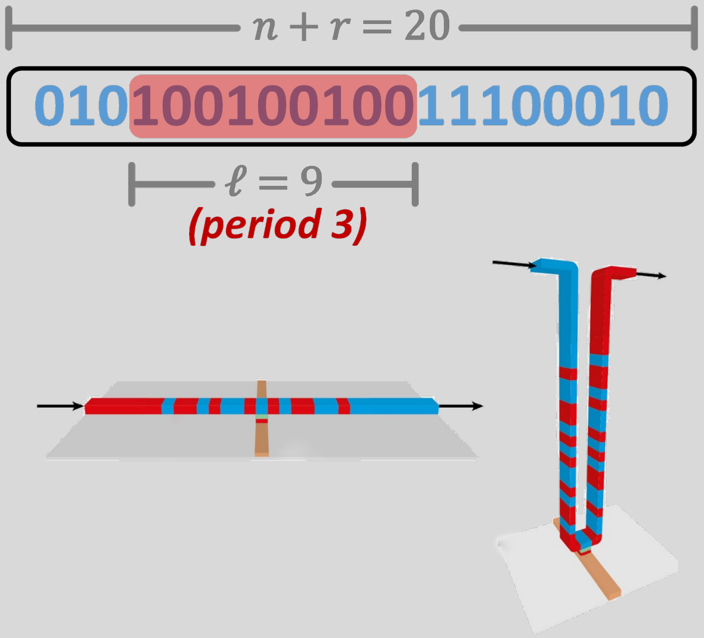
Block Constructions – Results

Work	Minimal ℓ	# Redundancy Symbols	Time Complexity
Chee et al.	$\log n + p + 1$	1	Existence Proof
Sima and Bruck	$\log n + 3p - 2$	$p + 1$	$O(n^2 p \log n)$
This Work	$\log n + p + 1$	1	$O(n)$ average
Lower Bound	$\log n + p - 4.5$	1	—
This Work	$2\left(\log\left(\frac{n}{k}\right) + p + 1\right)$	k	$O(n)$ average
This Work	$\log\left(\frac{n}{k}\right) + p + 1$	$3k - 2$	$O(n)$ average

Y. M. Chee *et al.*, "Coding for racetrack memories," *IEEE TIT*, 2018.
J. Sima and J. Bruck, "Correcting deletions in multiple-heads racetrack memories," *in IEEE ISIT*, 2019.

Periodicity – Conclusion

Constrained Periodicity

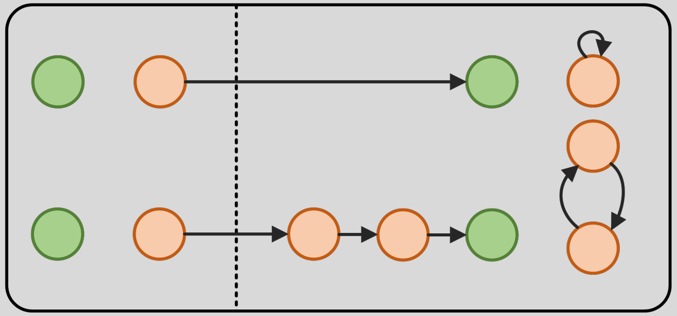


Iterative Construction

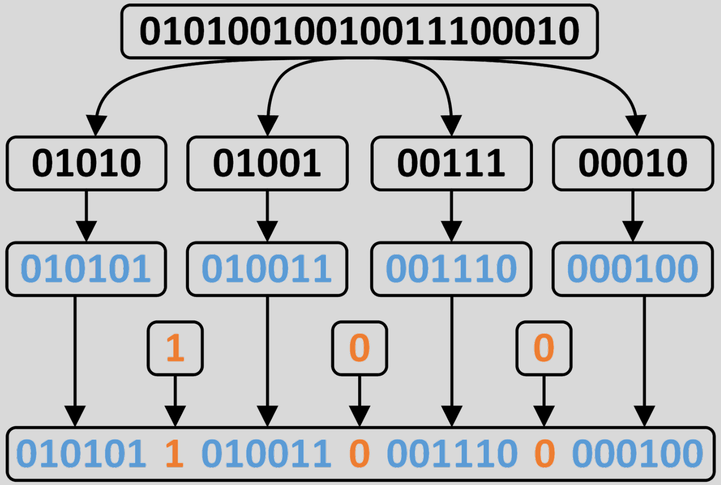
Near-Optimal Alg:

Algorithm 1 Encoder

1: **while** exists periodic window **do**
2: Fix identified window.
3: **end while**



