

NFA zu Regulärem Ausdruck

Hinweis zum Template

Die Klasse `Regex` ist eine andere als bei den früheren Aufgaben: Konkatenation und Alternative sind jetzt *n-äre* Operationen, nicht mehr *binäre*. Dies macht die Repräsentation deutlich kompakter.

Außerdem werden reguläre Ausdrücke nicht mehr über den Konstruktor erzeugt (`new Concat(new Single('a'), new Empty())`), sondern über *smart constructors*, z.B. `Regex.empty()`, `Regex.single('a')`, `Regex.concats(r1, r2, r3)` oder auch `r1.alternative(r2, r3)`. Diese *smart constructors* vereinfachen den erzeugten Ausdruck automatisch, sodass z.B. `abε|∅` direkt zu `ab` wird.

Verwenden Sie *nicht* die Konstruktoren (`new Concat(...)` etc.) um reguläre Ausdrücke zu erzeugen! Verwenden Sie *nur* die smart constructors! Durch die smart constructors werden die Ergebnisse sehr viel lesbarer. Dies gilt insb. auch für Haskell und Python.

Aufgabenstellung

In dieser Aufgabe sollen Sie die Konstruktion aus Satz 3.19, die auch in Übungsaufgabe 3.5 noch einmal behandelt wurde, implementieren. Der Kern dieser Prozedur ist eine Tabelle regulärer Ausdrücke α_{ij} , die am Ende der Prozedur für jedes Paar von Zuständen i und j beschreibt, mit welchen Wörtern man im Automaten von i nach j kommt. Hierfür wird die Tabelle zuerst initialisiert und dann in n Iterationen ($n = |Q|$) geupdated. Den Inhalt der Tabelle im k -ten Schritt bezeichnen wir wie in der Vorlesung mit α_{ij}^k , wobei α_{ij}^0 der Anfangszustand nach der Initialisierung ist.

Sie sollen in der Klasse `NFAToRegex` drei Methoden implementieren, die die folgenden Aufgaben übernehmen:

1. Initialisierung (d.h. Berechnung der Starttabelle α_{ij}^0)
2. Schritt von k zu $k + 1$, d.h. Berechnung der Tabelle α_{ij}^{k+1} aus der Tabelle α_{ij}^k und einem gegebenen Zustand k
3. Ablesen des Ergebnisses (also eines regulären Ausdrucks, der genau die Sprache des Automaten erkennt) aus der Endtabelle α_{ij}^n

Um das Einlesen, den Aufruf der von Ihnen implementierten Methoden und die Ausgabe der Ergebnisse kümmert sich das Template.

Um Ihnen eine eventuelle Fehlersuche zu erleichtern, wird in jedem Schritt die *gesamte* Tabelle α_{ij}^k ausgegeben und auch vom Testsystem überprüft. Dadurch können Sie sofort sehen, in welchem Schritt Sie einen Fehler gemacht haben. Die Reihenfolge, in der die Zustände durchlaufen werden ist beliebig und kann das Ergebnis beeinflussen – das Testsystem ist jedoch darauf ausgelegt und prüft nur, ob die Schritte in der von Ihnen (bzw. dem Template) gewählten Reihenfolge konsistent sind. Die regulären Ausdrücke in der Tabelle werden dabei auf Äquivalenz geprüft, nicht auf String-Gleichheit.

Eingabe

Eine Zeile, die den Modus angibt (entweder `Full` oder `Simple`).
Dann ein NFA im Format, das Sie aus dem Beispiel ablesen können.

Ausgabe

Da es bei einigen von Ihnen auf dem letzten Blatt Probleme mit Unicode-Zeichen gab, wurde die Syntax für reguläre Ausdrücke so angepasst, dass keine Unicode-Zeichen mehr verwendet werden. Die regulären Ausdrücke \emptyset bzw. ϵ werden als `{}` bzw. `()` ausgegeben.

Im `Simple`-Modus: Ein einzelner regulärer Ausdruck, der äquivalent zu dem gegebenen NFA ist.

Im `Full`-Modus: Ein Protokoll der Konstruktion aus Satz 3.19 mit allen Zwischenwerten der Tabelle α_{ij}^k .

Beispiele

Sample Input 1

```
Full
NFA
Alphabet: a;b
States: 1;2
Init: 1
Final: 2
Transitions:
1;a;2
END
```

Sample Output 1

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1):
(2, 2):
(1, 1):
(1, 2): a

Step 1
Processing state 1
(2, 1):
(2, 2):
(1, 1):
(1, 2): a

Step 2
Processing state 2
(2, 1):
(2, 2):
(1, 1):
(1, 2): a

Final result: a
END NFA TO REGEX TRACE
```

Sample Input 2

```
Full
NFA
Alphabet: a;b
States: 1;2
Init: 1
Final: 2
Transitions:
1;a;2
2;b;2
END
```

Sample Output 2

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1):
(2, 2): b|
(1, 1):
(1, 2): a

Step 1
Processing state 1
(2, 1):
(2, 2): b|
(1, 1):
(1, 2): a

Step 2
Processing state 2
(2, 1):
(2, 2): b| (b|) (b|)* (b|) |
(1, 1):
(1, 2): a|a(b|)* (b|)

Final result: a|a(b|)* (b|)
END NFA TO REGEX TRACE
```

Sample Input 3

```
Full
NFA
Alphabet: a;b
States: 1;2;3
Init: 1
Final: 3
Transitions:
1;a;2
2;b;3
END
```

