Chomsky-Grammatik als DNA-Algorithmus: Menge aller Quadratzahlen

Ruman Gerst, Marie Lataretu

Gliederung

1. Einführung
2. Grammatik
3. Beispiel
4. Korrektheit
5. Super-DNA-Mode
6. Fazit

Was ist eine (formale) Grammatik?

- 4er-Tupel: G = (S, N, T, R)
  - S … Startsymbol
  - N … endliche Menge von Nichtterminalsymbole
  - T … endliche Menge von Terminalsymbole
  - R … endliche Menge von Produktionsregeln

- Menge, der von G erzeugten Wörter …

  (formale) Sprache L(G)

Kleines Beispiel

Grammatik G =

(

S,

N = {S, M},

T = {a, b},

R

)

R =

{

(1) $S \rightarrow aMb$

(2) $M \rightarrow aMb \mid ab$

}

G erzeugt Wörter der Form

$a^n b^n$

Aufgabe

Gesucht:

Grammatik G, die alle Quadratzahlen der Form a, aaaa, aaaaaaaaa, … erzeugt.

Hier: a, $aaaa$, $aaaaaaaaa$, …

Notation: $a^n = \underbrace{a \ldots a}_{n \text{ Mal}}$

Grammatik

Grammatik G =

(

S,

N = {S, U, L, R},

T = {a, $},

R

)

R =

{

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

}

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

S

(1) → U

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

S

(1) →U

(2a) → RUL

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

S

(1) → U

(2a) → RUL

(2b) → $RRLL$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$RRLL$

(3) →$RLRaL$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$RRLL$

(3) →$RLRaL$

(3, 4) → $LRaaRL$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$RRLL$

(3) →$RLRaL$

(3, 4) → $LRaaRL$

(7, 4, 3) → $aRaLRa$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$RRLL$

(3) →$RLRaL$

(3, 4) → $LRaaRL$

(7, 4, 3) → $aRaLRa$

(4, 6) → $aaRLaR$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$aaRLaR$

(3) → $aaLRaa$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$aaRLaR$

(3) → $aaLRaa$

(5, 4) → $aLaaRa$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$aaRLaR$

(3) → $aaLRaa$

(5, 4) → $aLaaRa$

(5, 4) → $LaaaaR$

Beispiel

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

$aaRLaR$

(3) → $aaLRaa$

(5, 4) → $aLaaRa$

(5, 4) → $LaaaaR$

(7, 6) → $aaaa$

Korrektheit

Jedes Wort der Form a^m, $m \in \{n^2 | n \in \mathbb{N}\}$, ist erzeugbar.

1 mal	(1) S	
n-1 mal	(2) U → RUL	
1 mal	(2) U → RL	
n * n mal	(3) RL → LRa	jedes R (n Stück) muss ein L (n Stück) „übergehen“
dabei	(4) Ra → aR (5) aL → La (6) R$ → $ (7) $L → $	

Korrektheit

G erzeugt nur Wörter der Form a^m, $m \in \{n^2 | n \in \mathbb{N}\}$.

- Nur RL → LRa erzeugt a‘s und das n * n Mal
- R kann a nur nach rechts überholen
- L kann a nur nach links überholen
- R bzw. L „verschwinden“, wenn sie alle L‘s bzw. R‘s überholt haben

Praxis

- Implementierung der Grammatik in Python
- Algorithmus

  Wiederhole bis nur noch Terminalsymbole:

  - Wähle zufällige Regel
  - Wähle zufällige Position im aktuellen Wort
  - Wenn anwendbar → Regel anwenden

Bringe „DNA“ hinein

- ✔ Grammatik
- ✔ Grammatik korrekt
- ✔ Funktioniert auch praktisch
- → Jetzt fehlt noch die DNA!

Anwendung auf DNA

- Ziel: Grammatik nur mit ACGT
- Hintergrund: z.B. CRISPR/Cas kann *gezielt* bestimmte DNA durch andere *ersetzen*
- Probleme?
  - 6 Symbole in Grammatik, aber nur 4 Nukleotide
  - Wie verhindert man ungewollte Regeln?

Anwendung auf DNA: Lösung

- Spacer-Sequenzen + „Kodierende“ Sequenz

  AAXXXTT, X∈{G, C}

- A's und T's trennen Symbole + geben Richtung

  ➔ Keine ungewollte Regeln

- G's und C's kodieren binär die Information

- Ausnahme: „Zählsymbol“ = C

  ➔ Immer zwischen zwei Symbolen

  AACGCTT CCCCCCC AACCCTT

Grammatik

Grammatik G =

(

S,

N = {S, U, L, R},

T = {a, $},

R

)

R =

{

(1) S → U

(2) U → RUL | RL

(3) RL → LRa

(4) Ra → aR

(5) aL → La

(6) R$ → $

(7) $L → $

}

Grammatik auf DNA

Grammatik G =

(

AAGGGTT,

N = {AAGGGTT, AAGGCTT, AAGCCTT, AAGCGTT},

T = {AACCCTT, C},

R

)

R =

{

(1) AAGGGTT → AACCCTTAAGGCTTAACCCTT

(2) AAGGCTT → AAGCCTTAAGGCTTAAGCGTT | AAGCCTTAAGCGTT

(3) AAGCCTTAAGCGTT → AAGCGTTAAGCCTTC

(4) AAGCCTTC → CAAGCCTT

(5) CAAGCGTT → AAGCGTTC

(6) AAGCCTTAACCCTT → AACCCTT

(7) AACCCTTAAGCGTT → AACCCTT

}

Praxis

- Sieht schlimm aus, funktioniert aber!
- Mit dem gleichen Algorithmus wie vorher ausgeführt

Fazit

- Grammatik
- Grammatik funktioniert auch für DNA
  - Grammatik erweiterbar
- Gut parallelisierbar
- Ausblick: medizinische Anwendung?

Vielen Dank für Eure Aufmerksamkeit!

Fragen