Coq 証明支援系による DNA 計算の形式化

2024/11/11

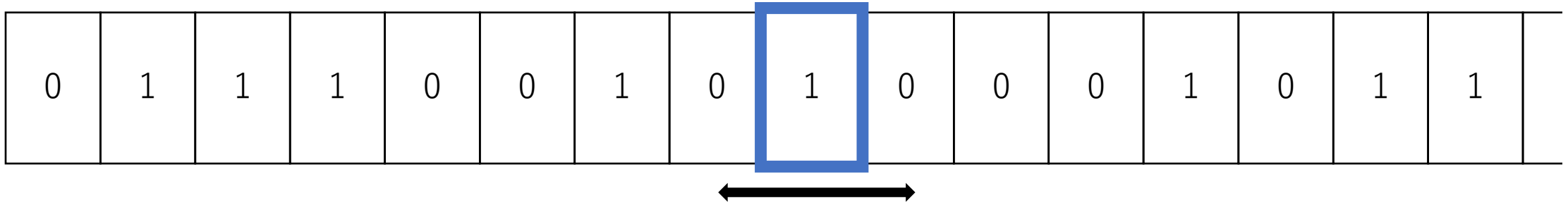
早川　銀河

目次

- DNA計算とは
- DNAドミノ
- スティッカーシステム
- Coq(証明支援系)
- SSReflect/finType
- オートマトンとスティッカーシステムの関係
- まとめと今後の課題

