Rozpoznawanie drugorzędowych struktur RNA za pomocą języków formalnych

Jacek Pospychała (66301), Katarzyna Rafalska (66305)

7 czerwca 2006

1 Streszczenie

Jedną z niezwykłych cech charakteryzujących RNA jest zdolność do sklejania się ze sobą jego komplementarnych podsekwencji. Podjęliśmy próbę formalnego zdefiniowania reguł dopasowania, tak by automatycznie identyfikować struktury. Opierając się na teorii języków formalnych zaproponowanej przez Noama Chomsky'ego (przypis), zdefiniowaliśmy szereg gramatyk. Korzystając z prologa zdefiniowaliśmy bazę wiedzy odkrywającą w liniowych sekwencjach złożone struktury. W dalszym kroku, dla wybranych organizmów, udało się nam także wygenerować całe złożone struktury i porównać z wynikami innych.

2 Charakterystyka drugorzędowych struktur RNA

Badana struktura liniowa to (1) ciąg znaków A C G U, (2) reprezentujący łańcuch nukleotydów kodujących RNA dowolnego organizmu. A więc nie każdy ciąg znaków można nazwać RNA, a tylko taki który rzeczywiście istnieje w naturze. Pomijając założenie (2), gramatyka regularna definiująca RNA, to:

$$s \to aS \qquad S \to cS \qquad S \to gS \qquad S \to uS \qquad S \to \varepsilon$$

(ε reprezentuje brak znaku. Bez ostatniej reguły ciąg musiałby być nieskończenie długi.) Przy obecnej wiedzy nie jesteśmy w stanie w pełni zdefiniować gramatyki RNA, uwzględniając wszystkie powyższe założenia. Do badań wykorzystaliśmy więc zasoby biblioteki sekwencji GenBank, należącej do NCBI [1].
Na podstawie łańcucha RNA można wyznaczyć jego strukturę drugorzędową. Jest to struktura powstająca w wyniku lokalnych oddziaływań komplementarnych ze sobą podsekwencji łańcucha. Jeden łańcuch może wyznaczać wiele struktur 2D. Korzystając z przykładowych danych doszliśmy do następujących założeń:

1. Łańcuchy dopasowujące się do siebie to taka para A, B, że B jest odwróceniem łańcucha komplementarnego do A.
2. Dopasowujące się do siebie łańcuchy komplementarne mają długość conajmniej 3bp. Zazwyczaj są długości 3-6bp.
3. Za maksymalną długość dopasowań przyjęto 20bp.
4. Łańcuchy komplementarne łączące się ze sobą są nie bliżej niż w odstępie 4bp.

Z założeń tych wynika że najprostsza struktura 2D ma długość conajmniej 10bp i jest postaci XAY, gdzie: X komplementarne do Y, X o długości 3bp, A o długości 4bp. Np. *acguacacgu*

3 Gramatyki języków formalnych

Noam Chomsky zaproponował w [1] klasyfikację języków formalnych. Wyróżnił trzy grupy, języki o gramatykach: regularnych, bezkontekstowych i kontekstowych. Gramatyka regularna, to taka że po prawej stronie produkcji występuje zawsze nie więcej niż jeden znak nieterminalny. Głównym narzędziem definiowania takich gramatyk są wyrażenia regularne (regexp), a przykładem jest gramatyka z 2 pkt. pracy. Gramatyka bezkontekstowa charakteryzuje się tym,

[1] http://www.ncbi.nih.gov/Genbank/index.html

że po lewej stronie produkcji występuje dokładnie jeden znak nieterminalny, a po prawej dowolny ciąg terminalnych i nieterminalnych. Dla przejżystości notacji, znaki terminalne oznacza się małymi, natomiast nieterminalne wielkimi literami. Wymieniona już wcześniej gramatyka, w tej klasyfikacji jest gramatyką regularną, równoważną wyrażeniu $[aucg]*$.
Zakładając że najprostsza struktura drugorzędowa spełnia tylko 1 z wymienionych wcześniej założeń, tj. "łańcuchy dopasowujące się do siebie to taka para A, B, że B jest odwróceniem łańcucha komplementarnego do A", możne spróbować określić gramatykę definiującą taką strukturę jako:

$$s \to aSu \qquad S \to cSg \qquad S \to gSc \qquad S \to uSa \qquad S \to \varepsilon$$

