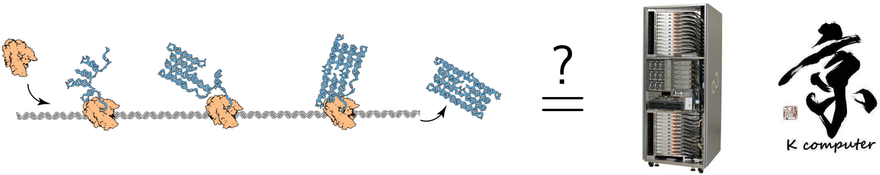
Single-stranded architectures for computing

Shinnosuke Seki

Algorithmic "Oritatami" Self-assembly Lab @ UEC Tokyo, Japan
LIP, École Normale Supérieure de Lyon, France

DLT2019, Warsaw, August 5-9

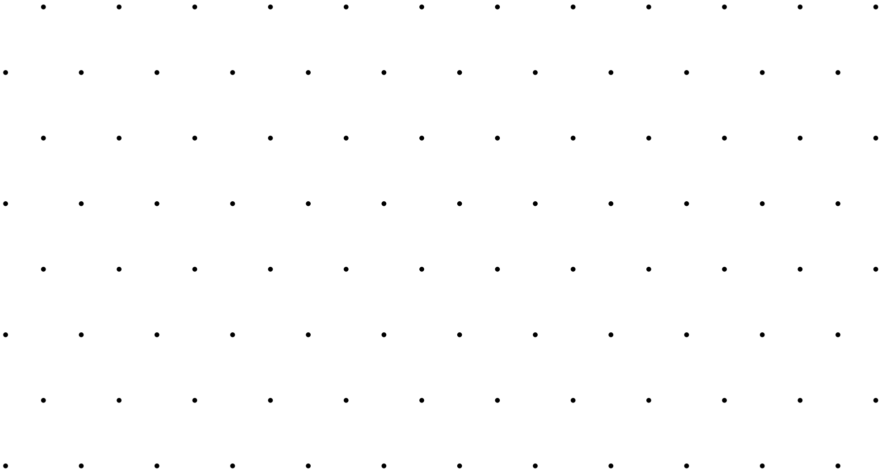


Fiat moleculis

~~Fiat moleculis~~

Oritatami [Geary, Meunier, Schabanel, S. 2016]

Playground and conformation



Oritatami [Geary, Meunier, Schabanel, S. 2016]

Playground and conformation

Conformation

is a tuple (P, w, H) of

- P Non-self-crossing directed path
- w String (transcript) as long as P
- H Set of (hydrogen) bonds

Conformation

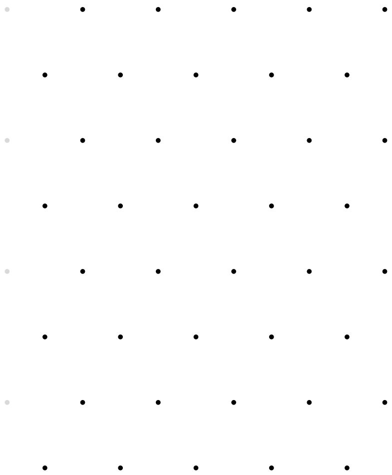
is a tuple (P, w, H) of

- P Non-self-crossing directed path
- w String (transcript) as long as P
- H Set of (hydrogen) bonds

Example

Let $\Sigma = \{a, a', b, b', \bullet\}$ and
$R = \{(a, a'), (b, b')\}$.

- P as shown right
- w $a \bullet b' b \bullet a' \bullet \bullet b' \bullet b' b$
- H $\{(0,5), (3,8), (2,11)\}$



Oritatami [Geary, Meunier, Schabanel, S. 2016]

Playground and conformation

Conformation

is a tuple (P, w, H) of

- P Non-self-crossing directed path
- w String (transcript) as long as P
- H Set of (hydrogen) bonds

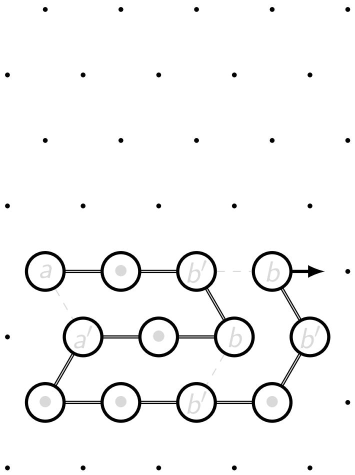
Example

Let $\Sigma = \{a, a', b, b', \bullet\}$ and
$R = \{(a, a'), (b, b')\}$.

P as shown right

w $a \bullet b' b \bullet a' \bullet \bullet b' \bullet b' b$

H $\{(0,5), (3,8), (2,11)\}$



Conformation

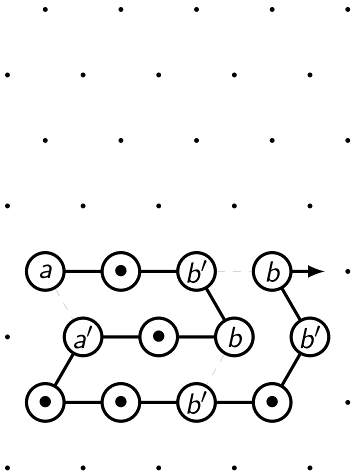
is a tuple (P, w, H) of

- P Non-self-crossing directed path
- w String (transcript) as long as P
- H Set of (hydrogen) bonds

Example

Let $\Sigma = \{a, a', b, b', \bullet\}$ and
$R = \{(a, a'), (b, b')\}$.

- P as shown right
- w $a \bullet b' b \bullet a' \bullet \bullet b' \bullet b' b$
- H $\{(0,5), (3,8), (2,11)\}$



Conformation

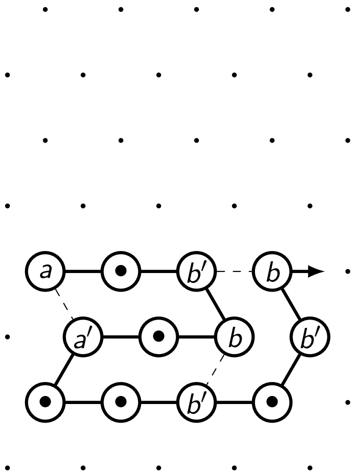
is a tuple (P, w, H) of

- P Non-self-crossing directed path
- w String (transcript) as long as P
- H Set of (hydrogen) bonds

Example

Let $\Sigma = \{a, a', b, b', \bullet\}$ and
$R = \{(a, a'), (b, b')\}$.

- P as shown right
- w $a \bullet b' b \bullet a' \bullet \bullet b' \bullet b' b$
- H $\{(0, 5), (3, 8), (2, 11)\}$



Oritatami [Geary, Meunier, Schabanel, S. 2016]

Definition

An *oritatami system* is a tuple $\Xi = (\Sigma, w, R, \delta, \alpha)$, where

- Σ finite alphabet of bead (abstract molecule) types
- $w \in \Sigma^* \cup \Sigma^\omega$ *transcript*, a string to be folded
- $R \subseteq \Sigma \times \Sigma$ *rule set* to specify which types of beads can interact
- δ *delay* (transcription rate)
- α *arity*, max # of bonds/bead

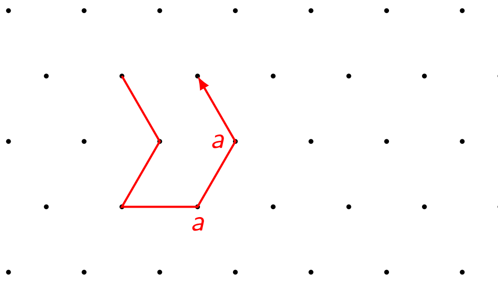
An input (seed) to Ξ is a finite conformation σ.

Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a'a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }

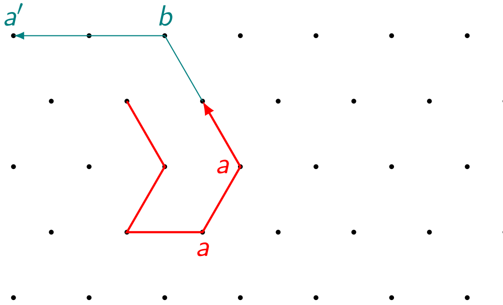


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for ($i = 1$, $i \leq |w|$, $i = i + 1$) {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }

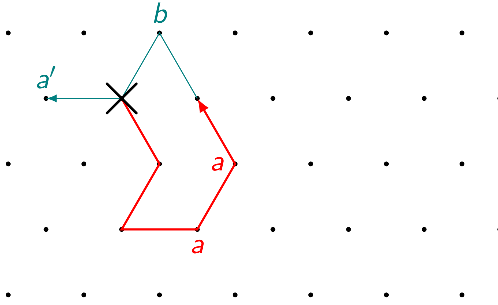


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }



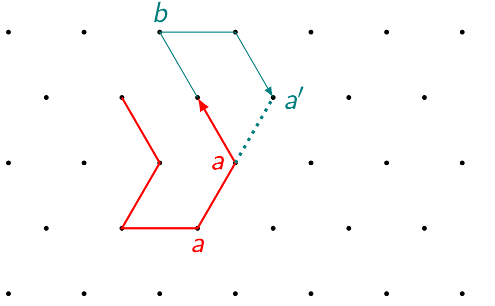
Already occupied!!

Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a'a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \le |w|,\ i = i + 1)$ {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }



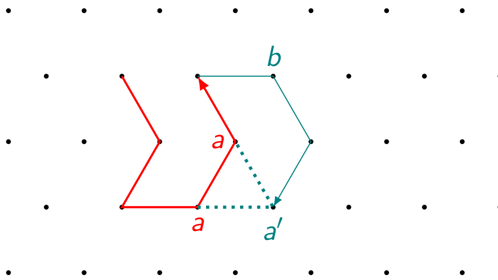
1 bond against 2 bonds for stabilization

Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }



1 bond against 2 bonds for stabilization

Oritatami [Geary, Meunier, Schabanel, S. 2016]

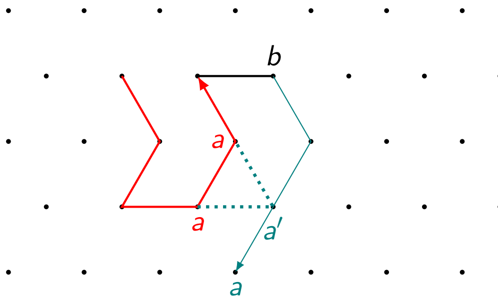
Dynamics

Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }



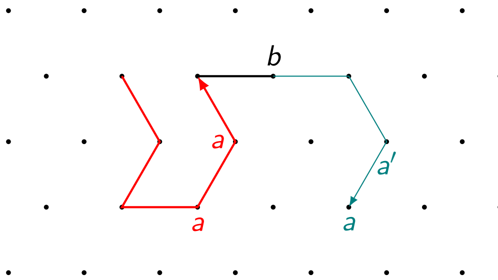
$w[1] = b$ is fixed accordingly and

Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }

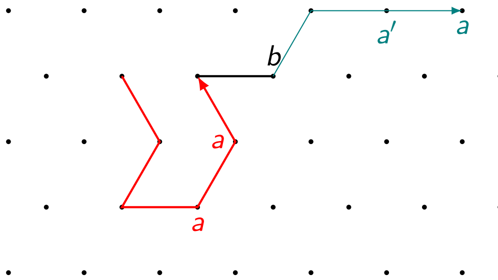


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }

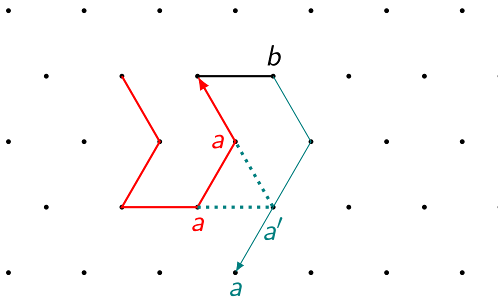


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a'a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }