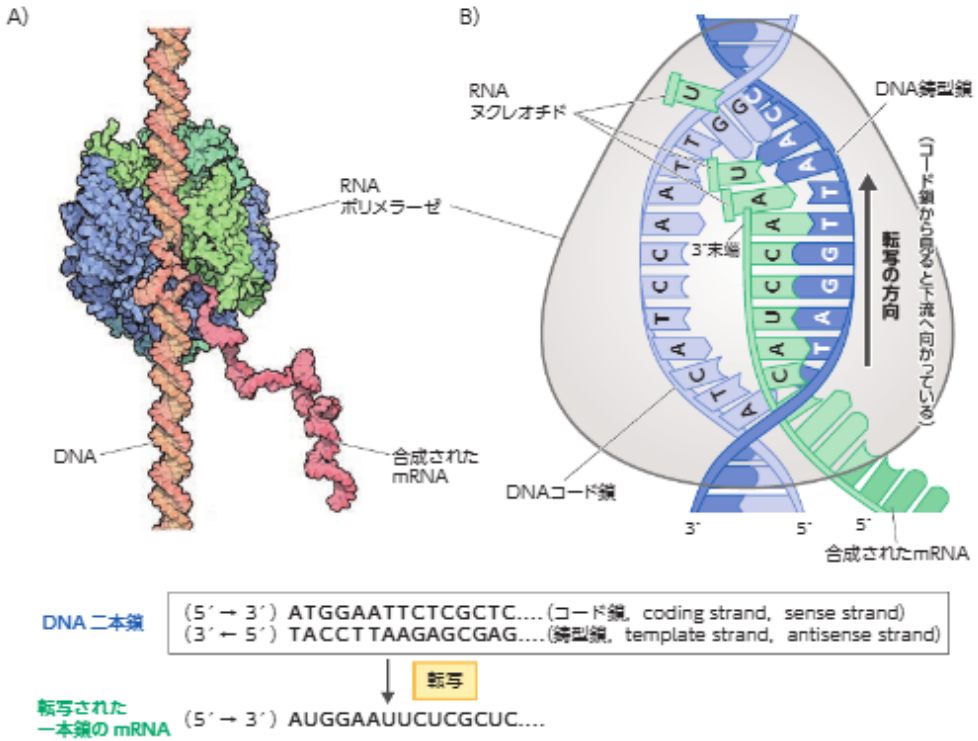
DNA計算とは　(1)チューリングマシン

- チューリングマシンは無限長のテープとテープ上を移動して読み書きを行うヘッドからなる、概念上の計算機である。

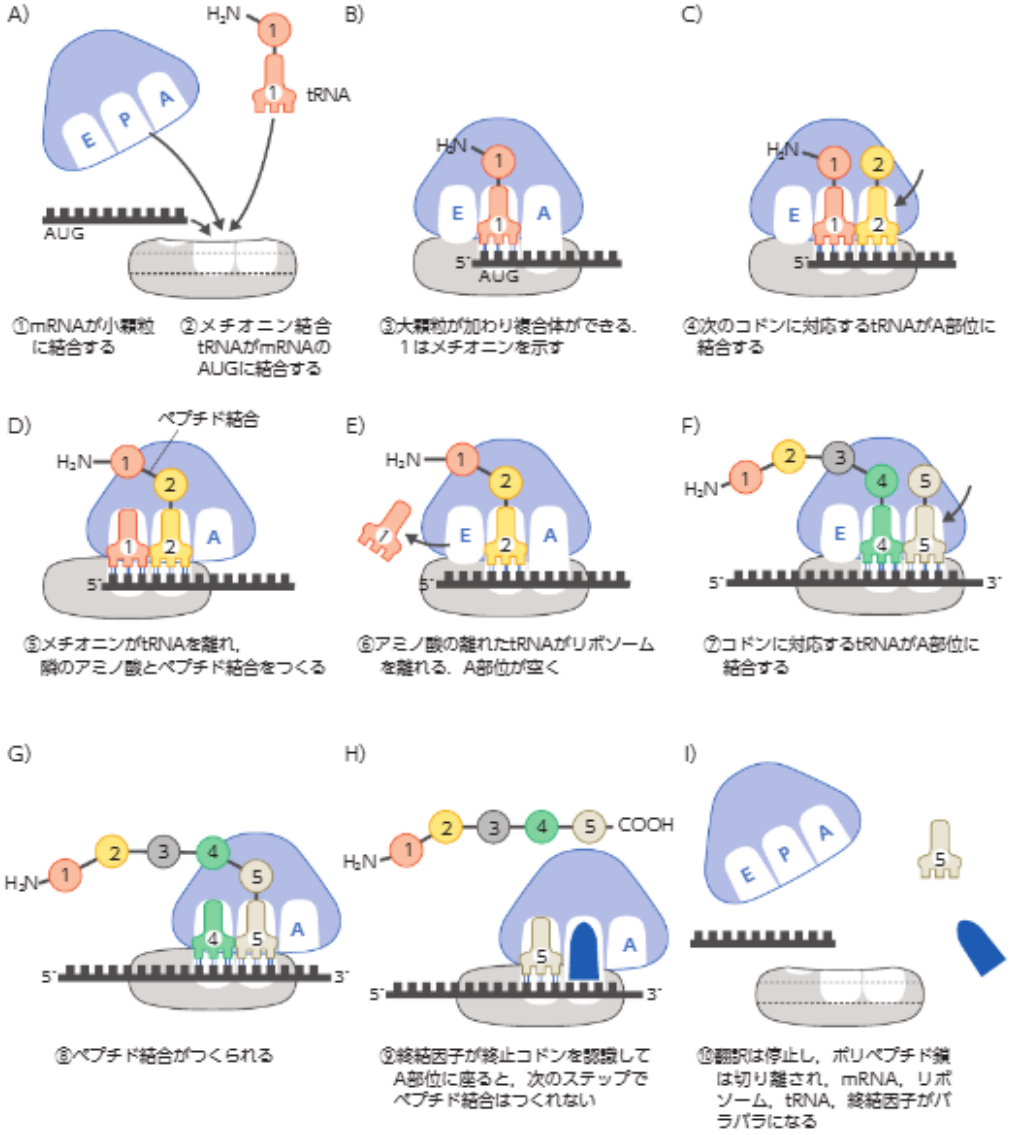


- チューリングが1936年に提唱し、その型を参考にしてノイマン型コンピュータが開発された。

DNA計算とは (2)
人体が持つ“計算機”



●図3-12　RNAポリメラーゼによるmRNAの転写
A）はApril 2003 Molecule of the Month by David Goodsell より，B）は参考図書1より作成



●図3-20　翻訳の過程
A〜E）開始，F〜G）伸長，H〜I）終了．開始コドンの上流には本来は5′非翻訳領域があるが省略している．参考図書3をもとに作成

基礎から学ぶ生物学・細胞生物学 第4版 P93

DNAドミノ(1/3)

- DNA ドミノは DNA の二本鎖構造を参考にして設計された分子装置
- 中央部分で相補的に結合する二本鎖と、両端に露出した粘着末端からなる

$$\left(\binom{V^*}{\lambda} \cup \binom{\lambda}{V^*}\right) \left[\begin{matrix} V^+ \\ V^+ \end{matrix}\right]_\rho \left(\binom{V^*}{\lambda} \cup \binom{\lambda}{V^*}\right)$$

V：アルファベット　　V^*：クリーネ閉包(+は非空)

λ：空文字　　$\rho \subset V \times V$：相補的対応関係

スティッカーシステム(1/3)

スティッカーシステム：$\gamma = (V, \rho, S, D)$

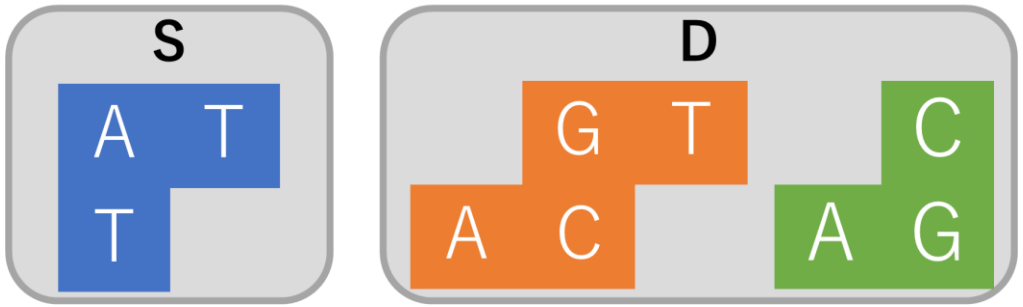
V：アルファベット（記号の有限集合）　　$\rho \subseteq V \times V$