Wówczas np. ciąg $acgu$ jest poprawnym zdaniem tej gramatyki. Jednak na drodze od gramatyki formalnej do weryfikacji konkretnych zdań jest jeszcze jeden krok - implementacja parsera. Chociaż istnieje wiele generatorów parserów dla gramatyk bezkontekstowych, powyższy przypadek nie jest prosty, bo gramatyka jest niejednoznaczna. W szczególności dla przykładowego łańcucha wejściowego cg, po napotkaniu znaku c, parser oczekiwałby dowolnego znaku wynikającego z S w regule cSg (a więc dowolnego z a, c, g, u), lub oczekiwałby znaku g, zakładając że $S = \varepsilon$. Powstała niejednoznaczność, czy g na wejściu, to początek S, czy koniec reguły, jest nie do rozstrzygnięcia dla parsera pobierającego z wejścia każdorazowo 1 token. Choć możnaby wykorzystać parser zawsze czytający na wejściu nie 1, a 2, 3 lub w ogólności k tokenów, to problem powróci zawsze dla sekwencji analogicznej do przykładu, ale o długości $k + 1$. Szczęśliwie, definicja języka nie wymaga istnienia parsera. A by zdanie należało do tego języka, musi jedynie istnieć taki ciąg wykonań reguł produkcji, że rozpoczynając od symbolu startowego, możliwe jest doprowadzenie do danego zdania (generatywność gramatyki). Pomijając więc problem fizycznej wykonalności przedstawionych gramatyk, można zaproponować gramatyki uwzględniające pozostałe odkryte wcześniej zależności. Oto przykładowa gramatyka:

$$\begin{aligned}
&S \to CBC\\
&B \to aBu\\
&B \to cBg\\
&B \to gBc\\
&B \to uBa\\
&B \to DDDC\\
&C \to DC\\
&C \to \varepsilon\\
&D \to a\\
&D \to c\\
&D \to g\\
&D \to u
\end{aligned}$$

Gramatyka ta definiuje struktury jak na rysunku 1. Znaki nieterminalne reprezentują:

S znak startowy gramatyki: dowolny łańcuch (C), dopasowanie (B), dowolny łańcuch(C).

B dopasowanie razem z łańcuchem niekomplementarnym o długości minimum 3 znaków (DDD) w środku.

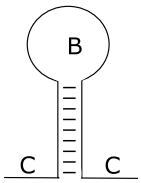
C lista nukleotydów o dowolnej (także zerowej) długości

D dowolny nukleotyd

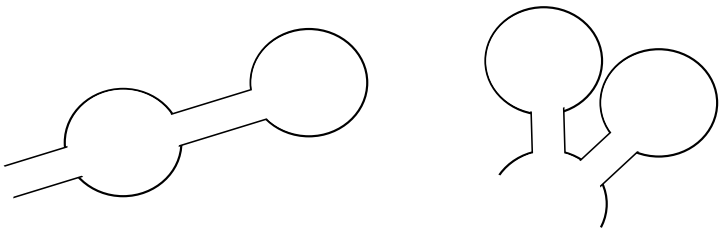
3.1 Gramatyka struktur złożonych

By gramatyka obejmowała także bardziej złożone struktury, jak np. zagnieżdżenia, czy ciągi kolejnych dopasowań (rys. 2, można ją rozbudować, np. dodając jeden znak nieterminalny, reprezentujący najprostsze dopasowanie. W poniższej gramatyce tym znakiem jest A. Nieterminal startowy produkuje listę nieterminali A, których definicja bazuje na poprzednim przykładzie. Jedyną zmianą jest fakt, że obok poprzedniej reguły $B \to DDDC$ jest $B \to S$, a więc "wewnątrz" jednego dopasowania może się znaleźć kolejne, a nawet cała lista.

$$S \to AS \tag{1}$$
$$S \to A \tag{2}$$



Rysunek 1: Dopasowanie w łańcuchu, uwzględniające typowe występowanie podsekwencji niekomplementarnych. Litery reprezentują znaki nieterminalne, które dopasują się do fragmentów sekwencji, zgodnie z powyższą gramatyką.



Rysunek 2: Typowe formy tworzące złożone struktury drugorzędowego RNA.

$$A \to CBC \tag{3}$$
$$B \to aBu \tag{4}$$
$$B \to cBg \tag{5}$$
$$B \to gBc \tag{6}$$
$$B \to uBa \tag{7}$$
$$B \to S \tag{8}$$
$$B \to DDDC \tag{9}$$
$$C \to DC \tag{10}$$
$$C \to \varepsilon \tag{11}$$
$$D \to a \tag{12}$$
$$D \to c \tag{13}$$
$$D \to g \tag{14}$$
$$D \to u \tag{15}$$

Wady powyższego zapisu są następujące: ciągle nie uwzględniają minimalnej długości dopasowań, oraz wykluczają struktury tzw. pseudoknot, tj. np. takie *aaccuugg*, czyli gdy ciągi dopasowują się na przemian ($ABAB$).