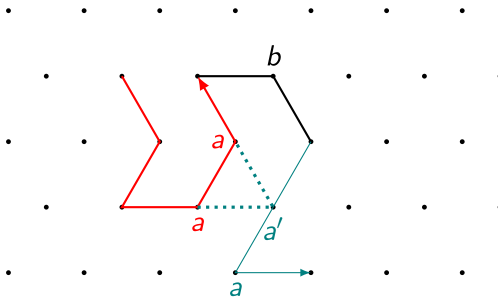


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }

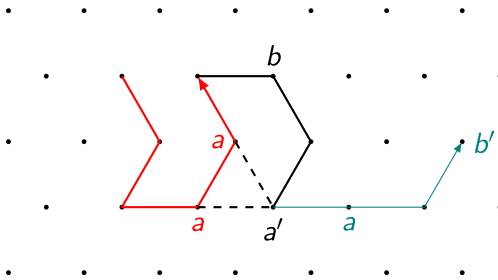


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a'a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }

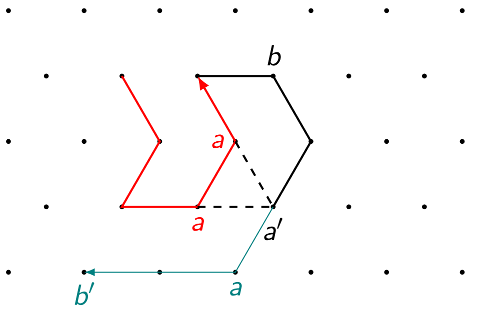


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }

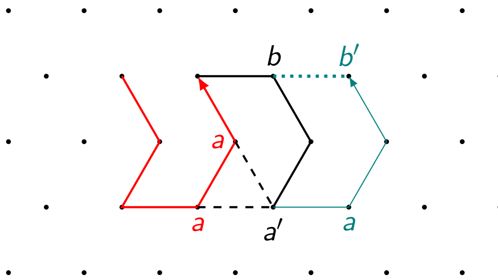


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a'a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }



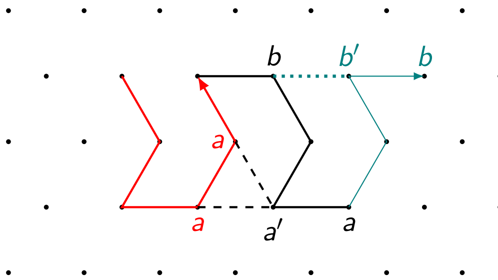
This time 1 bond is enough for stabilization

Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a'a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }

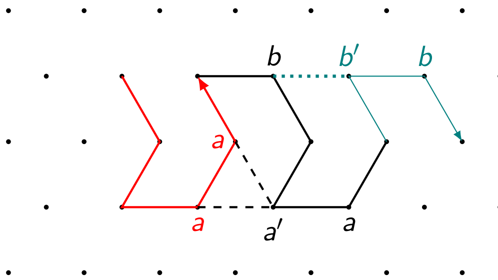


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }

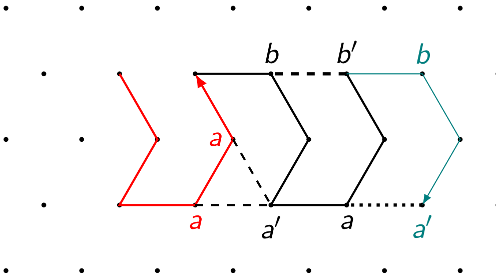


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }

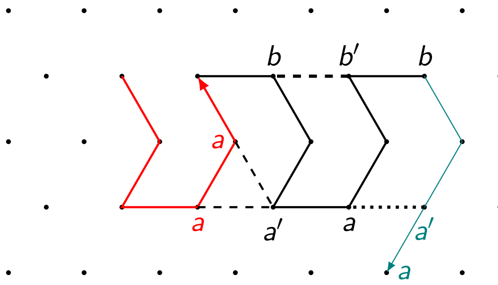


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a'a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }

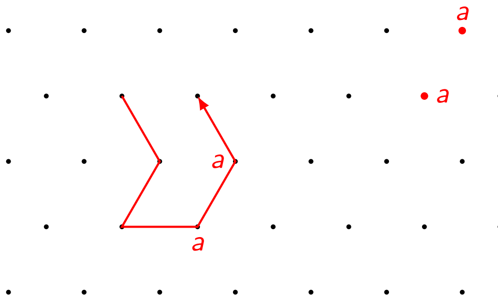


Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for ($i = 1$, $i \leq |w|$, $i = i + 1$) {
  1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
  2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
  3. Transcribe $w[i + \delta]$.

  }



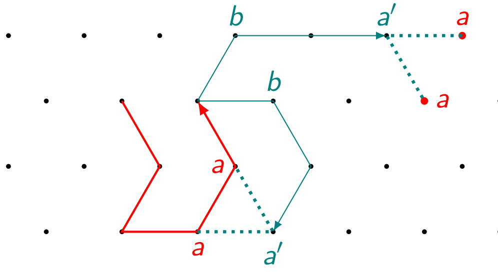
Another a-bead around causes ...

Example (Glider)

Consider an oritatami system with transcript $w = (b \bullet a' a \bullet b')^*$, rule set $R = \{(a, a'), (b, b')\}$, delay $\delta = 3$, and the seed below colored in red.

- Transcribe $w[1..\delta]$ at the 3'-end of the seed.
- for $(i = 1,\ i \leq |w|,\ i = i + 1)$ {
    1. Fold $w[i..i + \delta - 1]$ in all geometrically-possible manners.
    2. Stabilize $w[i]$ so as for $w[i..i + \delta - 1]$ to form as many bonds as possible.
    3. Transcribe $w[i + \delta]$.

  }



Another a-bead around causes ...

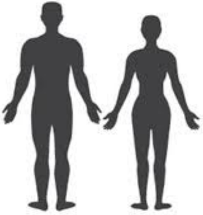
Theorem [Geary, Meunier, Schabanel, S. 2018]

There are a fixed set Σ of 542 bead types, a rule set R, and the following poly-time encodings:

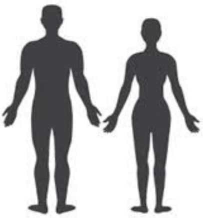
- a single-tape Turing machine $M \to$ a string $w_M \in \Sigma^*$
- M and its input $x \to$ a seed $\sigma_M(x)$ of size linear in $|x|$ and polynomial in $|M|$

with which at delay 3, starting from the seed $\sigma_M(x)$, the oritatami system stops folding w_M after $O_M(t^4 \log^2 t)$ beads iff M halts on x after t steps.

Fiat moleculis



Et creavit Deus self-assembly hominem ad imaginem suam from molecules.



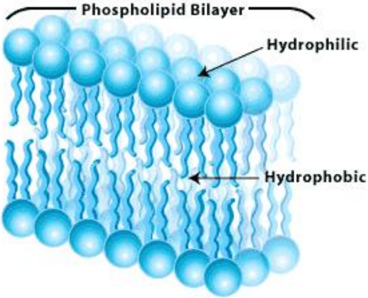
Et creavit Deus self-assembly hominem ad imaginem suam from molecules.

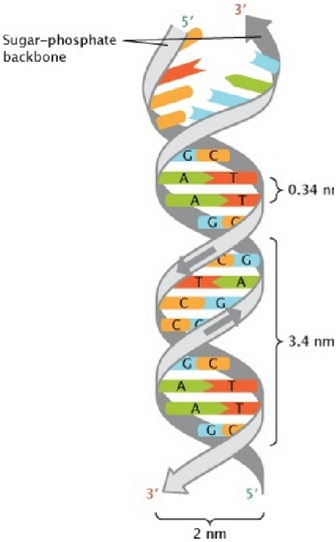
Self-assembly

Simple components (e.g., molecules)

- interact locally
- with little/no external control

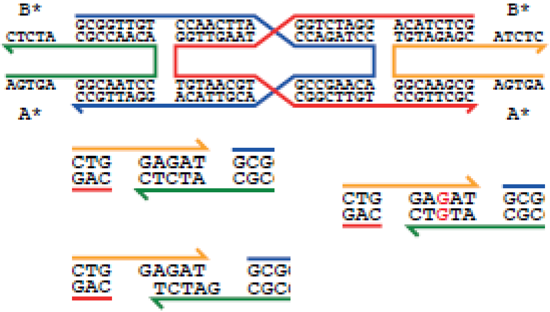
into sth. complex/sophisticated.





Molecular self-assembly

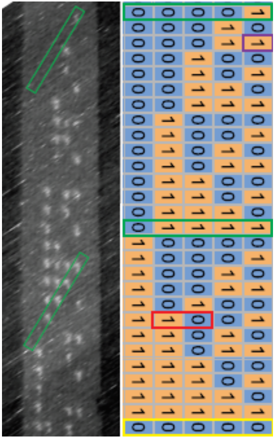
Interactive programmable DNA tiles [Evans 2014, Winfree+3 1998]



Sequences of 4 sticky ends (single-stranded parts, 2 green, 2 yellow) dominate how this tile interacts with other tiles.

Interactive programmable DNA tiles [Evans 2014, Winfree+3 1998]

Binary counting *in vitro* using a set of DNA tile types implementing half-adder. In principle, just 4 tile types suffice.