S：二本鎖部を持つドミノの有限集合D：ドミノ対の有限集合

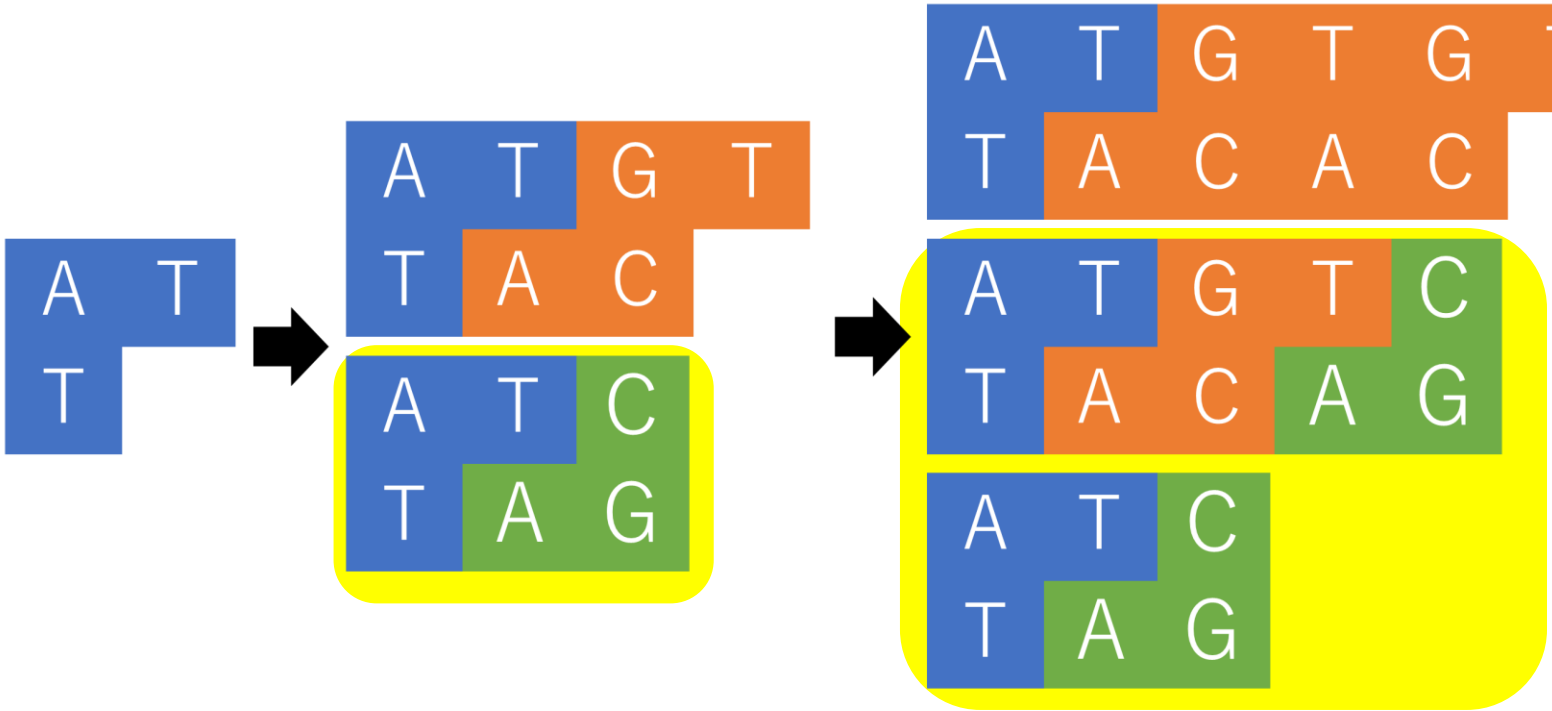
スティッカーシステムの生成言語LS=****

Sのドミノにのドミノ対を繰り返し作用させていき、作成される二本鎖ドミノの上側の文字列をスティッカーの生成する言語とする。

スティッカーシステム(2/3)



$$V = \{A, T, G, C\}$$
$$\rho = \{(A,T), (T,A), (G,C), (C,G)\}$$
$$S = \left\{\begin{bmatrix} A \\ T \end{bmatrix}\begin{pmatrix} T \\ \lambda \end{pmatrix}\right\}$$
$$D = \left\{\left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ A \end{pmatrix}\begin{bmatrix} G \\ C \end{bmatrix}\begin{pmatrix} T \\ \lambda \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ A \end{pmatrix}\begin{bmatrix} C \\ G \end{bmatrix}\right)\right\}$$