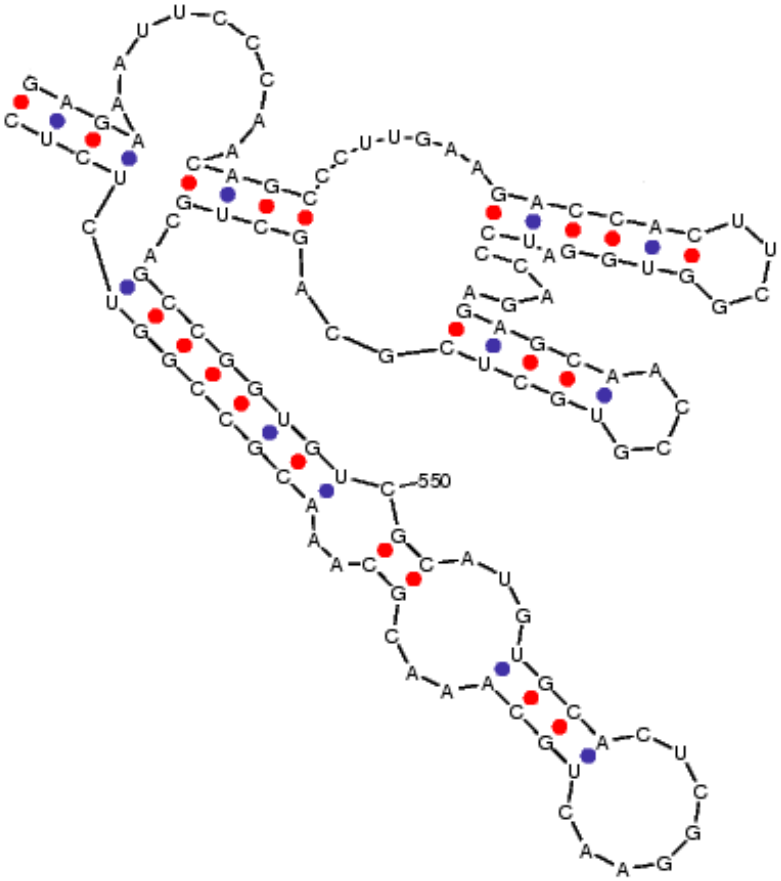
3.2 Przykład

Rozważmy przykład jak na rysunku 3. Łańcuch tworzący tą strukturę to:

```
GAGAAAUUCC CAACAGCCCU UGAAGACCAC UUCGGUGGUC
CAGAGAGCAA CCGUGCUCGC AGCUGCAGCC GGUGUCAUGU
GCACUCGGAA CUGCAAACGC AAACGCCGGU CUCUC
```

Poniżej przedstawiono ciąg produkcji ostatniej gramatyki, tworzących strukturę taką jak na rysunku. Znakiem startowym jest S, od niego zaczynamy. Numer po lewej stronie to numer zastosowanej reguły.

$$\begin{aligned} &(2) \quad S \to A \\ &(3) \quad A \to CBC \\ &(11) \quad CBC \to BC \end{aligned} \tag{16}$$