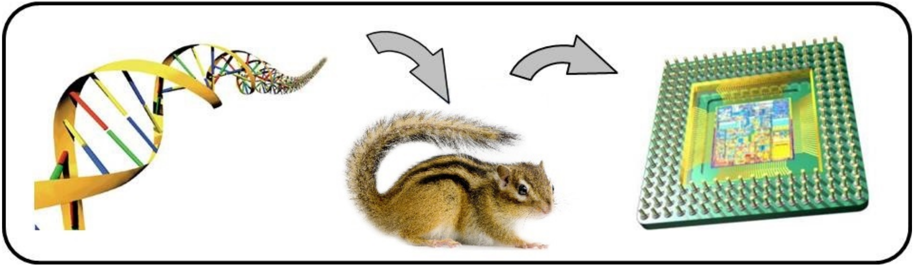
artwork by [Evans 2014]

Molecular self-assembly

In-vitro self-assembly (engineering goal so far)

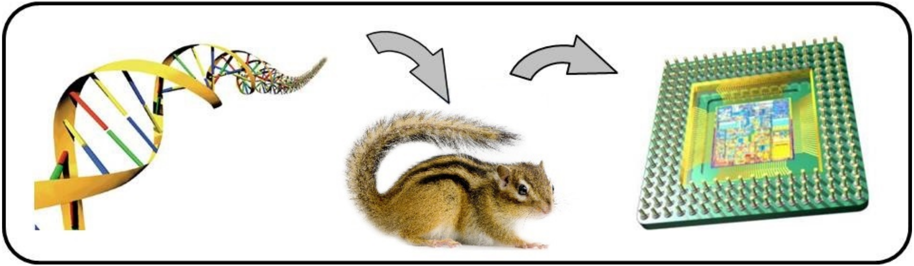
Molecular self-assembly

In-vivo self-assembly (next step)



Molecular self-assembly

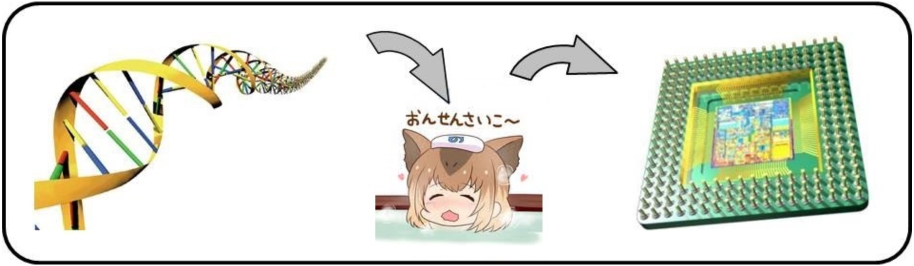
In-vivo self-assembly (next step)



Thermal control is necessary in order to proceed self-assembly of DNA, which is quite stable, that is, ...

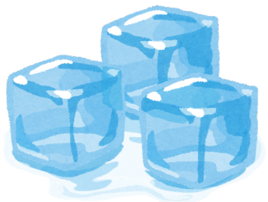
Molecular self-assembly

In-vivo self-assembly (next step)



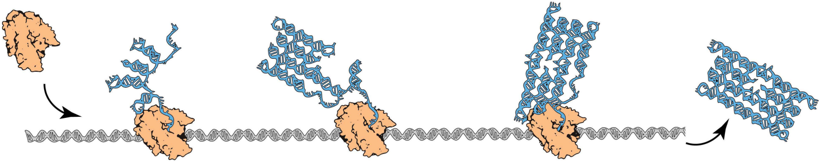
Molecular self-assembly

In-vivo self-assembly (next step)



One solution

Cotranscriptional folding



A copy (RNA transcript) of a DNA sequence folds upon itself while being transcribed (synthesized), that is, cotranscriptionally.

This process is isothermal.

Cotranscriptional folding

RNA origami (*in-vitro* self-assembly of RNA tile by CF)[a]

[a] Geary, Rothemund, Andersen, *Science* 345:798-802, 2014

$$\text{Specific RNA tile } T \xrightarrow{\text{program}} \text{a DNA sequence} \xrightarrow{\text{CF}} T$$

Cotranscriptional folding

RNA origami (*in-vitro* self-assembly of RNA tile by CF)[a]

[a] Geary, Rothemund, Andersen, *Science* 345:798-802, 2014

$$\text{Specific RNA tile } T \xrightarrow{\text{program but how?}} \text{a DNA sequence} \xrightarrow{\text{CF}} T$$

RNA polymerase: Loss-less (1-to-1) synthesis from DNA to RNA

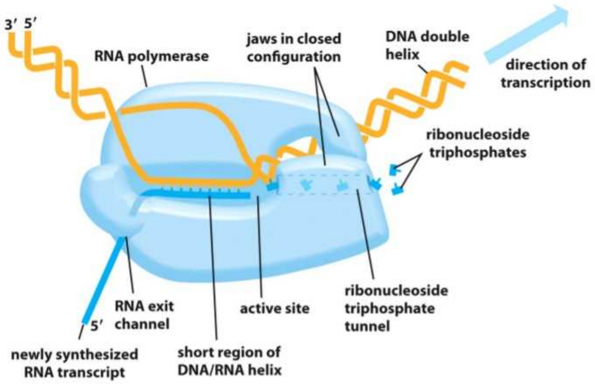
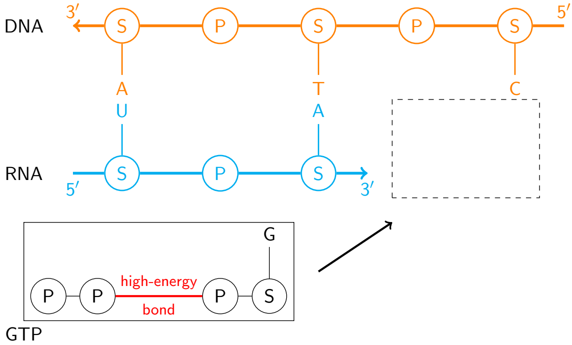


Figure 7-7 *Essential Cell Biology* (© Garland Science 2010)

DNA	$\xrightarrow{\theta}$	RNA
A	$\to$	U
C	$\to$	G
G	$\to$	C
T	$\to$	A

θ is extended to an anti-morphic involution from a DNA sequence to an RNA sequence.

RNA polymerase: Loss-less (1-to-1) synthesis from DNA to RNA

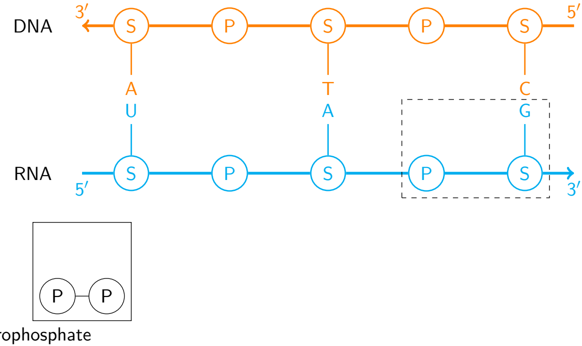


DNA	$\xrightarrow{\theta}$	RNA
A	$\to$	U
C	$\to$	G
G	$\to$	C
T	$\to$	A

The idea of this diagram is taken from [Feynman 1996].

θ is extended to an anti-morphic involution from a DNA sequence to an RNA sequence.

RNA polymerase: Loss-less (1-to-1) synthesis from DNA to RNA

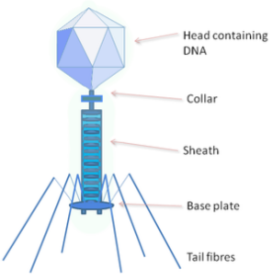


DNA	$\xrightarrow{\theta}$	RNA
A	$\rightarrow$	U
C	$\rightarrow$	G
G	$\rightarrow$	C
T	$\rightarrow$	A

θ is extended to an anti-morphic involution from a DNA sequence to an RNA sequence.

The idea of this diagram is taken from [Feynman 1996].

Transcription rates for survival



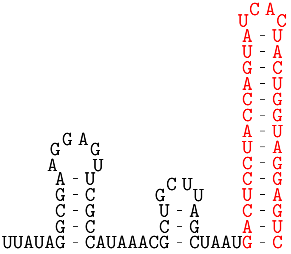
super-fast	Speed	100x slower
erroneous	Accuracy	precise
high but doesn't care	Load on NTP production	low
T7 RNA polymerase	Type	3 specialized polymerases

Regulate gene expression by CF [Watters et al. 2016]

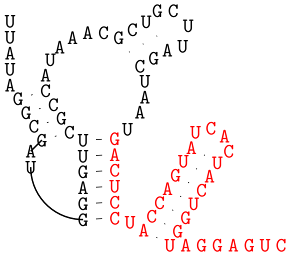
Concentration of NaF (sodium fluoride) may promote or inhibit the transcript

UUAUAGGCGAUGGAGUUCGCCAUAAACGCUGCUUAGCUAAUGACUCCUACCAGUAUCACUACUGGUAGGAGUC

to fold into the terminator stem cotranscriptionally.



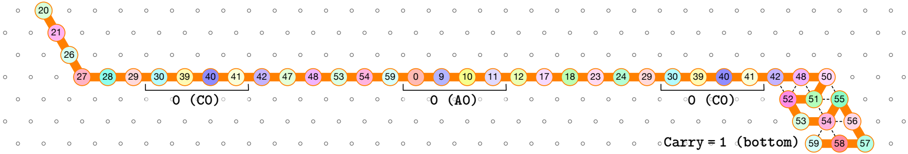
Terminated (0 mM NaF)



Anti-terminated (10 mM NaF)

Oritatami system [Geary, Meunier, Schabanel, S. 2016]

a mathematical model to study computational aspects of CF.

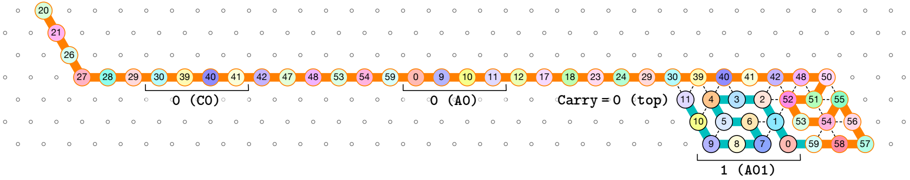


Fixed bit-width binary counter system in oritatami

[Geary, Meunier, Schabanel, S. 2016]

Oritatami system [Geary, Meunier, Schabanel, S. 2016]

a mathematical model to study computational aspects of CF.

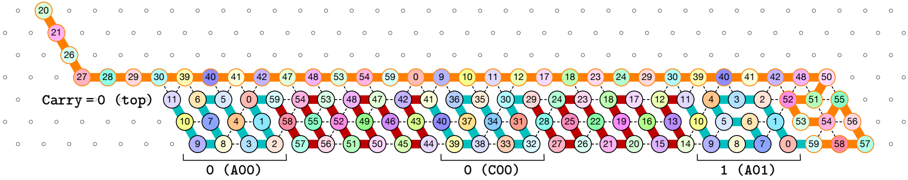


Fixed bit-width binary counter system in oritatami

[Geary, Meunier, Schabanel, S. 2016]

Oritatami system [Geary, Meunier, Schabanel, S. 2016]

a mathematical model to study computational aspects of CF.



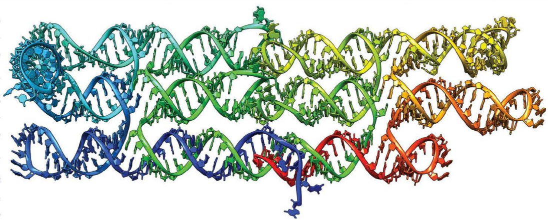
Fixed bit-width binary counter system in oritatami

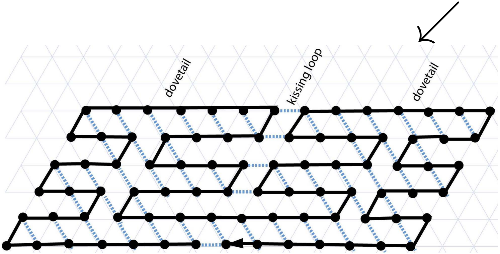
[Geary, Meunier, Schabanel, S. 2016]

Abstraction[a]

[a]Idea and artworks by Cody Geary.