Sample Output 3

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): b
(1, 2): a
(1, 3):
(3, 1):

Step 1
Processing state 1
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): b
(1, 2): a
(1, 3):
(3, 1):

Step 2
Processing state 2
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): b
(1, 2): a
(1, 3): ab
(3, 1):

Step 3
Processing state 3
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): b
(1, 2): a
(1, 3): ab
(3, 1):

Final result: ab
END NFA TO REGEX TRACE
```

Sample Input 4

```
Full
NFA
Alphabet: a;b
States: 1;2;3
Init: 1
Final: 1;3
Transitions:
1;a;2
2;b;3
3;a;2
END
```

Sample Output 4

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1):
(3, 2): a
(2, 2):
(3, 3):
(1, 1):
(2, 3): b
(1, 2): a
(1, 3):
(3, 1):

Step 1
Processing state 1
(2, 1):
(3, 2): a
(2, 2):
(3, 3):
(1, 1):
(2, 3): b
(1, 2): a
(1, 3):
(3, 1):

Step 2
Processing state 2
(2, 1):
(3, 2): a
(2, 2):
(3, 3): ab|
(1, 1):
(2, 3): b
(1, 2): a
(1, 3): ab
(3, 1):

Step 3
Processing state 3
(2, 1):
(3, 2): a|(ab|)(ab|)*a
(2, 2): b(ab|)*a|
(3, 3): ab|(ab|)(ab|)*(ab|)|
(1, 1):
(2, 3): b|b(ab|)*(ab|)
(1, 2): a|ab(ab|)*a
(1, 3): ab|ab(ab|)*(ab|)
(3, 1):

Final result: ab|ab(ab|)*(ab|)|
END NFA TO REGEX TRACE
```

Sample Input 5

```
Full
NFA
Alphabet: a;b
States: 1;2;3
Init: 1
Final: 1;3
Transitions:
1;a;2
1;b;3
2;a;3
2;b;3
END
```

Sample Output 5

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): a|b
(1, 2): a
(1, 3): b
(3, 1):

Step 1
Processing state 1
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): a|b
(1, 2): a
(1, 3): b
(3, 1):

Step 2
Processing state 2
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): a|b
(1, 2): a
(1, 3): b|a(a|b)
(3, 1):

Step 3
Processing state 3
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): a|b
(1, 2): a
(1, 3): b|a(a|b)
(3, 1):

Final result: b|a(a|b)|
END NFA TO REGEX TRACE
```

Sample Input 6

```
Full
NFA
Alphabet: a;b
States: 1;2
Init: 1
Final: 1
Transitions:
1;a;2
2;a;1
END
```

Sample Output 6

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1): a
(2, 2):
(1, 1):
(1, 2): a

Step 1
Processing state 1
(2, 1): a
(2, 2): aa|
(1, 1):
(1, 2): a

Step 2
Processing state 2
(2, 1): a|(aa|)(aa|)*a
(2, 2): aa|(aa|)(aa|)*(aa|)|
(1, 1): a(aa|)*a|
(1, 2): a|a(aa|)*(aa|)

Final result: a(aa|)*a|
END NFA TO REGEX TRACE
```

Sample Input 7

```
Full
NFA
Alphabet: a;b
States: 1;2;3
Init: 1
Final: 2
Transitions:
1;a;2
2;a;3
3;a;1
END
```

Sample Output 7

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1):
(3, 2):
(2, 2):
(3, 3):
(1, 1):
(2, 3): a
(1, 2): a
(1, 3):
(3, 1): a

Step 1
Processing state 1
(2, 1):
(3, 2): aa
(2, 2):
(3, 3):
(1, 1):
(2, 3): a
(1, 2): a
(1, 3):
(3, 1): a

Step 2
Processing state 2
(2, 1):
(3, 2): aa
(2, 2):
(3, 3): aaa|
(1, 1):
(2, 3): a
(1, 2): a
(1, 3): aa
(3, 1): a

Step 3
Processing state 3
(2, 1): a(aaa|)*a
(3, 2): aa|(aaa|)(aaa|)*aa
(2, 2): a(aaa|)*aa|
(3, 3): (aaa|)(aaa|)*(aaa|)|aaa|
(1, 1): aa(aaa|)*a|
(2, 3): a|a(aaa|)*(aaa|)
(1, 2): a|aa(aaa|)*aa
(1, 3): aa|aa(aaa|)*(aaa|)
(3, 1): a|(aaa|)(aaa|)*a

Final result: a|aa(aaa|)*aa
END NFA TO REGEX TRACE
```

Sample Input 8

```
Full
NFA
Alphabet: a;b;c
States: 1;2
Init: 1
Final: 2
Transitions:
1;a;1
1;b;2
2;c;2
END
```

Sample Output 8

```
BEGIN NFA TO REGEX TRACE
Step 0
(2, 1):
(2, 2): c|
(1, 1): a|
(1, 2): b|

Step 1
Processing state 1
(2, 1):
(2, 2): c|
(1, 1): a|(a|)(a|)*(a|)|
(1, 2): b|(a|)(a|)*b|

Step 2
Processing state 2
(2, 1):
(2, 2): c|(c|)(c|)*(c|)|
(1, 1): a|(a|)(a|)*(a|)|
(1, 2): b|(b|(a|)(a|)*b)(c|)*(c|)|(a|)(a|)*b|

Final result: b|(b|(a|)(a|)*b)(c|)*(c|)|(a|)(a|)*b|
END NFA TO REGEX TRACE
```