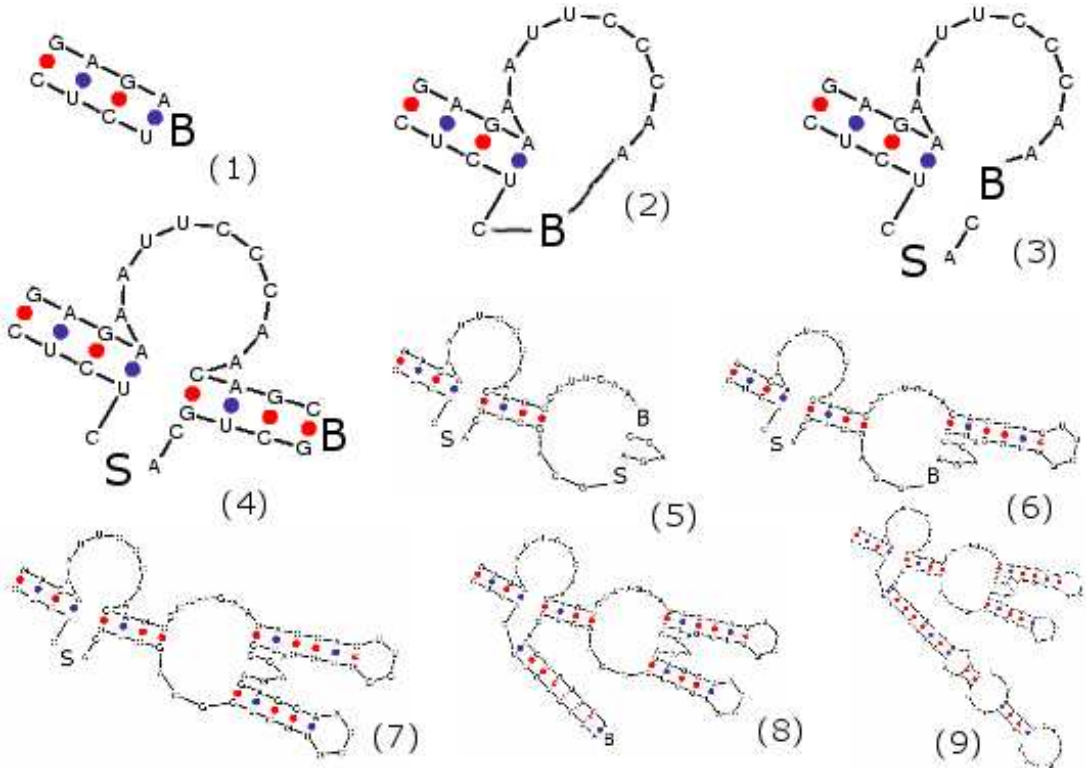


Rysunek 3: Przykładowe dopasowanie.

$$
\begin{aligned}
(11)\quad & BC \to B \\
(6)\quad & B \to gBc \\
(4)\quad & gBc \to gaBuc \\
(6)\quad & gaBuc \to gagBcuc \\
(4)\quad & gagBcuc \to gagaBucuc
\end{aligned}
\tag{17}
$$

W pierwszych ośmiu krokach dopasowane zostały do siebie pierwsze 4 i ostatnie 4 znaki łańcucha. Dla uproszczenia zapisu, w dalszych krokach pominiemy wyprodukowane już ciągi *gaga* i *ucuc*. Tak więc symbolem pozostającym po lewej stronie produkcji jest nieterminal B. Dopasowywany ciąg zaczyna się od 4 znaku i kończy 4 znaki przed końcem. Sytuację tą ilustruje rysunek 4.1. Kontynuując od pozostałego B:

$$
\begin{aligned}
(8)\quad & B \to S \\
(2)\quad & S \to A \\
(3)\quad & A \to CBC \\
(10)\quad & CBC \to DCBC \\
(12)\quad & DCBC \to aCBC \\
(10,12)\quad & aCBC \to aaCBC \\
(10,12)\quad & aaCBC \to aaaCBC \\
(10,16)\quad & aaaCBC \to aaauCBC \\
(10,16)\quad & aaauCBC \to aaauuCBC \\
(10,13)\quad & aaauuCBC \to aaauucCBC
\end{aligned}
\tag{18}
$$



Rysunek 4: Przykład. Etapy produkcji.

$$
\begin{array}{cl}
(10,13) & aaauucCBC \rightarrow aaauuccCBC \\
(10,13) & aaauuccCBC \rightarrow aaauucccCBC \\
(10,12) & aaauucccCBC \rightarrow aaauucccaCBC \\
(10,12) & aaauucccaCBC \rightarrow aaauucccaaCBC \\
(11) & aaauucccaCBC \rightarrow aaauucccaaBC \\
(10.13) & aaauucccaBC \rightarrow aaauucccaaBcC \\
(11) & aaauucccaBCcC \rightarrow aaauucccaaBc
\end{array} \tag{19}
$$

Kolejna seria kroków produkuje łańcuch nie dopasowanych znaków $aaauucccaa$. Całość dopasowanej sekwencji to (rysunek 4.2):

`gagaaauucccaaBcucuc`

Ponownie, nie będziemy pisać już wyprodukowanych sekwencji, a jedynie to, co może jeszcze powstać z jedynego znaku nieterminalnego - B.

$$
\begin{array}{cl}
(1) & B \rightarrow AS \\
(3) & AS \rightarrow CBCS \\
(11) & CBCS \rightarrow BCS \\
(10) & BCS \rightarrow BDCS \\
(13) & BCS \rightarrow BcCS \\
(10) & BcCS \rightarrow BcDCS \\
(12,11) & BcDCS \rightarrow BcaS
\end{array} \tag{20, 21}
$$

Sekwencja ac to sekwencja rozdzielająca dwie złożone struktury, z których jedna zaczyna się dopasowaniem $cagc$, a druga $gccggugu$. Ponieważ z ciągu $BcaS$, jedna część pozostałej struktury może być wyprodukowana z B, a druga z S, zatem można je dalej rozpatrywać osobno (rysunek 4.3). Bieżąca postać całej sekwencji:

`gagaaauucccaaBcaScucuc`

Rozpatrujemy tylko B. Zauważmy, że stosując ciąg produkcji analogiczny do tego w krokach (16-17), dokonamy przejścia $B \to cagcBgcug$ (rys. 4.4), natomiast postępując podobnie jak w krokach (18-19), $B \to ccuugaaBgca$. Cała sekwencja:

`gagaaauucccaacagcccuugaaBgcagcugcaScucuc`

Ponownie ograniczymy się jedynie do produkcji wynikających z B.

