Chris Bielow
Institut für Informatik
Bioinformatics Solution Center

Praktikum: Algorithmen und Datenstrukturen in der Bioinformatik
7. Programmieraufgabe

Abgabe Do, 18.02., 23:59 Uhr per GIT

Kellerautomat für RNA Hairpins (10 Punkte)

In dieser Programmieraufgabe werden Sie einen deterministischen Kellerautomaten (Push-Down-Automaton (PDA)) implementieren, welcher RNA-Hairpins erkennt. Gegeben ist eine Headerdatei *PDA.hpp*, welche Sie wie üblich noch um private Member und Methoden erweitern sollen.

- Lesen sie im Hauptprogramm *aufgabe7_main.cpp* eine RNA Sequenz (als erstes Argument von der Kommandozeile) ein
- Instanzieren Sie ein PDA-Objekt und übergeben Sie zeichenweise die zu validierende Sequenz.
- Nach jedem Zeichen gibt der Automat seinen Zustand zurück (ACCEPT, FAIL, IN_PROGRESS)
- Das Hauptprogramm gibt 0 zurück wenn, ein RNA-Hairpin erkannt wurde; andernfalls 1. Geben Sie ebenfalls etwas auf der Konsole aus (ACCEPT, FAIL), bevor das Programm beendet wird.

Die Produktionsregeln der Grammatik sind im VL-Script auf S. 8066 bzw. VL-07 gegeben.

Denken Sie sich eine geeignete Datenstruktur als Klassenmember aus, um die Übergangsfunktionen zu speichern. Diese sollte im Konstruktor der Klasse befüllt werden. Das akzeptierte Alphabet ist a,c,g,u (Kleinschreibung). Alle anderen Symbole werden nicht akzeptiert und führen zu FAIL.
Hinweis: Als letztes Zeichen sollte ihrer Funktion immer ein zusätzliches $ übergeben werden, um das Ende der Eingabe zu signalisieren. Auf der Kommandozeile hingegen muss es ohne $ funktionieren.

```
$ ./aufgabe7_main gccgcaaggc
ACCEPT
```

Checken Sie *PDA.hpp, PDA.cpp und aufgabe7_main.cpp* ins GIT in den Unterordner *./aufgabe7/* ein.

Praktikumshinweise

- Im Header *PDA.hpp* sind die Funktionen genau beschrieben, auch evtl. exceptions die geworfen werden sollen.
- Implementieren Sie den Automaten auf jeden Fall als Kellerautomat (d.h. mit einem Stack) und mit Hilfe einer Datenstruktur die Regeln befolgt (also allgemein anwendbar ist). Andere Lösungen bringen keine Punkte. next() sollte voellig unabhaengig von der Sprache/Regeln sein, d.h. Unterscheidung zwischen Hairpin und Brackets darf dort nicht stattfinden. Auch die Erweiterung des Hairpins um ein Basenpaar muss allein durch Erweiterung der Regeln moeglich sein.
- Compilieren sie ihr Programm (und den Test) mit Hilfe des Makefiles (oder zumindest dessen compile flags) auf einem der Poolrechner.
- Am Montag zum Tutorium wird eine Test-Klasse online gestellt mit der Sie ihre Implementierung überprüfen können.

Zusatzaufgabe (3 Punkte)

Implementieren Sie einen zweiten Automaten in der gleichen PDA Klasse, welcher eine Sequenz von öffnenden und schliessenden runden Klammern validiert. Valide Worte sind Klammerausdrücke, in welchen die Anzahl der öffnenden und schliessenden Klammern übereinstimmen und an jedem Präfix des Ausdrucks die Anzahl der öffnenden Klammern grösser oder gleich der Anzahl der schliessenden Klammern ist. Andere Symbole sollen ignoriert werden.

Beispiel:

```
"if (i==1) {"    ACCEPT
"((*ptr).count())"  ACCEPT
"if ((i == j)" FAIL
"if (i == j))" FAIL
")(" FAIL
```

Diese Funktionalität wird aktiviert über den Konstruktor der PDA Klasse (gegeben in PDA.hpp). In der Main Methode wird angenommen dass die Klammersprache (und nicht die RNA-Hairpins) erkannt werden soll, wenn das erste Argument mindestens eine runde Klammer (offen ODER geschlossen) enthält.
Hinweis: Übergeben Sie an den Automaten (PDA::next) nur die Klammern – alle anderen Symbole sollten vorher in main() herausgefiltert werden, weil sonst die Anzahl der Regeln sehr gross werden könnte.