このスティッカーが
生成する言語は
$LS(\gamma) = \{AT(GT)^n C | n \in \mathbb{N}\}$

オートマトン vs スティッカー(1/7)

(右側)スティッカーシステムは有界遅延の原始的な計算で正則文法と同じ生成能力を持つことが知られている。

$REG = RSL$　(REG: 正則言語, RSL: 右側スティッカーの生成言語)

今回、オートマトンを模倣するスティッカーシステムを構成することで$REG \subseteq RSL$を形式的に示した。

```
Theorem REG_RSL{state symbol:finType}(M:@automaton state symbol)(s:seq symbol)
(m:nat):s <> nil -> exists n:nat , n <= m ->
accept M s = (s\in(ss_language_prime m (Aut_to_Stk M))).
```

オートマトンvsスティッカー(2/7)

有限オートマトン $\mathrm{M} = (K, V, s0, F, \delta)$

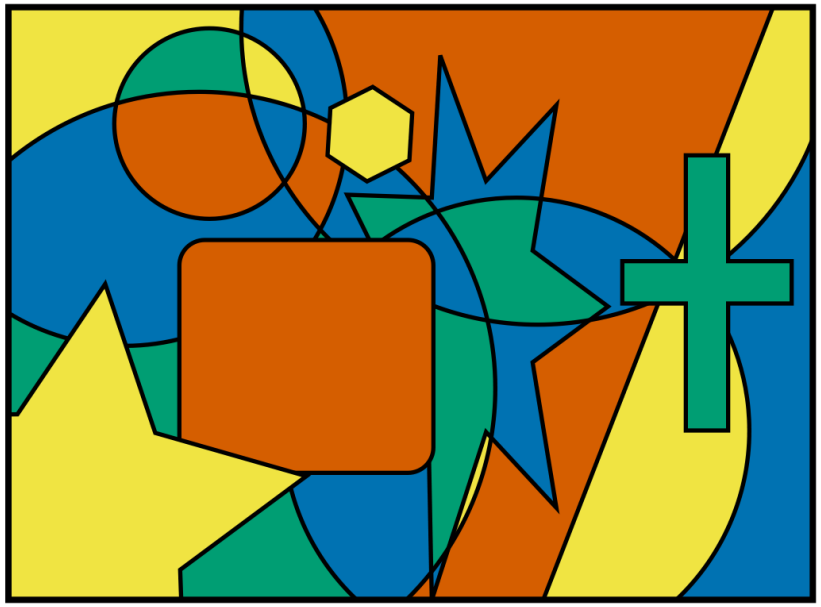
K:状態の有限集合,　　V: アルファベット

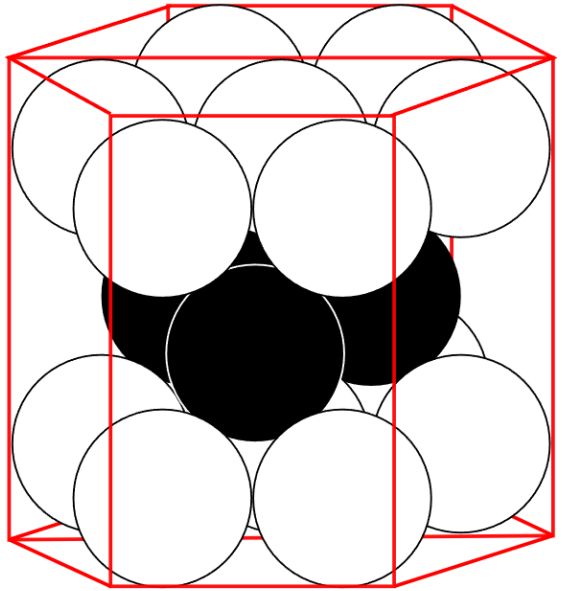
δ: $(K, V\ ) \rightarrow K$: 遷移関数　　$s_0 \in K$初期状態,　$F \subset K$受理状態

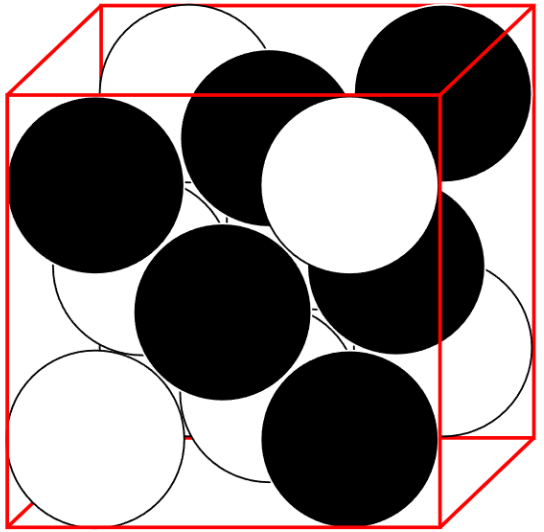
（グラフ形式のオートマトンの図）

Coq(証明支援系)