Extentions

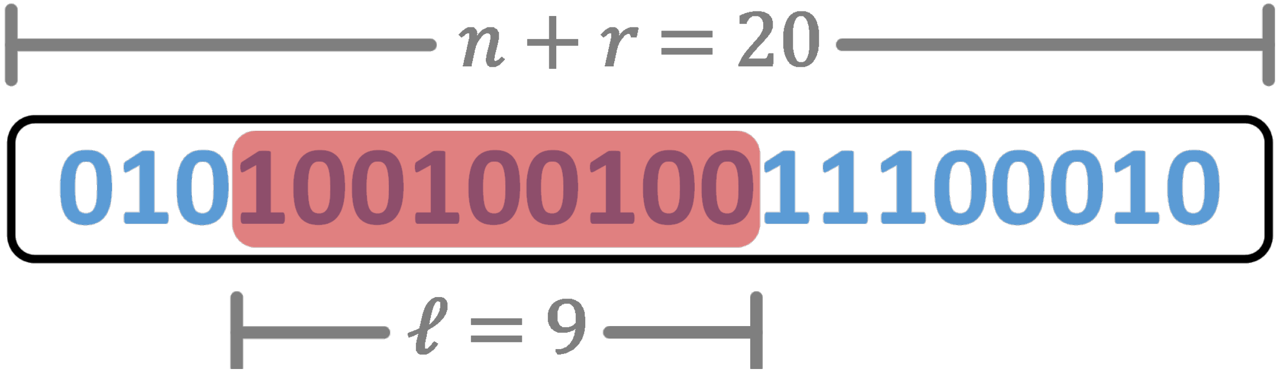


Can we generalize beyond periodicity?

Parametric Constraint

Problem:

- *Input:* $x \in \Sigma^n$
- *Output:* $y \in \Sigma^{n+r}$ **without** windows <u>that satisfy some parametric property</u> ($r = \#\, redundancy\ symbols$)



Constraint-Avoiding (CA) Vector

- $x \in \Sigma^{n+r}$ is an ℓ-window $\boldsymbol{S}$-avoiding vector (CA) if all windows are **not** in $\boldsymbol{S}$ (where ℓ, S may depend on n)

$n + r = 20$

$$\underbrace{010\boxed{100100100}11100010}_{} \notin \boldsymbol{CA}(\ell = 9, \boldsymbol{S} = \{100100100\})$$

$\ell = 9$

$n + r = 20$

$$11001101001000011000 \in \boldsymbol{CA}(\ell = 9, \boldsymbol{S} = \{100100100\})$$

CA Lower Bound

- Let $a(n, \ell, S)$ be the number of words that satisfy the constraint,

$$a(n, \ell, S) \geq 2^n \cdot \left(1 - \frac{n}{2^{\ell - \log|S|}}\right)$$

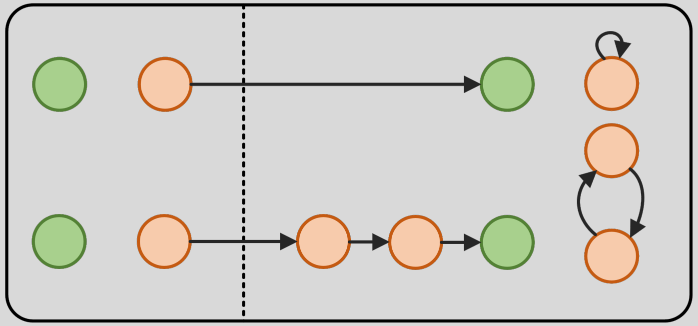
- **Notice:** $|S| \leq 2^{\ell} \cdot \frac{1}{2n} \Longrightarrow a(n, \ell, S) \geq 2^{n-1}$
- **Corollary:** there exists a code with a single bit of redundancy ($r = 1$) for $|S| \leq 2^{\ell} \cdot \frac{1}{2n}$

CA Iterative Construction

- Given **injective** function $\phi: S \to \Sigma^{\ell'}$ such that $\ell' = \ell - \lceil \log n \rceil - 1$, we get a construction with:
  - 1 redundancy symbol
  - $O(n \cdot f(\ell))$ time for $f(\ell)$ the time complexity of ϕ
- For any S with $|S| \leq 2^{\ell} \cdot \frac{1}{2n}$, there always exists a trivial $\phi: S \to \Sigma^{\ell'}$:
  - 1 redundancy symbol
  - $O(\ell \cdot \log|S|)$ time and $O(S)$ space (using binary search)

Future Work

Worst-Case Complexity



Universal Constrained Code

$x \in \Sigma^{n+r}$ is $\ell(n)$-window $S(n)$-avoiding if all windows do **not** belong to $S(n)$

Thanks!

European Research Council

The research was Funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or European Research Council.

TECHNION Israel Institute of Technology