$$\begin{array}{cl} (8) & B \to S \\ (1) & S \to AS \\ (3) & AS \to CBCS \\ (11) & CBCS \to BCS \\ (10,13,10,13,10,12,10,14,10,12,11) & BCS \to BccagaS \end{array} \tag{22, 23}$$

Całość (rys. 4.5):

`gagaaauucccaacagcccuugaaBccagaSgcagcugcaScucuc`

Produkcje wynikające z pierwszego nieterminala (B):

$$\begin{array}{cl} (6) & B \to gBc \\ (4) & gBc \to gaBuc \\ (5) & gaBuc \to gacBguc \\ (5,4,5) & gacBguc \to gaccacBgugguc \\ (9,10,11) & gaccacBgugguc \to gaccacDDDDgugguc \\ (15,15,13,14) & gaccacDDDDgugguc \to gaccacuucgguggguc \end{array} \tag{24, 25}$$

Całość, wstawiając w miejsce nieterminala, wyprodukowaną sekwencję wynikową (rys. 4.6):

`gagaaauucccaacagcccuugaagaccacuucggugguccccagaSgcagcugcaScucuc`

Postępując analogicznie, jak w krokach między (24) a (25), pierwsze lewe S produkuje $gagcaaccgugcuc$. Tym samym całość ma postać (rys. 4.7):

`gagaaauucccaacagcccuugaagaccacuucggugguccccagagagcaaccgugcucgcagcugcaScucuc`

Produkcje z pozostałego ostatniego nieterminala S (rys. 4.8):

$$\begin{array}{cl} (2,3,11,11) & S \to B \\ (x,5,5,6,6,x,6,7) & B \to gccgguguBacgccggu \end{array} \tag{26, 27}$$

Należy zwrócić uwagę, że dopasowanie na rysunku 3 przewiduje także sklejenia $u - g$. Wynik dopasowania pochodzi z innego programu, przyjęliśmy na wiarę taki rodzaj sklejenia i odpowiednią regułę produkcji oznaczyliśmy x. Kontynuując produkcje z pozostałego B:

$$\begin{array}{cl} (8,2,3) & B \to CBC \\ (10,11,13,10,10,11,12,12) & CBC \to cBC \to cBaa \\ (6,5) & cBaa \to cgcBgcaa \\ (\ldots) & cgcBgcaa \to cgcaugBaacgcaa \\ (7,6,5,4) & cgcaugBaacgcaa \to cgcaugugcaBugcaaacgcaa \\ (\ldots) & cgcaugugcaBugcaaacgcaa \to cgcaugugcacucggaacugcaaacgcaa \end{array} \tag{28, 29}$$

Tym samym rozpoczynając od znaku nieterminalnego S, osiągnięto stan, gdzie są same znaki terminalne, co więcej tworzą one łańcuch taki sam, jak łańcuch przedstawiony na rysunku 3:

`gagaaauucccaacagcccuugaagaccacuucggugguccagagagcaaccgugcuc`
`gcagcugcagccggugucaugugcacucggaacugcaaacgcaaacgccggucucuc`

3.3 Abstrakcyjna gramatyka struktur złożonych

W pracy nad gramatykami opisującymi RNA, szczególnie w trakcie poszukiwań najbardziej elementarnych struktur, doszliśmy do wniosku, że można opracować także zupełnie inną koncepcję gramatyki. Definiuje ona na zupełnie innym poziomie abstrakcji złożone układy dopasowań i przypomina trochę notację matematyczną. Zauważając że łańcuch składa się z ciągów komplementarnych (po dwa na jedno dopasowanie), oraz nie komplementarnych, można je oznaczyć jako nawiasy (otwierający - pierwszy ciąg, zamykający - drugi ciąg komplementarny), oraz podkreślenia - sekwencje nie komplementarne. Przykładowo zapis _(_)_ ilustruje sekwencję z rys. 1, a _(_(_)_)_ i _(_)_(_)_ odpowiednio sekwencje z z rys. 2. Jednoznaczna gramatyka obejmująca takie sekwencje ma postać:

$$S \to SA \qquad S \to A \qquad A \to \_ \qquad A \to (S)$$

Powstaje jednak problem oceny jaka sekwencja jest częścią dopasowania, a jaka nie łączy się z żadną inną. Choć nie jest to oczywiste, również struktury typu pseudoknot mogą być reprezentowane w takiej gramatyce. Wynika to faktu, że dopasowanie reprezentowane jedynie przy pomocy nawiasów traci swoją jednoznaczność. Wcześniej każda para ciągów komplementarnych wiązały się jednoznacznie tylko ze sobą. W przypadku nawiasów, zapis (()) może reprezentować zarówno sekwencję *acgu* (dopasowanie $ABBA$), jak i *acug* (dopasowanie $ABAB$). W teorii języków takie zachowanie nie jest niczym nowym, gdyż często jedna struktura gramatyczna może przyjmować wiele znaczeń (np. angielskie "She saw the man with the telescope").