- フランスの INRIA研究所で開発された証明支援系
- 形式的な検証により、曖昧さのない厳密な証明が可能
- 四色問題や最密充填問題の証明にも用いられた







Wikipedia(四色問題、ケプラー予想)より引用

SSReflect/finType

有限の要素のみからなる型

enumにより要素の全列挙が可能となる

(例)enum bool_finType = [::true;false]

オートマトンやスティッカーシステムで扱う文字,文字列をascii,stringの代わりにfinType,seq finTypeとすることでより厳密な定義が可能となる.

オートマトンの実装

有限オートマトン $\mathrm{M} = (K, V, s0, F, \delta)$
K:状態の有限集合,　　V: アルファベット(文字の有限集合)
δ: (K, V) → K: 遷移関数　$s_0 \in K$初期状態,　　$F \subseteq K$受理状態

```
Structure automaton{state symbol : finType}:={
        init     : state;
        final    : {set state};
        delta    : state -> symbol -> state
}.
```

DNAドミノの実装

中央の二本鎖部と末端の一本鎖部をそれぞれ定義し，

スティッカーシステムの実装

スティッカーシステム：$\gamma = (V, \rho, S, D)$
V：アルファベット（記号の有限集合）　$\rho \subseteq V \times V$
S：二本鎖部を持つドミノの有限集合　　D：ドミノ対の有限集合

```
Structure sticker{symbol : finType}{rho : seq(symbol*symbol)} :={
        start : seq (@domino symbol rho);
        extend : seq ((@domino symbol rho)*(@domino symbol rho))
}.
```

オートマトン vs スティッカー(7/7)

約2800行からなる形式証明の全文
証明の正しさはCoqが保証するため、
手動で確かめる必要はない

```
Theorem REG_RSL{state symbol:finType}(M:@automaton state symbol)(s:seq symbol)
(m:nat):s <> nil -> exists n:nat , n <= m ->
accept M s = (s\in(ss_language_prime m (Aut_to_Stk M))).
```

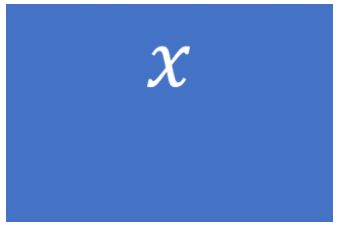
まとめと今後の課題

- 今回、(片側)スティッカーシステムが正則文法以上の生成能力を持つことをCoqで形式的に検証した。
- DNA計算モデルは他にも多数存在し、中には帰納的加算言語を生成可能なものも存在する。
  - (両側)スティッカーシステム
  - ワトソンクリックオートマトン
  - 挿入・削除システム
  - スプライシングシステム
  - 環状スプライシングシステム
- 今後、これらのモデルに関する証明も形式化していきたい。

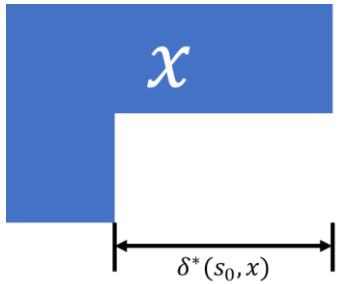
ご清聴ありがとうございました。

オートマトン vs スティッカー(3/7)

S
(開始ドミノ)

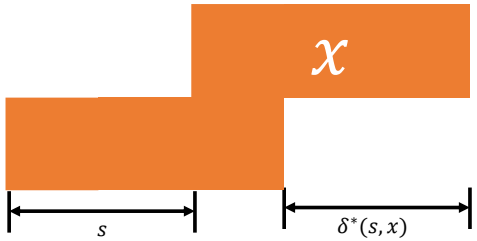


オートマトンがxを受理する
$\delta^*(s_0, x) \in F$

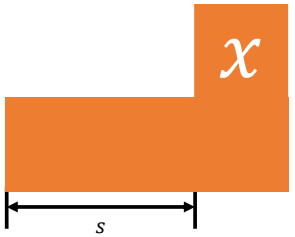


オートマトンにxを作用させた状態
$\delta^*(s_0, x)$を粘着末端で表現

D(結合させる
ドミノ)

