Einführung in die Computerlinguistik – Pumpinglemma für reguläre Sprachen – Suche mit regulären Ausdrücken

Dozentin: Wiebke Petersen

17.5.2010

Pumping-lemma für reguläre Sprachen

Lemma (Pumping-Lemma)

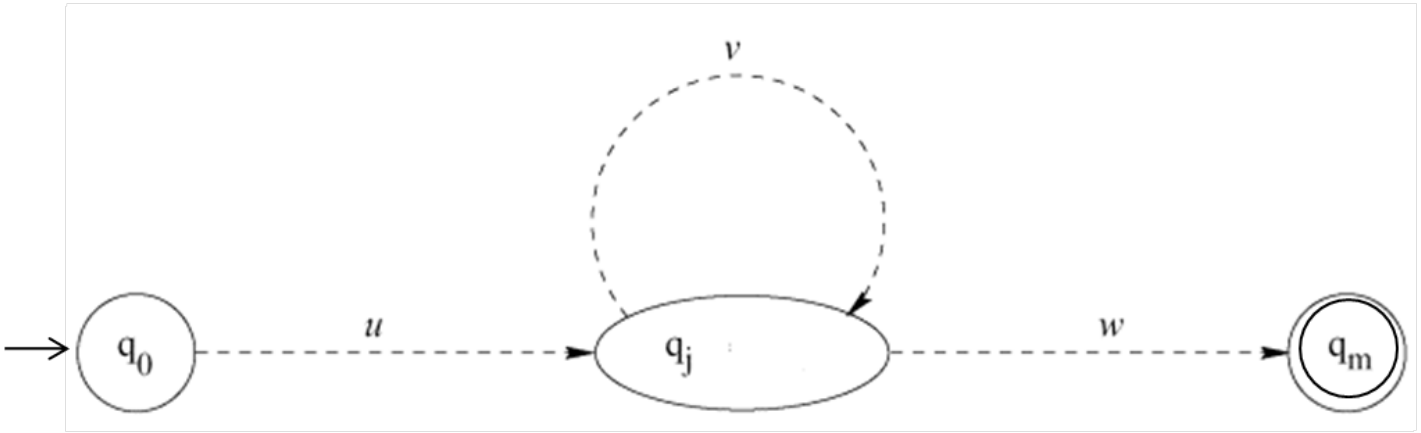
Sei L eine unendliche reguläre Sprache, dann gilt für jedes genügend lange Wort $z \in L$, daß es so in Teilworte $z = uvw$ $(u, w \in \Sigma^*, v \in \Sigma^+)$ zerlegt werden kann, dass jedes der Worte $uv^i w \in L$ $(i \geq 0)$ ein Wort der Sprache L ist.

Pumping-lemma für reguläre Sprachen

Lemma (Pumping-Lemma)

Sei L eine unendliche reguläre Sprache, dann gilt für jedes genügend lange Wort $z \in L$, daß es so in Teilworte $z = uvw$ $(u, w \in \Sigma^*, v \in \Sigma^+)$ zerlegt werden kann, dass jedes der Worte $uv^i w \in L$ $(i \geq 0)$ ein Wort der Sprache L ist.

Beweisidee:



Wenn ein Wort länger ist, als der Automat Zustände hat, dann muß bei der Verarbeitung des Wortes ein Zustand zweimal besucht werden. Es gibt somit eine Schleife, die beliebig oft durchlaufen werden kann; das Wort kann in dem Schleifenbereich aufgepumpt werden.

$L = \{a^n b^n : n \geq 0\}$ ist nicht regulär

- $L = \{a^n b^n : n \geq 0\}$:
  L ist unendlich. Wäre L regulär, dann müßte es für genügend lange Worte die geforderte pumpbare Zerlegung geben: aber

$L = \{a^n b^n : n \geq 0\}$ ist nicht regulär

- $L = \{a^n b^n : n \geq 0\}$:
  L ist unendlich. Wäre L regulär, dann müßte es für genügend lange Worte die geforderte pumpbare Zerlegung geben: aber
  1. das pumpbare Teilwort kann nicht nur aus a's bestehen, sonst würden beim Pumpen zuviele a's entstehen ($aa(aa)^2bbbb = aaaaaabbbb$).

$L = \{a^n b^n : n \geq 0\}$ ist nicht regulär

- $L = \{a^n b^n : n \geq 0\}$:
  L ist unendlich. Wäre L regulär, dann müßte es für genügend lange Worte die geforderte pumpbare Zerlegung geben: aber
  1. das pumpbare Teilwort kann nicht nur aus a's bestehen, sonst würden beim Pumpen zuviele a's entstehen ($aa(aa)^2bbbb = aaaaaabbbb$).
  2. das pumpbare Teilwort kann nicht nur aus b's bestehen, sonst würden beim Pumpen zuviele b's entstehen. ($aaaab(bb)^2b = aaaabbbbbb$).