RNA origami tile



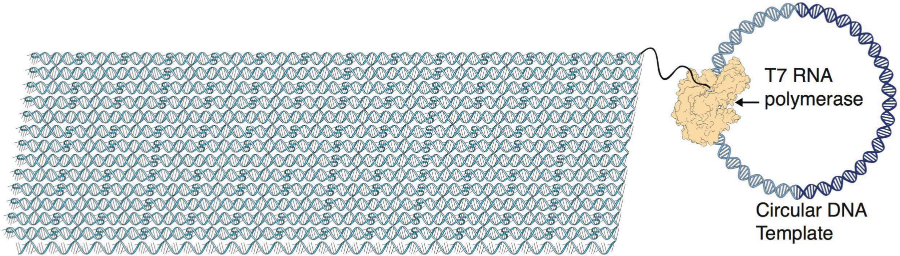


Oritatami conformation

Each • (bead) is labeled with a sequence of several nucleotides such as AACU.

An oritatami system is *cyclic* if its transcript w is of the form $u^* u_p$ for a word $u \in \Sigma^*$ and its prefix u_p.

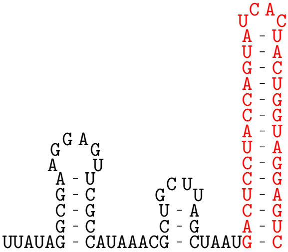
Idea: CF of periodic transcript from a circular DNA



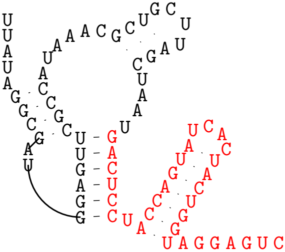
Cyclicity is the only one finite mean considered so far to describe infinite oritatami systems.

Question

Can we design a single transcript *w* that folds as specified in each of given environments?



Terminated (0 mM NaF)

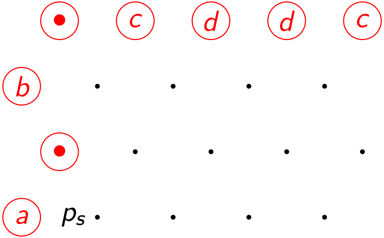


Anti-terminated (10 mM NaF)

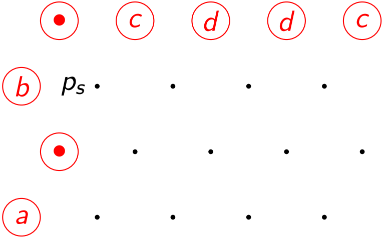
Question

Can we design a single transcript w that folds as specified in each of given environments?

An *environment* e is a pair (B_e, p_s) of a set B_e of points labeled with a letter in Σ (beads) and a point p_s to start folding.



Environment e_1

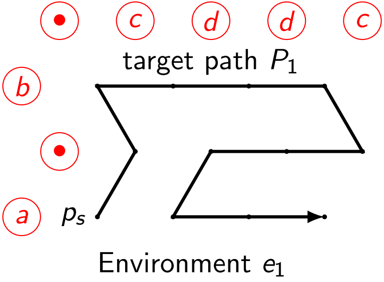


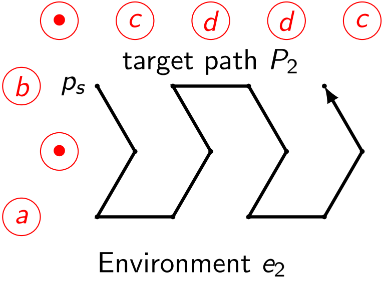
Environment e_2

Question

Can we design a single transcript w that folds as specified in each of given environments?

An *environment* e is a pair (B_e, p_s) of a set B_e of points labeled with a letter in Σ (beads) and a point p_s to start folding.





Oritatami design problem (ODP)

Given: Set of k pairs (e_i, P_i) of an environment and target path s.t. $|P_1| = |P_2| = \cdots = |P_k| = n$

Find: an oritatami system $(\Sigma, R, w, \delta, \alpha)$ that folds w along the path P_i in the environment e_i for all $1 \leq i \leq k$

Oritatami design problem (ODP)

Given: Set of k pairs (e_i, P_i) of an environment and target path s.t. $|P_1| = |P_2| = \cdots = |P_k| = n$

Find: an oritatami system $(\Sigma, R, w, \delta, \alpha)$ that folds w along the path P_i in the environment e_i for all $1 \leq i \leq k$

Rule set design is NP-hard [Ota, S. 2017]

If ODP is modified such that w, δ, α are given instead, then the problem becomes NP-hard even when $k = 1$.

Oritatami design problem (ODP)

Given: Set of k pairs (e_i, P_i) of an environment and target path s.t. $|P_1| = |P_2| = \cdots = |P_k| = n$

Find: an oritatami system $(\Sigma, R, w, \delta, \alpha)$ that folds w along the path P_i in the environment e_i for all $1 \leq i \leq k$

Fixed parameter tractable algorithm [Geary, Meunier, Schabanel, S. 2016]

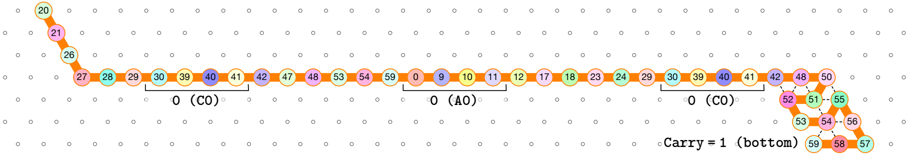
With Σ large enough so that

- w doesn't have to share a bead type with any e_i, and
- any two beads of w are of pairwise-distinct type (hardcodable transcript),

ODP can be solved in time linear in n but exponential in k or δ.

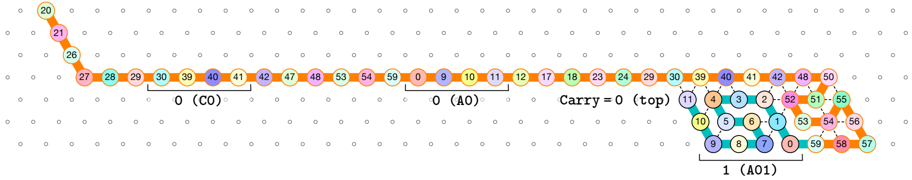
$\rightarrow$ module-based design

Binary counter in oritatami [Geary, Meunier, Schabanel, S. 2016]



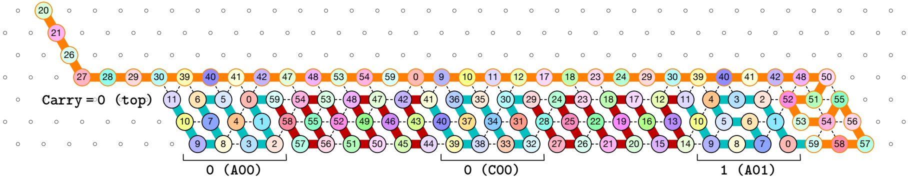
There is a rule set R over $\Sigma = \{0, 1, \ldots, 59\}$ (shown later) according to which, starting from a seed (colored in orange) encoding an integer x in binary, the periodic transcript $w = (0{-}1{-}\cdots{-}59)^*$ folds into zigzags that encode $x+1$, $x+2$, ..., $2^{2k+1}-1$, respectively.

Binary counter in oritatami [Geary, Meunier, Schabanel, S. 2016]



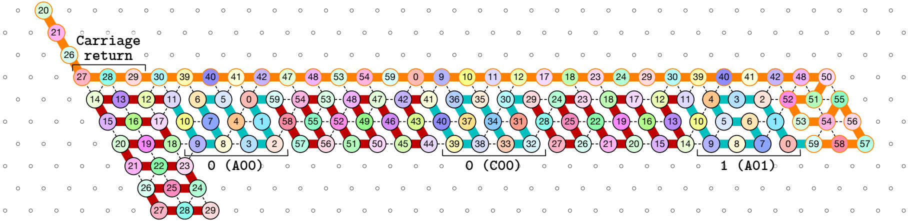
There is a rule set R over $\Sigma = \{0, 1, \ldots, 59\}$ (shown later) according to which, starting from a seed (colored in orange) encoding an integer x in binary, the periodic transcript $w = (0\text{--}1\text{--}\cdots\text{--}59)^*$ folds into zigzags that encode $x+1$, $x+2$, $\ldots$, $2^{2k+1}-1$, respectively.

Binary counter in oritatami [Geary, Meunier, Schabanel, S. 2016]



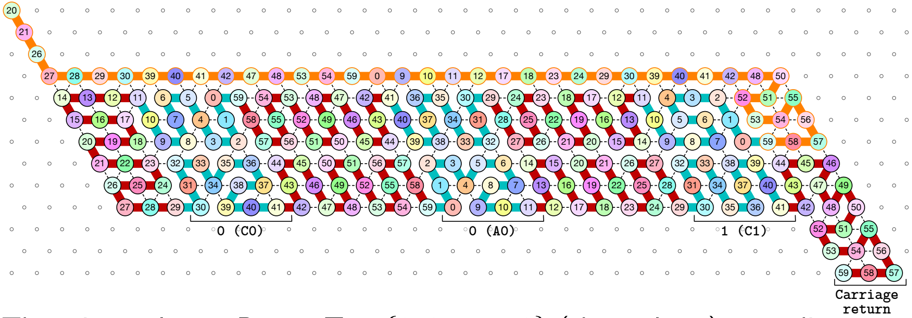
There is a rule set R over $\Sigma = \{0, 1, \ldots, 59\}$ (shown later) according to which, starting from a seed (colored in orange) encoding an integer x in binary, the periodic transcript $w = (0\text{--}1\text{--}\cdots\text{--}59)^*$ folds into zigzags that encode $x+1$, $x+2$, ..., $2^{2k+1} - 1$, respectively.

Binary counter in oritatami [Geary, Meunier, Schabanel, S. 2016]



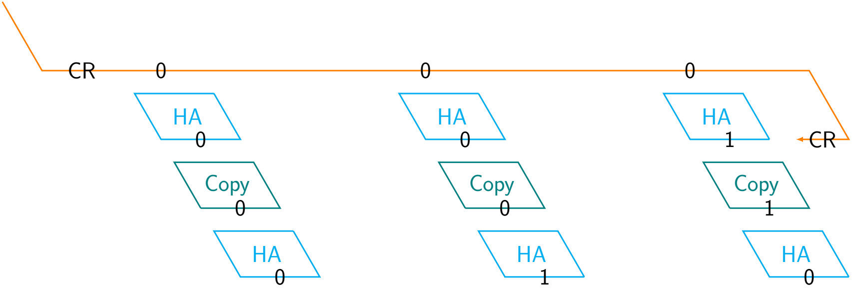
There is a rule set R over $\Sigma = \{0, 1, \ldots, 59\}$ (shown later) according to which, starting from a seed (colored in orange) encoding an integer x in binary, the periodic transcript $w = (0{-}1{-}\cdots{-}59)^*$ folds into zigzags that encode $x + 1$, $x + 2$, ..., $2^{2k+1} - 1$, respectively.