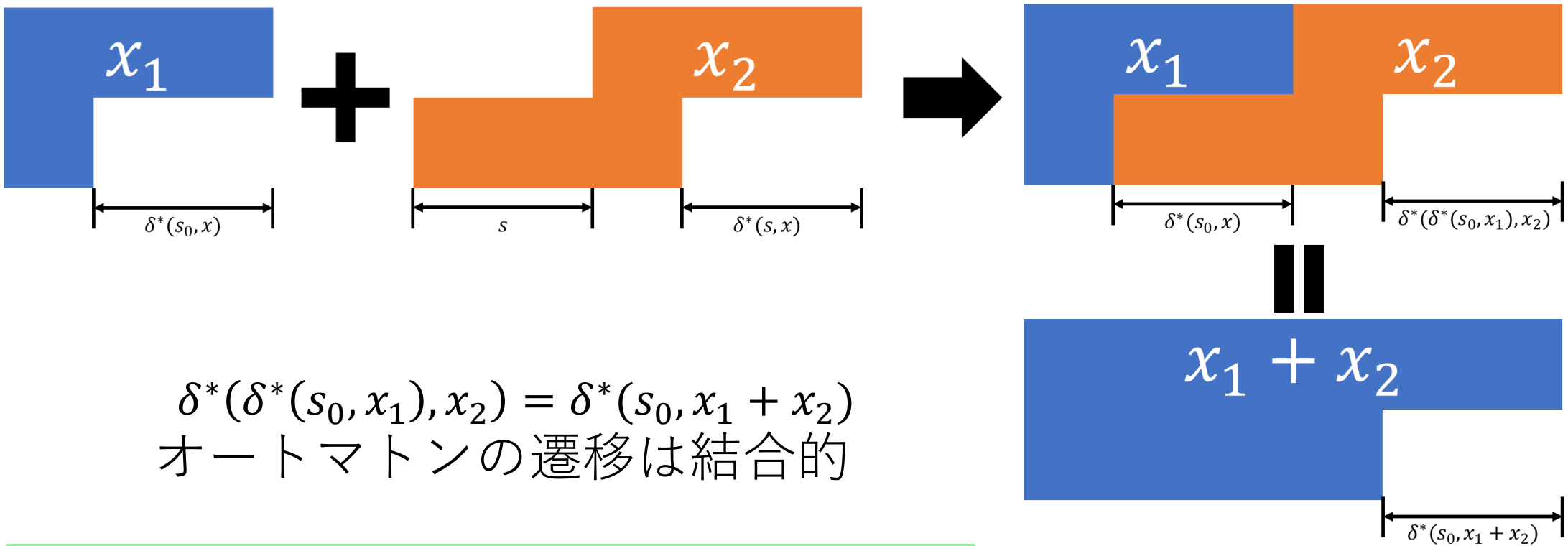


オートマトンがある状態sからxにより
遷移した状態$\delta^*(s, x)$を粘着末端で表現



オートマトンがある状態sから
xを受理する　$\delta^*(s, x) \in F$

オートマトン vs スティッカー(4/7)