4 Zastosowanie programowania deklaratywnego

Teoria języków formalnych rozwijana na potrzeby informatyki, a w szczególności kompilatorów, kładła zawsze nacisk na jednoznaczność i scisłą strukturalizację gramatyk. Jednym z narzędzi pozbawionych takich barier jest Prolog - język programowania logicznego. Program tworzą tu reguły wnioskowania oraz celu rozumowania (przypominające reguły produkcji). Rola komputera polega natomiast na takim zastosowaniu tych reguł aby znaleźć rozwiązanie. Zamiast sekwencji przetwarzane są listy obiektów (w naszym przypadku obkiety to a, c, g, u). Można szukać jeszcze wielu podobieństw, natomiast zasadniczą różnicą jest sposób działania. O ile parsery "pochłaniają" łańcuch wejściowy i operują na stosie tokenów, o tyle prolog działa na przestrzeni stanów, takiej że w każdej chwili możliwe jest przerwanie przetwarzania i cofnięcie się do dowolnego wcześniejszego wyniku (backtracking). Zależnie od obszaru nauki, w którym wykorzystywane jest programowanie deklaratywne, program prologowy nazywa się bazą wiedzy, regułami wnioskowania, czy bazą faktów. Pojęć tych będziemy używać zamiennie. Elementarną wiedzą, jaką należy zdefiniować jest pojęcie komplementarności. W notacji prologa są to fakty:

$$cmpltry(a, u).$$
$$cmpltry(u, a).$$
$$cmpltry(c, g).$$
$$cmpltry(g, c).$$

Zatem zapytanie, co jest komplementarne z a: $cmpltry(a, X)$, zwróci odpowiedź $X = u$, na podstawie pierwszej reguły. Definicja komplementarności dla listy (sekwencji) składa się z reguł:

$$cmpltry([H1|T1], [H2|T2]) : -cmpltry(H1, H2), cmpltry(T1, T2).$$
$$cmpltry([H1], [H2]) : -cmpltry(H1, H2).$$

Jest to definicja rekurencyjna. Druga reguła to warunek końcowy rekurencji, definiuje fakt, że jednoelementowe listy $[H1]i[H2]$ składają się z elementów ($H1$ i $H2$), które są komplementarne. Pierwsza reguła definiuje fakt, że pierwsze elementy obu list są komplementarne, oraz "ogony" (tails) są także komplementarne. Reguł prologa można używać na dwa sposoby, aby zweryfikować czy zdanie jest prawdziwe, oraz aby odszukać rozwiązanie. Np. odpowiedź na zapytanie $cmpltry([a, c, u], [u, g, a])$ to "Yes" - listy są komplementarne. Odpowiedź na zapytanie $cmpltry([a, c, u], X)$ to $X = [u, g, a]$. Z kolei na pytanie $cmpltry(X, Y)$ będzie nieskończenie wiele odpowiedzi.
Poszukując rozwiązania ponownie staraliśmy się zdefiniować taki zbiór reguł, który najpierw identyfikowałby najprostsze pojedyńcze dopasowanie, a następnie na tej podstawie tworzyć bardziej kompletne definicje. Najbardziej intuicyjna jest koncepcja, że dopasowanie to taka lista, w której pewien charakterystyczny ciąg występuje dwa razy.

$$equalheads([X|T1], [X|T2], Y) : -atom(X), equalheads(T1, T2, Y).$$
$$equalheads([], X, X).$$

$$contains([X|T1],[X|T2],T3) :- equalheads(T1,T2,T3).$$
$$contains([X|T1],[\_|T2],T3) :- contains([X|T1],T2,T3).$$

$$bubble(S) :- contains(A,S,Tail), contains(A,Tail,\_).$$

Początkowe reguły mają charakter pomocniczy. Ostatnią regułę można interpretować w sposób następujący: jeżeli na liście S występuje podlista A, taka że $Tail$ stanowi ciąg dalszy elementów S bezpośrednio po A ($contains(A,S,Tail)$), oraz w $Tail$ także występuje A ($contains(A,Tail,\_)$), to jest dopasowanie w S ($bubble(S)$ jest spełnione). Reguła $equalheads(A,B,C)$ jest spełniona, jeżeli początki list A i B są takie same. Lista B może być dłuższa, wówczas jej dalszy ciąg jest zawarty w C.
Reguły można wzbogadzić o wcześniej zdefiniowaną własność komplementarności, tak by drugi ciąg nie był taki sam, tylko komplementarny do pierwszego. Ponadto konieczne jest uwzględnienie że drugie wystąpienie będzie w odwróconej kolejności, oraz że dopasowanie będzie dostatecznej długości, a odstęp między dopasowaniami także nie zerowy. Główna reguła w najbardziej zaawansowanej postaci:

$$\begin{aligned} fold(S,Result,N) \;:-\; & contains(A,S,Pref1,Postf1),\\ & atleast(4,A),\\ & cmpltry(A,ComplA), reverse(ComplA,ComplRevA),\\ & contains(ComplRevA,Postf1,Pref2,Postf2),\\ & atleast(4,Pref2),\\ & join(Pref1,[[N,A]],Pref2,[[N,ComplRevA]],Postf2,R),\\ & NewNisN+1,\\ & fold(R,Result,NewN).\\ fold(X,X,\_). & \end{aligned}$$