Binary counter in oritatami [Geary, Meunier, Schabanel, S. 2016]



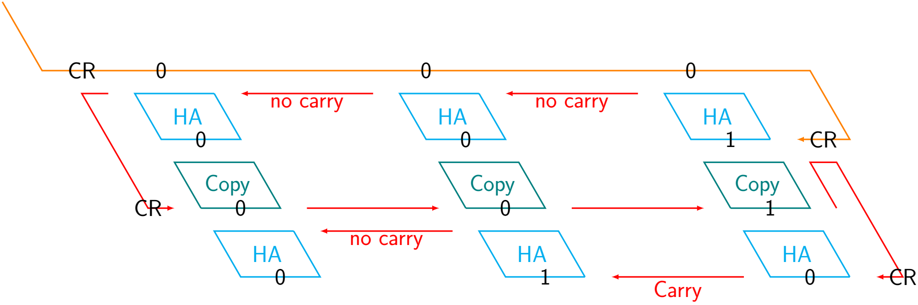
There is a rule set R over $\Sigma = \{0, 1, \ldots, 59\}$ (shown later) according to which, starting from a seed (colored in orange) encoding an integer x in binary, the periodic transcript $w = (0{-}1{-}\cdots{-}59)^*$ folds into zigzags that encode $x+1$, $x+2$, $\ldots$, $2^{2k+1}-1$, respectively.

Global folding pathway design



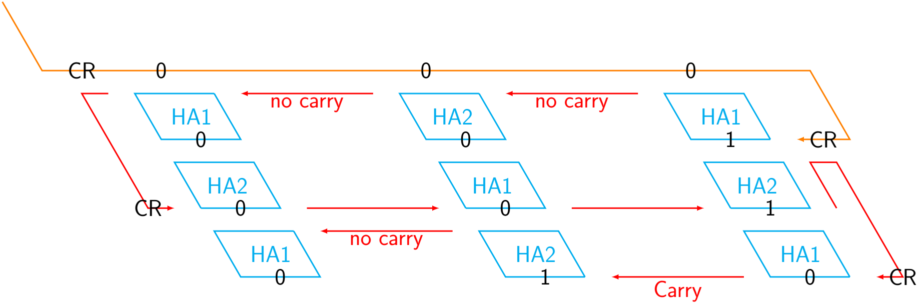
1. Modularization, deployment of modules and signals
2. Wiring with semantics
3. Optimization (shortening period of transcript from the order of bit-width to $O(1)$)

Global folding pathway design



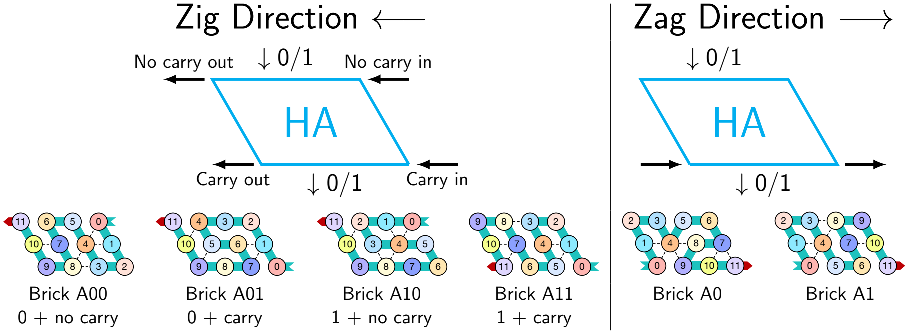
1. Modularization, deployment of modules and signals
2. Wiring with semantics
3. Optimization (shortening period of transcript from the order of bit-width to $O(1)$)

Global folding pathway design



1. Modularization, deployment of modules and signals
2. Wiring with semantics
3. Optimization (shortening period of transcript from the order of bit-width to $O(1)$)

Module and Brick

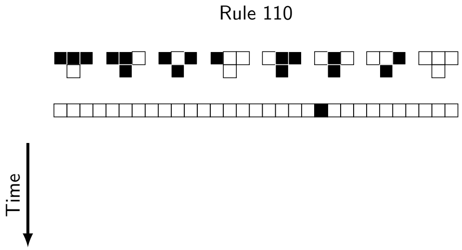


The FPT algorithm lets brick-based programming.

Computation by cotranscriptional folding

Turing universality without RAM

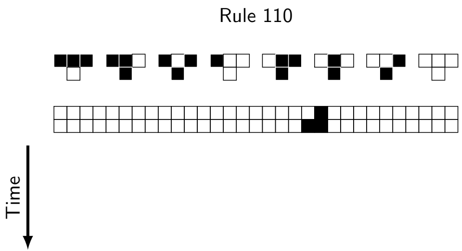
It is extremely challenging to simulate a TM without using RAM, as well exemplified by the Cook's constructive proof for the Turing universality of rule-110 1-d CA [Cook at DNA22, Cook 2004].



Computation by cotranscriptional folding

Turing universality without RAM

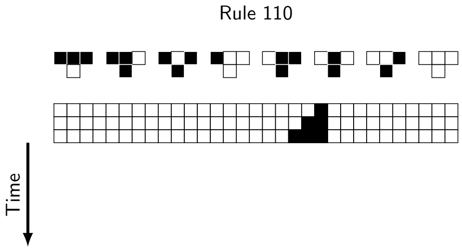
It is extremely challenging to simulate a TM without using RAM, as well exemplified by the Cook's constructive proof for the Turing universality of rule-110 1-d CA [Cook at DNA22, Cook 2004].



Computation by cotranscriptional folding

Turing universality without RAM

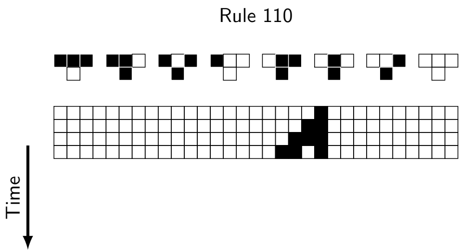
It is extremely challenging to simulate a TM without using RAM, as well exemplified by the Cook's constructive proof for the Turing universality of rule-110 1-d CA [Cook at DNA22, Cook 2004].



Computation by cotranscriptional folding

Turing universality without RAM

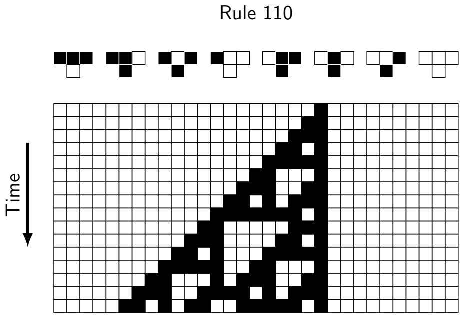
It is extremely challenging to simulate a TM without using RAM, as well exemplified by the Cook's constructive proof for the Turing universality of rule-110 1-d CA [Cook at DNA22, Cook 2004].



Computation by cotranscriptional folding

Turing universality without RAM

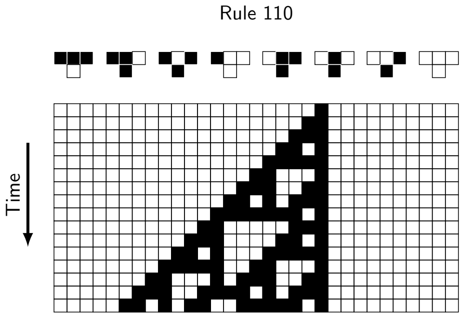
It is extremely challenging to simulate a TM without using RAM, as well exemplified by the Cook's constructive proof for the Turing universality of rule-110 1-d CA [Cook at DNA22, Cook 2004].

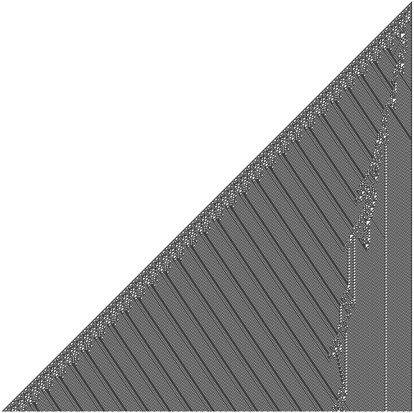


Computation by cotranscriptional folding

Turing universality without RAM

It is extremely challenging to simulate a TM without using RAM, as well exemplified by the Cook's constructive proof for the Turing universality of rule-110 1-d CA [Cook at DNA22, Cook 2004].





Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$u_0 =$	0	1	0					
$u_1 =$		1	0					
		1	0	α_2				
$u_2 =$			0	1	1			
$u_3 =$				1	1			
				1	1	α_1		
$u_4 =$					1			
					1	α_3		
$u_5 =$						0		
$u_6 =$							Halt	

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$u_0 =$ 0 1 0
$u_1 =$ 1 0
1 0 α_2
$u_2 =$ 0 1 1
$u_3 =$ 1 1
1 1 α_1
$u_4 =$ 1
1 α_3
$u_5 =$ 0
$u_6 =$ Halt

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$
$\rightarrow \quad \alpha_1 = \lambda$
$\rightarrow \quad \alpha_2 = 11$
$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$u_0 =$	0	1	0				
$u_1 =$		1	0				
		1	0	α_2			
$u_2 =$			0	1	1		
$u_3 =$				1	1		
				1	1	α_1	
$u_4 =$					1		
					1	α_3	
$u_5 =$						0	
$u_6 =$							Halt

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$$
\begin{array}{lllllll}
u_0 = & 0 & 1 & 0 & & & \\
u_1 = & & 1 & 0 & & & \\
 & & 1 & 0 & \alpha_2 & & \\
u_2 = & & & 0 & 1 & 1 & \\
u_3 = & & & & 1 & 1 & \\
 & & & & 1 & 1 & \alpha_1 \\
u_4 = & & & & & 1 & \\
 & & & & & 1 & \alpha_3 \\
u_5 = & & & & & & 0 \\
u_6 = & & & & & & \texttt{Halt}
\end{array}
$$

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$u_0 =$	0	1	0				
$u_1 =$		1	0				
		1	0	α_2			
$u_2 =$			0	1	1		
$u_3 =$				1	1		
				1	1	α_1	
$u_4 =$					1		
					1	α_3	
$u_5 =$						0	
$u_6 =$							Halt

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$u_0 =$	0	1	0				
$u_1 =$		1	0				
		1	0	α_2			
$u_2 =$			0	1	1		
$u_3 =$				1	1		
				1	1	α_1	
$u_4 =$					1		
					1	α_3	
$u_5 =$						0	
$u_6 =$							Halt

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$$
\begin{array}{lllllll}
u_0 = & 0 & 1 & 0 & & & \\
u_1 = & & 1 & 0 & & & \\
 & & 1 & 0 & \alpha_2 & & \\
u_2 = & & & 0 & 1 & 1 & \\
u_3 = & & & & 1 & 1 & \\
 & & & & 1 & 1 & \alpha_1 \\
u_4 = & & & & & 1 & \\
 & & & & & 1 & \alpha_3 \\
u_5 = & & & & & & 0 \\
u_6 = & & & & & & \texttt{Halt}
\end{array}
$$

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$$
\begin{array}{lllllll}
u_0 = & 0 & 1 & 0 & & & \\
u_1 = & & 1 & 0 & & & \\
 & & 1 & 0 & \alpha_2 & & \\
u_2 = & & & 0 & 1 & 1 & \\
u_3 = & & & & 1 & 1 & \\
 & & & & 1 & 1 & \alpha_1 \\
u_4 = & & & & & 1 & \\
 & & & & & \color{red}{1} & \color{blue}{\alpha_3} \\
u_5 = & & & & & & 0 \\
u_6 = & & & & & & \texttt{Halt}
\end{array}
$$

Cyclic list of appendants

$$
\begin{array}{ll}
\rightarrow & \alpha_0 = 110 \\
\rightarrow & \alpha_1 = \lambda \\
\rightarrow & \alpha_2 = 11 \\
\rightarrow & \alpha_3 = 0
\end{array}
$$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$u_0 =$	0	1	0				
$u_1 =$		1	0				
		1	0	α_2			
$u_2 =$			0	1	1		
$u_3 =$				1	1		
				1	1	α_1	
$u_4 =$					1		
					1	α_3	
$u_5 =$						0	
$u_6 =$							Halt

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

Turing universality without RAM

Skipping cts

Derivation

$u_0 =$	0	1	0				
$u_1 =$		1	0				
		1	0	α_2			
$u_2 =$			0	1	1		
$u_3 =$				1	1		
				1	1	α_1	
$u_4 =$					1		
					1	α_3	
$u_5 =$						0	
$u_6 =$							`Halt`

Cyclic list of appendants

$\rightarrow \quad \alpha_0 = 110$

$\rightarrow \quad \alpha_1 = \lambda$

$\rightarrow \quad \alpha_2 = 11$

$\rightarrow \quad \alpha_3 = 0$

Rewriting rule

1. The leftmost letter (0/1) is read.
2. Rotate the list by 1,
3. If it is 1, append the current appendant at the end of the current word, and rotate the list by 1.
4. Delete the (leftmost) read letter.