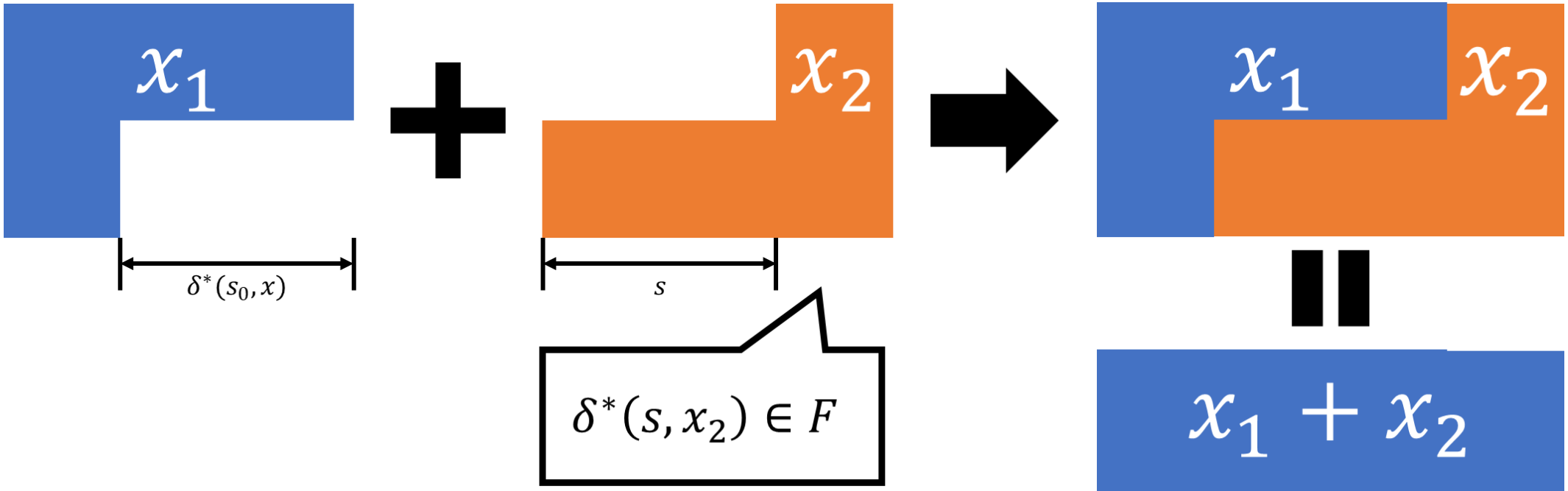


$$\delta^*(\delta^*(s_0,x_1),x_2) = \delta^*(s_0,x_1+x_2)$$

オートマトンの遷移は結合的

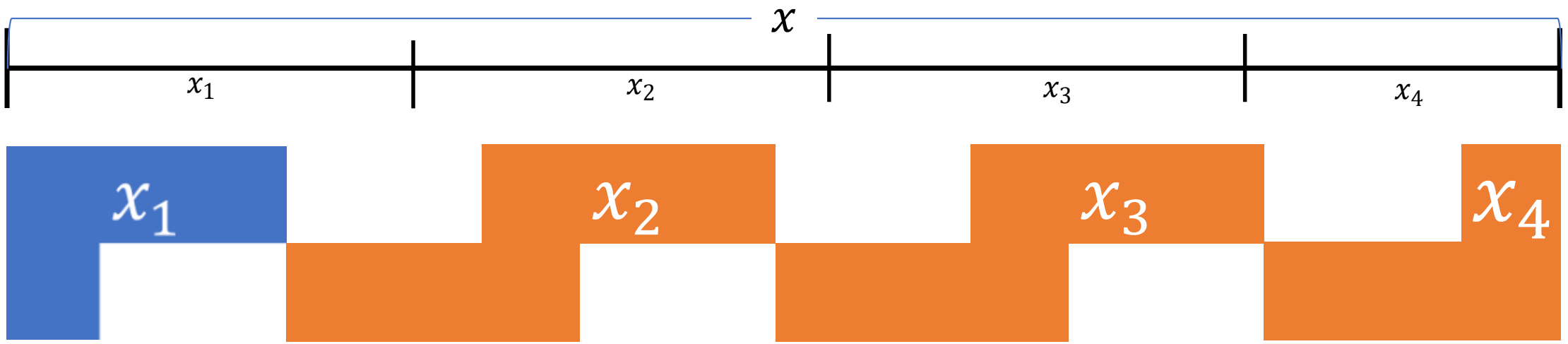
```
Lemma mu'lemma{state symbol:finType}(M:@automaton state symbol)
(s t u:seq symbol):#|state|.+1<=size s -> size t = #|state|.+1 -> u<>nil->
mu' (startDomino M s)(extentionDomino M t u)=
if drop(size s-((index(dstar(delta M)(init M)s)(enum state)).+1))s == u then
  Some (startDomino M (s++t))
else
  None.
```

オートマトン vs スティッカー(5/7)




```
Lemma mu'lemma2{state symbol:finType}(M:@automaton state symbol)
(s t u:seq symbol)(d:domino*domino): #|state|<size s ->
Some d = (stopDomino M u t)->
mu'(startDomino M s)d =
if (drop (size s-(index(dstar(delta M)(init M)s)(enum state)).+1)s)
  ==u then
  mkWK (s++t)
else
  None.
```

オートマトン vs スティッカー(6/7)



$$\delta^*(\delta^*(\delta^*(\delta^*(s_0, x_1), x_2), x_3), x_4) \in F \quad \Leftrightarrow \delta^*(s_0, x_1 + x_2 + x_3 + x_4)$$
$$\Leftrightarrow \delta^*(s_0, x) \in F \Leftrightarrow x\text{はオートマトンにより受理される}$$

任意の文字列は分割した有限長の文字列に対応するドミノの組み合わせで受理判定が可能
⇒オートマトンの受理判定をスティッカーで模倣できる

付録　チョムスキー階層

- ノーム・チョムスキーは言語を生成文法と対応付け、その包含関係で階層分けした。(1956)
- 各階層はオートマトンなどの受理機械によっても特徴づけられることが知られている。

チョムスキー階層	言語	受理機械
Type-0	帰納的加算	チューリングマシン
Type-1	文脈依存	線型拘束オートマトン
Type-2	文脈自由	プッシュダウンオートマトン
Type-3	正規　正則	（有限）オートマトン

スティッカー(StickerModule.v)

```
(*ドミノを定義　二本鎖部分と粘着末端、もしくは空白部の組み合わせからなる*)
Inductive domino{symbol:finType}{rho:Rho symbol}:=
|null : domino
|Simplex : @stickyend symbol -> domino
|WK : @wk symbol rho -> domino
|L : @stickyend symbol -> @wk symbol rho -> domino
|R : @wk symbol rho -> @stickyend symbol -> domino
|LR : @stickyend symbol -> @wk symbol rho -> @stickyend symbol -> domino.
(*スティッカーの定義　開始ドミノとそこに結合させていくドミノ対の情報を持ち、
開始ドミノが二本鎖部分を持つ証明を要求する*)
Structure sticker{symbol:finType}{rho:Rho symbol}:= Sticker{
  start : seq (@domino symbol rho);
  extend : seq (@domino symbol rho*@domino symbol rho);
  startP : all st_correct start
}.
```

DNAドミノ(2/3)

$V = \{A, T, G, C\}, \rho = \{(A, T), (T, A), (G, C), (C, G)\}$ とした場合に現実の DNA分子とみなすことができる.