Reguła ta w argumencie *Result* przechowuje wynik dopasowania, w postaci listy. Elementy komplementarne do siebie stanowią podlisty wraz z identyfikatorem. Na podstawie identyfikatora w przyszłości łatwo będzie odnaleźć dopasowane do siebie podciągi. Np. dla sekwencji $[a,g,g,g,c,a,u,g,c,c,c,g,a]$, $Result = [a,[0,[g,g,g]],c,a,u,g,[0,[c,c,c]],g,a],\ldots]$. A więc dopasowano jedną sekwencję (przydzielając numer 0) *ggg* do *ccc*. Prolog znajdzie zbiór wszystkich możliwych rozwiązań spełniających reguły.
Powyższa reguła zawiera także rozwiązanie problemu tzw. pseudoknotów. Jeżeli zostanie znalezione jakieś dopasowanie (a więc po prawej stronie spełnione będą predykaty *contains*), należące do niego łańcuchy zostaną oznaczone, a cała sekwencja jeszcze raz rekurencyjnie przetworzona. Tym samym w którejś z interacji jeden z ciągów nowego dopasowania może się znaleźć pomiędzy już dopasowanymi fragmentami, a drugi poza nimi.

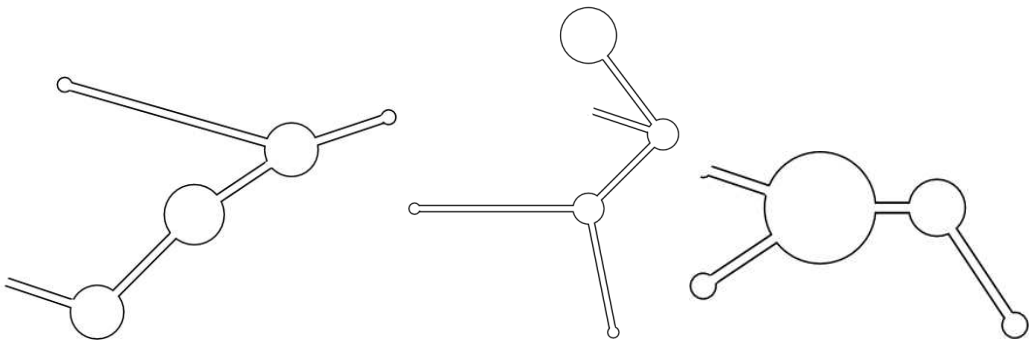
4.1 Przykład

Korzystając z tej reguły przetworzono przykłady prawdziwych sekwencji RNA z bazy danych GenBank. Ponownie wykorzystano przykładowy łańcuch znaków należący do organizmu z królestwa *Archea*, przedstawiany już na rysunku 3. Z kolei na podstawie otrzymanych rozwiązań wygenerowano rysunki dopasowań. Do generowania rysunków posłużyła gramatyka definiująca rozwiązania prologowe wraz z prostym kodem w C. Poniżej natomiast przedstawiono zapytanie wraz z trzema przykładowymi odpowiedziami z prologa oraz wygenerowane na ich podstawie rysunki poglądowe.

```
?- fold([g,a,g,a,a,a,u,u,c,c,c,a,a,c,a,g,c,c,c,u,u,g,a,a,g,a,c,c,a,c,u,u,c,g,g,u,g,
g,u,c,c,a,g,a,g,a,g,c,a,a,c,c,g,u,g,c,u,c,g,c,a,g,c,u,g,c,a,g,c,c,g,g,u,g,u,c,a,u,g,
u,g,c,a,c,u,c,g,g,a,a,c,u,g,c,a,a,a,c,g,c,a,a,a,c,g,c,c,g,g,u,c,u,c,u,c], Result).

Result = [[1857,[g,a,g,a]],a,a,[6109,[u,u,c,c]],c,a,a,[5019,[c,a,g,c]],c,c,u,u,[7161,
[g,a,a,g]],a,c,c,a,[7161,[c,u,u,c]],g,g,u,g,g,u,c,c,a,g,a,[3993,[g,a,g,c,a]],a,c,c,g,
[3993,[u,g,c,u,c]],g,c,a,[5019,[g,c,u,g]],c,a,g,c,c,g,g,u,g,u,c,a,u,g,u,g,c,a,c,u,c,
[6109,[g,g,a,a]],c,u,g,c,a,a,a,c,g,c,a,a,a,c,g,c,c,g,g,[1857,[u,c,u,c]],u,c];

Result = [[5430,[g,a,g,a]],a,a,u,u,c,c,c,a,a,[2546,[c,a,g,c]],c,c,u,u,g,a,a,[2541,
[g,a,c,c,a]],c,u,u,c,g,g,[2541,[u,g,g,u,c]],c,a,g,a,[8941,[g,a,g,c,a]],a,c,c,g,[8941,
```

```
[u,g,c,u,c]],g,c,a,[2546,[g,c,u,g]],c,a,g,[2670,[c,c,g,g]],u,g,u,c,a,u,g,u,g,c,a,c,u,
c,g,g,a,a,c,u,g,c,a,a,a,c,g,c,a,a,a,c,g,[2670,[c,c,g,g]],[5430,[u,c,u,c]],u,c];

Result = [[5730,[g,a,g,a]],a,a,u,u,c,c,c,a,a,[758,[c,a,g,c]],c,c,u,u,g,a,a,[9069,
[g,a,c,c]],a,c,u,u,c,g,g,u,[9069,[g,g,u,c]],c,a,g,a,g,[3500,[a,g,c,a]],a,c,c,g,[3500,
[u,g,c,u]],c,g,c,a,[758,[g,c,u,g]],c,a,g,c,c,g,g,u,g,u,c,a,u,g,[2568,[u,g,c,a]],c,u,
c,g,g,a,a,c,[2568,[u,g,c,a]],a,a,c,g,c,a,a,a,c,g,c,c,g,g,[5730,[u,c,u,c]],u,c]
```