Computation by cotranscriptional folding

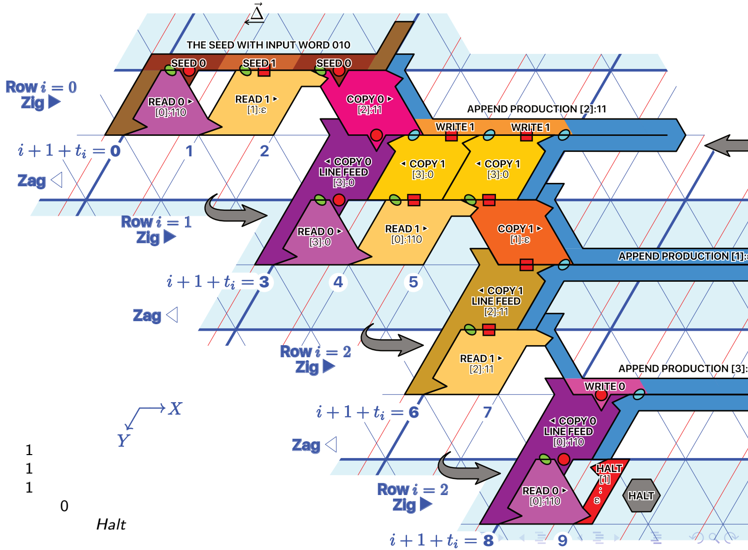
Simulation of SCTS in oritatami: High level overview

Cyclic list of appendants

$$\alpha_0 = 110$$
$$\alpha_1 = \lambda$$
$$\alpha_2 = 11$$
$$\alpha_3 = 0$$

Derivation

$u_0 =$	0	1	0				
$u_1 =$		1	0				
$u_2 =$			0	1	1		
$u_3 =$				1	1		
$u_4 =$					1		
$u_5 =$						0	
$u_6 =$							*Halt*



Computation by cotranscriptional folding

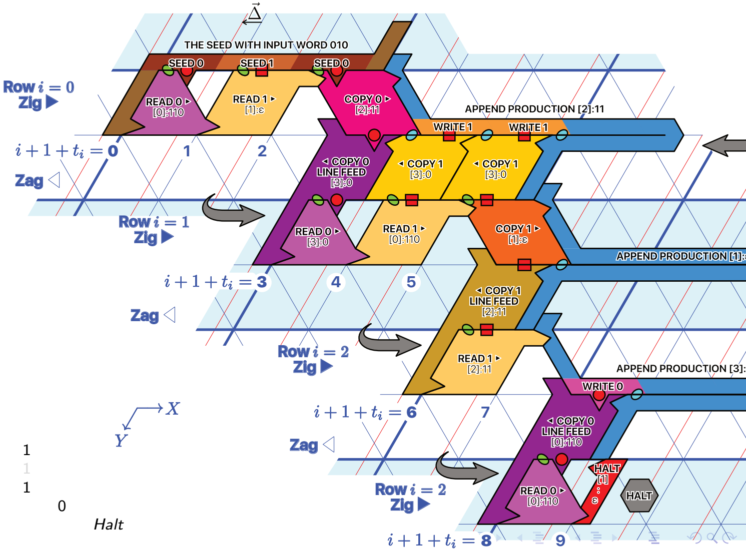
Simulation of SCTS in oritatami: High level overview

Cyclic list of appendants

$\alpha_0 = 110$
$\alpha_1 = \lambda$
$\alpha_2 = 11$
$\alpha_3 = 0$

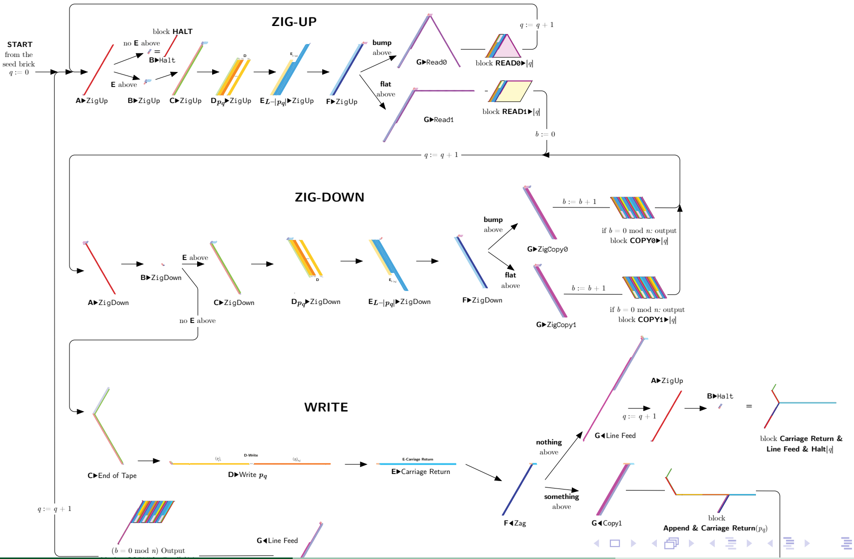
Trimmed derivation

$u_0 =$	0	1	0				
$u_1 =$		1	0				
$u_2 =$			0	1	1		
$u_3 =$				1	1		
$u_4 =$					1		
$u_5 =$						0	
$u_6 =$							*Halt*



Computation by cotranscriptional folding

Brick automaton

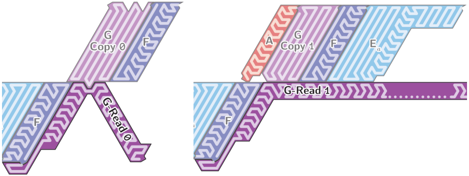


Computation by cotranscriptional folding

Brick example

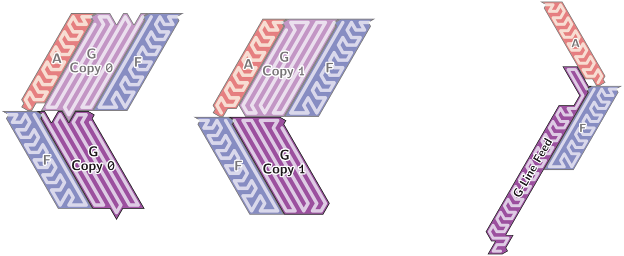
Module G takes 5 bricks

Read-0/1 in zig-up



Linefeed

Copy-0/1 in zig/zag-down



Computation by cotranscriptional folding

Brick and Transcript

The oritatami system involves 7 modules A, B, C, D, E, F, G.

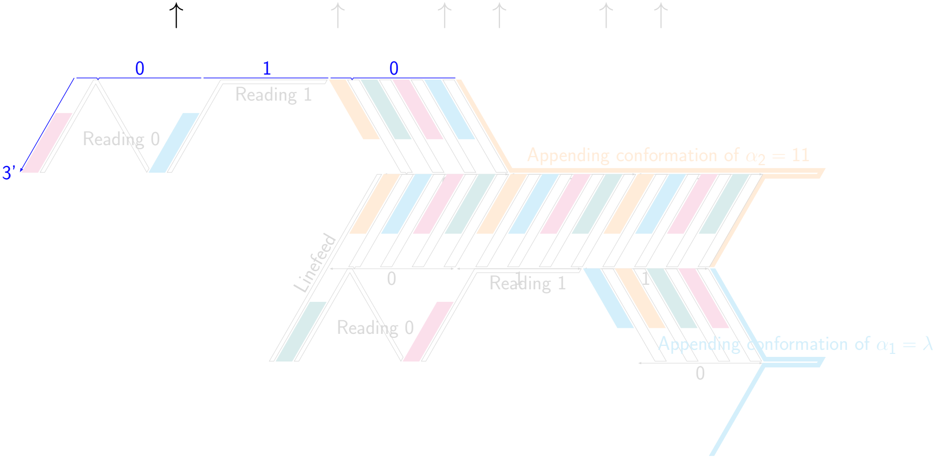
- Each appendant α_i of an SCTS simulated is encoded as $ABCD(\alpha_i)E(\alpha_i)FG$.
- Let $P(\alpha_i) = ABCD(\alpha_i)E(\alpha_i)F$
- The transcript is cyclic as:

$$w = (P(\alpha_0)G \cdot P(\alpha_1)G \cdots P(\alpha_{n-1})G)^*.$$

Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

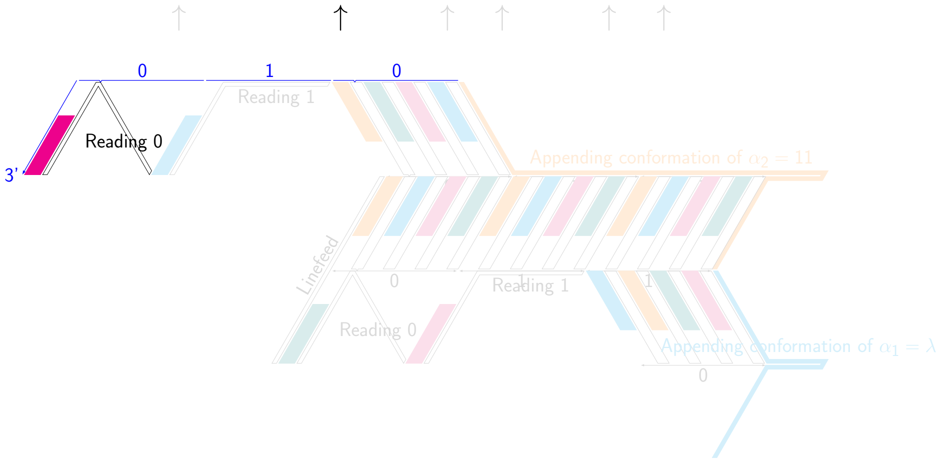
Let $w = (\ P(\alpha_0) \ G \ P(\alpha_1) \ G \ P(\alpha_2) \ G \ P(\alpha_3) \ G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