$d1 := \begin{pmatrix} \lambda \\ CG \end{pmatrix} \begin{bmatrix} AT \\ TA \end{bmatrix}_\rho \begin{pmatrix} GC \\ \lambda \end{pmatrix}$ としたとき,ドミノの結合演算 μ により

$$\mu(d1, d1) = \begin{pmatrix} \lambda \\ CG \end{pmatrix} \begin{bmatrix} ATGCAT \\ TACGTA \end{bmatrix}_\rho \begin{pmatrix} GC \\ \lambda \end{pmatrix}$$

となるが, これは現実の DNA 分子同士がライゲーションとアニーリングにより結合する様子をよく再現している.

オートマトン(AutomatonModule.v)

```
(* 有限オートマトンの定義 *)
Structure automaton{state symbol:finType}:= Automaton {
  init : state;
  delta : state -> symbol -> state;
  final : {set state}
}.
(*受理判定の関数*)
Definition accept{state symbol:finType}(M:@automaton state
symbol)
  (str:seq symbol):bool := dstar (delta M) (init M) str¥in final M.
```

スティッカーシステム(3/3)　Coqで定義

```
Structure sticker{symbol:finType}{rho:Rho symbol}:= Sticker{
  start : seq (@domino symbol rho);
  extend : seq (@domino symbol rho*@domino symbol rho);
  startP : all st_correct start
}.

Fixpoint ss_generate_prime{symbol:finType}{rho:Rho symbol}
(n:nat)(stk:@sticker symbol rho):seq domino:=
match n with
|0 => start stk
|S n' =>
  let A' := ss_generate_prime n' stk in
  let A_wk := [seq a <- A'|is_wk a] in
  let A_nwk := [seq a <- A'|~~ is_wk a] in
  undup(A_wk++filter_option[seq mu' a d|a<-A_nwk,d <- (extend stk)])
end.
```

DNAドミノ(3/3)　Coqで定義

```
(*二本鎖部分の構造体を定義
文字列情報を持ち、空で無いこと、対象関係を満たしていることを要求する*)
Structure wk{symbol:finType}{rho:Rho symbol} := Wk{
  str : seq (symbol*symbol);
  nilP : str <> nil;
  rhoP : all(fun p=>p\in rho)str
}.
(*粘着末端の定義　上側鎖かどうかと文字列の情報を持ち、文字が空でないことを要求する*)
Structure stickyend{symbol:finType}:= Se{
  is_upper : bool;
  end_str : seq symbol;
  end_nilP : end_str <> nil
}.
(*ドミノを定義　二本鎖部分と粘着末端、もしくは空白部の組み合わせからなる*)
Inductive domino{symbol:finType}{rho:Rho symbol}:=
|null : domino
|Simplex : @stickyend symbol -> domino
|WK : @wk symbol rho -> domino
|L : @stickyend symbol -> @wk symbol rho -> domino
|R : @wk symbol rho -> @stickyend symbol -> domino
|LR : @stickyend symbol -> @wk symbol rho -> @stickyend symbol -> domino.
```

証明(REG_RSL.v)

```
Definition Aut_to_Stk{state symbol:finType}(M:@automaton state symbol):=
let A1 := filter_option[seq wkaccept M s|s<-language'(#|state|.+1)symbol] in
let A2 := [seq startDomino M s|s <- language(#|state|.+1)symbol] in
let D1 := [seq extentionDomino M s t|s<-language(#|state|.+1)symbol,
                                    t<-language'(#|state|)symbol] in
let D2 := filter_option[seq stopDomino M t s|
  s <- language'(#|state|.+1)symbol,t <- language'(#|state|)symbol] in
{|start:=(A1++A2);extend:=(D
1++D2);startP:=st_correctP M|}.
```

```
Theorem REG_RSL{state symbol:finType}(M:@automaton state symbol)(s:seq symbol)
(m:nat):s <> nil -> exists n:nat , n <= m ->
accept M s = (s\in(ss_language_prime m (Aut_to_Stk M))).
```

オートマトン vs スティッカー(7/7)

約2800行からなる形式証明の全文
証明の正しさはCoqが保証するため、
手動で確かめる必要はない

```
Theorem REG_RSL{state symbol:finType}(M:@automaton state symbol)(s:seq symbol)
(m:nat):s <> nil -> exists n:nat , n <= m ->
accept M s = (s\in(ss_language_prime m (Aut_to_Stk M))).
```

補題集(myLemma.v)