Graficzną ilustrację trzech wymienionych wyników przedstawia rysunek 4.1. Są to tylko poglądowe szkice, pozwalające ocenić ilość nie dopasowanych sekwencji (im więcej, tym większe okręgi), strukturę rozgałęzień i proporcje długości dopasowanych sekwencji (po długości linii równoległych).



Rysunek 5: Poglądowe rysunki struktur zaproponowanych przez prologa.

5 Wnioski

Mimo silnych podstaw teoretycznych, współczesne języki formalne wydają się być dosyć mocno ukierunkowane na wąski wycinek zagadnienia gramatyk bezkontekstowych. Istniejąca klasyfikacja tylko w bardzo ogólnym stopniu rozróżnia typy gramatyk. Zaproponowane przez nas gramatyki mieszczą się w definicji gramatyk bezkontekstowych, jednak są niejednoznaczne, przez co jakby zupełnie poza obiektem zainteresowania teorii parserów.
Definicja języka formalnego jest bardzo zwięzła, prosta i szeroka. Niczym nie ogranicza twórcy języka. Jednym z największych ograniczeń w naszym problemie był ubogi alfabet, składający się jedynie z czterech liter. O ile w innych językach, o alfabetach nieznacznie większych (do 30 znaków), elementarnymi jednostkami leksykalnymi są słowa, w analizie RNA trudno szukać porównywalnych struktur. Natomiast dopasowania przywodzą na myśl proste reguły gramatyczne, jednak ich poszukiwanie wymaga całościowego spojrzenia na sekwencję.
Zaproponowane gramatyki pozwalają reprezentować dowolne struktury RNA istniejące w świecie rzeczywistym. Odzwierciedlają hierarchię najprostszych elementów struktury (dopasowanie, niedopasowanie), figur kształtowanych z tych elementów (stem - najbardziej typowe sklejenie z sekwencją nie dopasowaną w środku), oraz złożonych struktur zbudowanych z tych figur (rekurencyjne zagnieżdżenia). Jest to dobra podstawa do szukania znaczenia tych struktur na wyższym poziomie abstrakcji.
Prolog, jako narzędzie dobrze radzące sobie z problemem niejednoznaczności, stanowi wygodną podstawę do implementacji użytecznych gramatyk, nie poddających się standardowym generatorom parserów. Nie zmniejsza złożoności obliczeniowej problemów, jego wewnętrzna implementacja poszukiwania rozwiązań jest bardzo ogólna i nie porównywalnie słabsza w porównaniu z algorytmami specjalizowanymi. Podobnie opracowane dla niego programy są dosyć generyczne, znajdują wszystkie struktury o zdefiniowanych przez nas cechach. Dalszy ich rozwój mógłby objąć optymalizację wyszukiwanych struktur, np. pod względem siły lub liczby powiązań. Podobnie, jak w przypadku języków formalnych, otwarta pozostaje także ścieżka dalszej analizy struktur znajdowanych przez program.

Literatura

[1] N. Chomsky: *Syntactic Structures*, Mouton, The Hauge, 1957

[2] D. B. Searls: *The language of genes*, Nature 420, 211-217, 2002