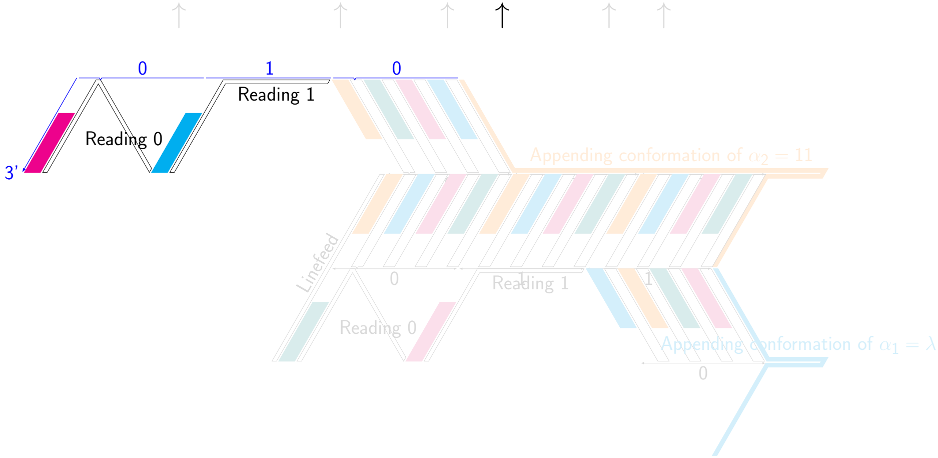
Let $w = (\ P(\alpha_0)\ \ G\ \ P(\alpha_1)\ \ G\ \ P(\alpha_2)\ \ G\ \ P(\alpha_3)\ \ G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

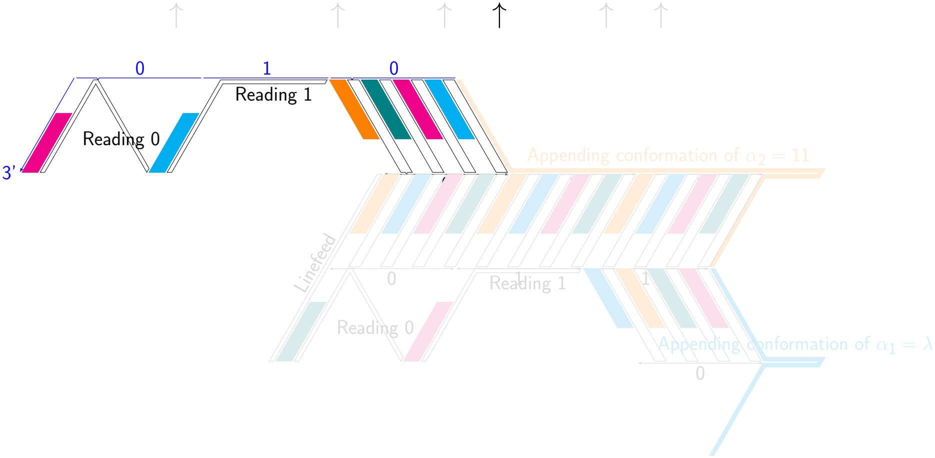
Let $w = (\quad P(\alpha_0) \quad G \quad P(\alpha_1) \quad G \quad P(\alpha_2) \quad G \quad P(\alpha_3) \quad G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

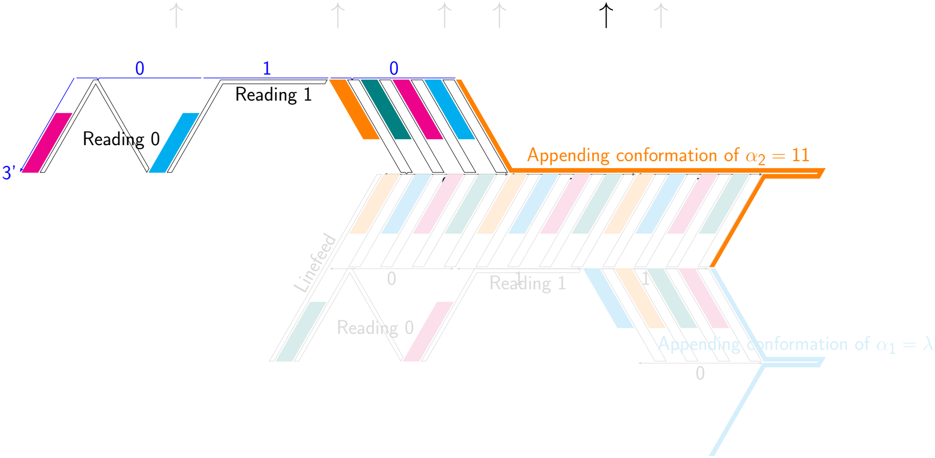
Let $w = (\; P(\alpha_0) \quad G \quad P(\alpha_1) \quad G \quad P(\alpha_2) \quad G \quad P(\alpha_3) \quad G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

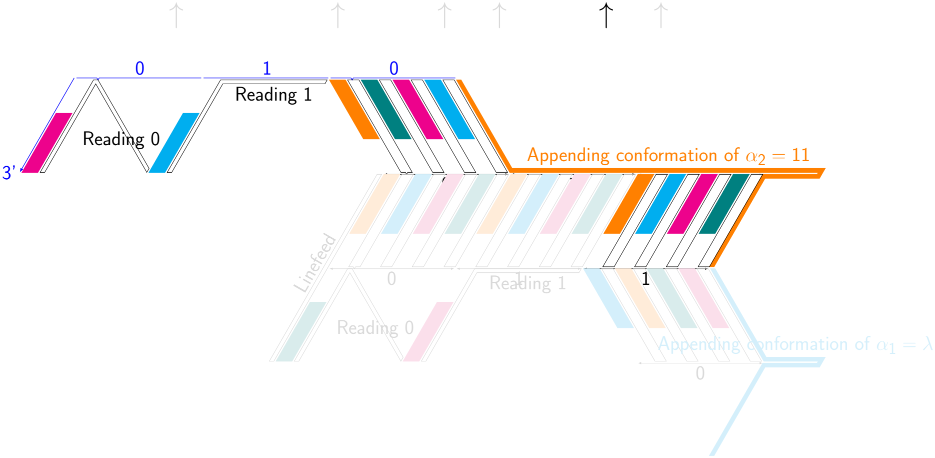
Let $w = (\ P(\alpha_0)\ \ G\ \ P(\alpha_1)\ \ G\ \ P(\alpha_2)\ \ G\ \ P(\alpha_3)\ \ G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

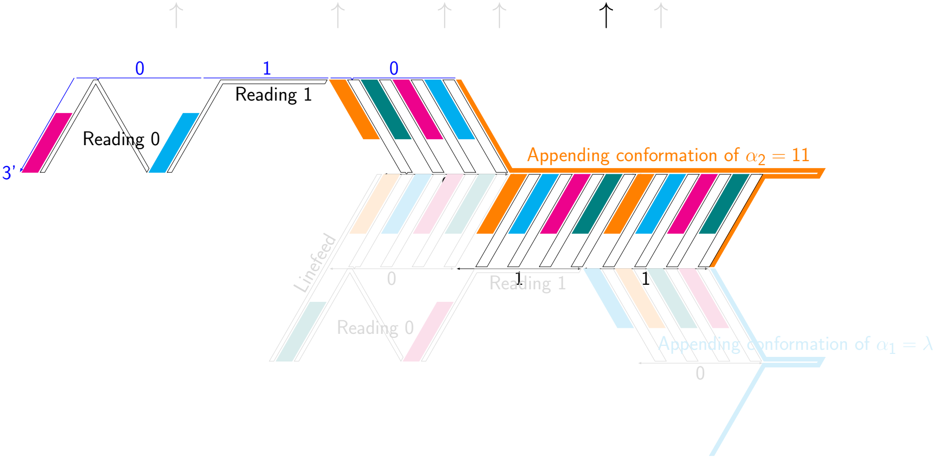
Let $w = (\quad P(\alpha_0) \quad G \quad P(\alpha_1) \quad G \quad P(\alpha_2) \quad G \quad P(\alpha_3) \quad G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

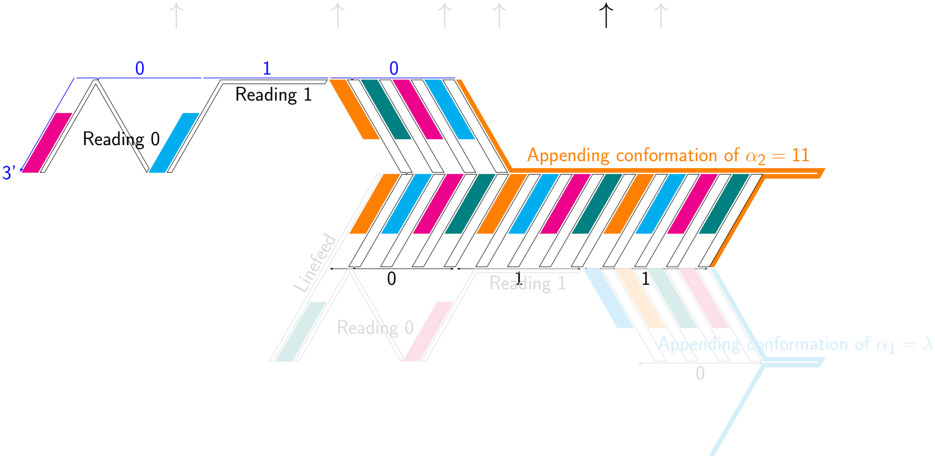
Let $w = (\ P(\alpha_0)\ \ G\ \ P(\alpha_1)\ \ G\ \ P(\alpha_2)\ \ G\ \ P(\alpha_3)\ \ G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

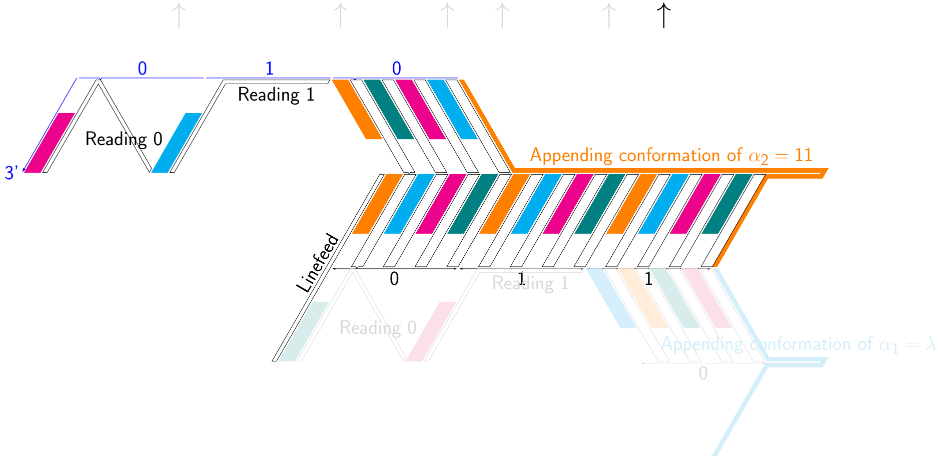
Let $w = (\;\; P(\alpha_0) \;\; G \;\; P(\alpha_1) \;\; G \;\; P(\alpha_2) \;\; G \;\; P(\alpha_3) \;\; G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

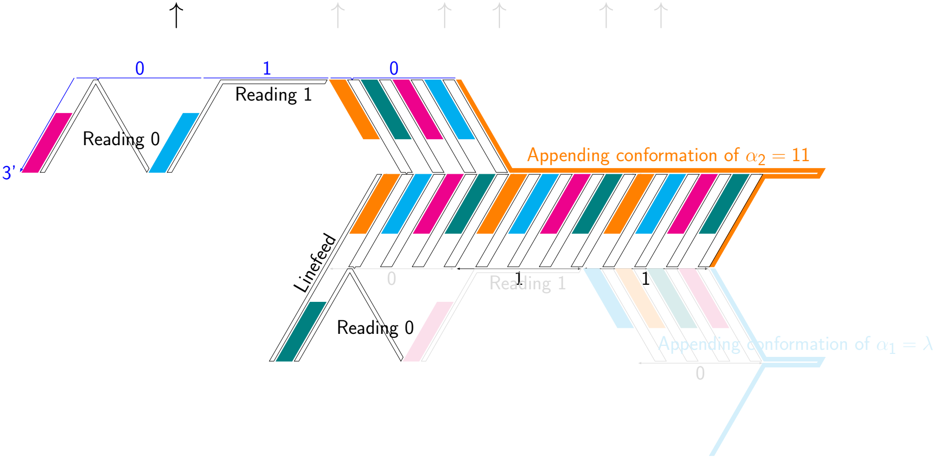
Let $w = (\ P(\alpha_0)\ \ G\ \ P(\alpha_1)\ \ G\ \ P(\alpha_2)\ \ G\ \ P(\alpha_3)\ \ G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

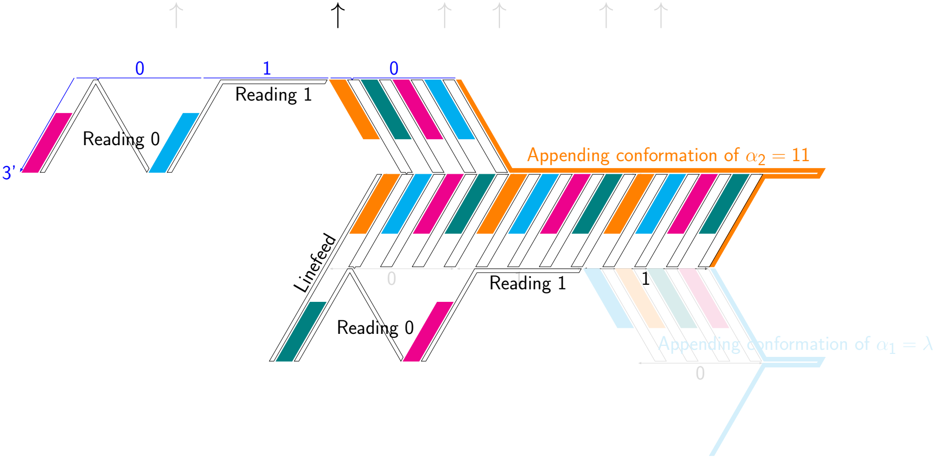
Let $w = (\; P(\alpha_0) \;\; G \;\; P(\alpha_1) \;\; G \;\; P(\alpha_2) \;\; G \;\; P(\alpha_3) \;\; G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

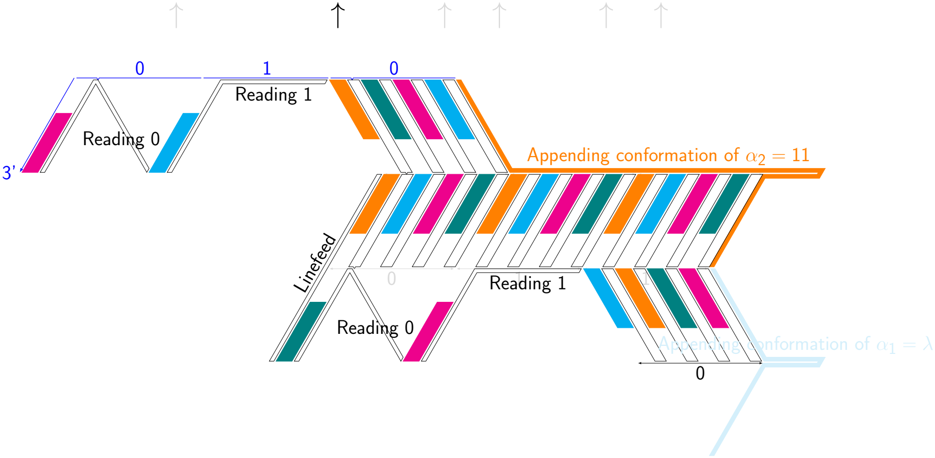
Let $w = (\ P(\alpha_0) \ \ G \ \ P(\alpha_1) \ \ G \ \ P(\alpha_2) \ \ G \ \ P(\alpha_3) \ \ G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

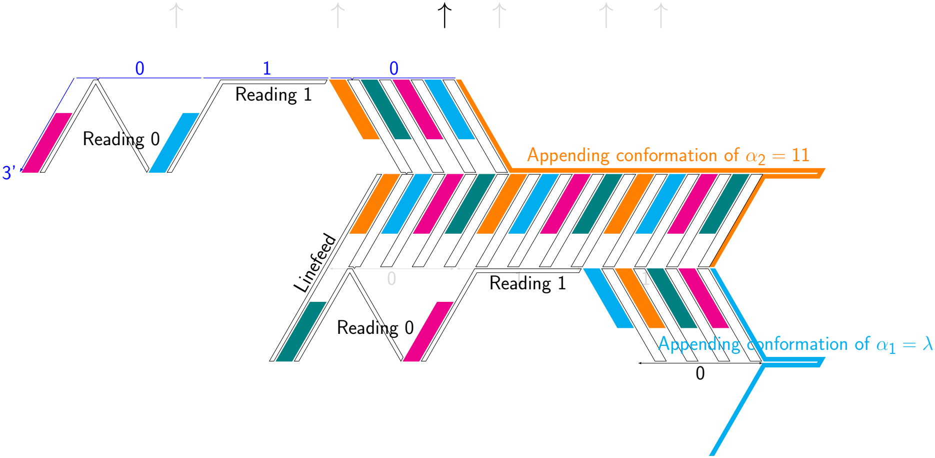
Let $w = (\quad P(\alpha_0) \quad G \quad P(\alpha_1) \quad G \quad P(\alpha_2) \quad G \quad P(\alpha_3) \quad G)^*$



Computation by cotranscriptional folding

Simulation of SCTS by OS: Brick level overview

Let $w = (\quad P(\alpha_0) \quad G \quad P(\alpha_1) \quad G \quad P(\alpha_2) \quad G \quad P(\alpha_3) \quad G)^*$



Computation by cotranscriptional folding

Proof sketch and efficiency

$$\text{Universal TM} \xrightarrow{\text{[Neary 2008]}} \text{Skipping cts} \xrightarrow{\text{Our contribution}} \text{Oritatami}$$

Theorem

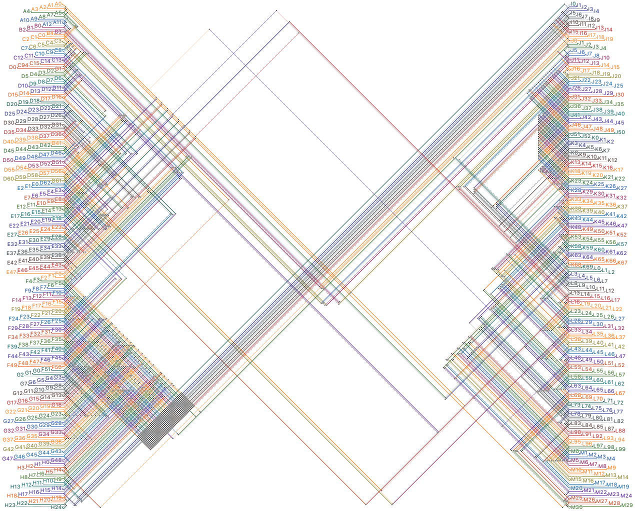
There are a fixed set Σ of 542 bead types, a rule set R, and the following two encodings:

- *π, which maps in polynomial time, any single tape Turing machine $\mathcal{M}$ to a sequence $\pi_{\mathcal{M}} \in \Sigma^*$*
- *(s, σ), which maps in polynomial time, $\mathcal{M}$ and its input x to a seed $\sigma_{\mathcal{M}}(x)$ of size linear in $|x|$ (and polynomial in $|\mathcal{M}|$)*

with which at delay 3, the oritatami system halts its folding from $\sigma_{\mathcal{M}}(x)$ after $O_{\mathcal{M}}(t^4 \log^2 t)$ beads if and only if $\mathcal{M}$ halts on x after t steps.

Computation by cotranscriptional folding

Rule set



References I

R. D. Barish, R. Schulman, P. W. K. Rothemund, and E. Winfree.
An information-bearing seed for nucleating algorithmic self-assembly.
PNAS 106(15): 6054-6059, 2009.

M. Cook
Universality in elementary cellular automata.
Complex Systems 15 (2004) 1-40

M. Cook
Computing without random access memory: Cyclic tag systems for proofs and interpretation.
Invited talk at DNA 22, Sept. 4-8, 2016.

References II

C. G. Evans
Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly
Ph. D. thesis, Caltech, 2014

R. P. Feynman
Feynman Lectures on Computation.
Perseus (for Hbg), 1996

C. Geary, P-E. Meunier, N. Schbanel, S. Seki
Programming biomolecules that fold greedily during translation.
MFCS 2016, LIPIcs 58, 43:1-43:14, 2016

C. Geary, P-E. Meunier, N. Schbanel, S. Seki
Proving the Turing universality of oritatami co-transcriptional folding.
ISAAC 2018, LIPIcs 123, 23:1-23:13, 2018

References III

C. Geary, P. W. K. Rothemund, E. S. Andersen
A single-stranded architecture for cotranscriptional folding of RNA nanostructures.
Science 345 (2014) 799-804

Y-S. Han and H. Kim
Ruleset Optimization on Isomorphic Oritatami Systems
DNA 23, LNCS 10467, 33-45, 2017

Y. Masuda, S. Seki, and Y. Ubukata
Towards the algorithmic molecular self-assembly of fractals by cotranscriptional folding
CIAA 2018, LNCS 10977, 261-73, 2018

References IV

T. Neary
Small universal Turing machines
Ph.D. thesis, National University of Ireland, 2008

M. Ota and S. Seki
Ruleset design problems for oritatami systems
Theor. Comput. Sci. 671 (2017) 26-35

K. E. Watters, E. J. Strobel, A. M. Yu, J. T. Lis, and J. B. Lucks
Cotranscrptional folding of a riboswtich at nucleotide resolution
Nat. Struct. & Mol. Biol. 23(12) (2016) 1124-31

E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman.
Design and self-assembly of two-dimensional DNA crystals.
Nature 394: 539-544, 1998.