$L = \{a^n b^n : n \geq 0\}$ ist nicht regulär

- $L = \{a^n b^n : n \geq 0\}$:
  L ist unendlich. Wäre L regulär, dann müßte es für genügend lange Worte die geforderte pumpbare Zerlegung geben: aber
  1. das pumpbare Teilwort kann nicht nur aus a's bestehen, sonst würden beim Pumpen zuviele a's entstehen ($aa(aa)^2bbbb = aaaaaabbbb$).
  2. das pumpbare Teilwort kann nicht nur aus b's bestehen, sonst würden beim Pumpen zuviele b's entstehen. ($aaaab(bb)^2b = aaaabbbbbb$).
  3. das pumpbare Teilwort kann nicht aus a's und b's bestehen, da beim Pumpen die Sortierung der a's und b's verloren ginge. ($aaa(ab)^2bbb = aaaababbbb$).

Suche mit regulären Ausdrücken

Suche nach allen Wortformen des Wortes “Student”

Suche mit regulären Ausdrücken

Suche nach allen Wortformen des Wortes “Student”

- Student
- Studenten
- Studentin
- Studentinnen

Suche mit regulären Ausdrücken

Suche nach allen Wortformen des Wortes “Student”

- Student
- Studenten
- Studentin
- Studentinnen

4 Suchstrings oder 1 regulärer Ausdruck:

Suche mit regulären Ausdrücken

Suche nach allen Wortformen des Wortes "Student"

- Student
- Studenten
- Studentin
- Studentinnen

4 Suchstrings oder 1 regulärer Ausdruck:

$\sqcup$Student(ϵ+en+in+innen)$\sqcup$

$\sqcup$Student(ϵ+en+in(ϵ+nen))$\sqcup$

Wichtige Begriffe zur Evaluation von Suchanfragen

- true positive (tp)
- true negative (tn)
- false positive (fp): "Studentenwohnheim"
- false negative (fn): "Er traf einen Studenten."
- precision
- recall
- accuracy

Wichtige Begriffe zur Evaluation von Suchanfragen

- true positive (tp)
- true negative (tn)
- false positive (fp): "Studentenwohnheim"
- false negative (fn): "Er traf einen Studenten."
- precision $= \frac{tp}{tp+fp}$
- recall
- accuracy

Wichtige Begriffe zur Evaluation von Suchanfragen

- true positive (tp)
- true negative (tn)
- false positive (fp): "Studentenwohnheim"
- false negative (fn): "Er traf einen Studenten."
- precision $= \frac{tp}{tp+fp}$
- recall $= \frac{tp}{tp+fn}$
- accuracy

Wichtige Begriffe zur Evaluation von Suchanfragen

- true positive (tp)
- true negative (tn)
- false positive (fp): "Studentenwohnheim"
- false negative (fn): "Er traf einen Studenten."
- precision $= \frac{tp}{tp+fp}$
- recall $= \frac{tp}{tp+fn}$
- accuracy $= \frac{tp+tn}{tp+fp+tn+fn}$

Regex: Reguläre Ausdrücke in Perl-Notation (1)

- String (Sequenz von Symbolen):
  - /Student/ $\neq$ /student/
  - /1234/

Regex: Reguläre Ausdrücke in Perl-Notation (1)

- String (Sequenz von Symbolen):
  - /Student/ $\neq$ /student/
  - /1234/
- Disjunktion:
  - /(Student|student)/

Regex: Reguläre Ausdrücke in Perl-Notation (1)

- String (Sequenz von Symbolen):
  - /Student/ $\neq$ /student/
  - /1234/
- Disjunktion:
  - /(Student|student)/
- Zeichenklasse:
  - /[aA]/ = /(a|A)/
  - /[1-4]/= /[1234]/ = /(1|2|3|4)/
  - /[a-z]/: Kleinbuchstabe
  - /[A-D]/= /[ABCD]/

Regex: Reguläre Ausdrücke in Perl-Notation (1)

- String (Sequenz von Symbolen):
  - /Student/ $\neq$ /student/
  - /1234/
- Disjunktion:
  - /(Student|student)/
- Zeichenklasse:
  - /[aA]/ = /(a|A)/
  - /[1-4]/= /[1234]/ = /(1|2|3|4)/
  - /[a-z]/: Kleinbuchstabe
  - /[A-D]/= /[ABCD]/
- Negation:
  - /[^a-z]/ kein Kleinbuchstabe
  - /[^a]/ nicht 'a'
  - /[^a^b]/ = /[^ab]/ nicht 'a' und nicht 'b'

Regex: Reguläre Ausdrücke in Perl-Notation (2)

- Zähler:
  - `?`: Eine oder keine Instanz des Vorangegangenen (/*colou*?*r*/)
  - `*`: beliebig viele Instanzen d.V.
  - `+`: beliebig viele, aber mindestens eine Instanz d.V.
  - `{n}`: genau *n* Instanzen d.V.
  - `{n,}`: mindestens *n* Instanzen d.V.
  - `{n,m}`: mindestens *n* und höchstens *m* Instanzen d.V.

Regex: Reguläre Ausdrücke in Perl-Notation (2)

- Zähler:
  - `?`: Eine oder keine Instanz des Vorangegangenen (/*colou*?*r*/)
  - `*`: beliebig viele Instanzen d.V.
  - `+`: beliebig viele, aber mindestens eine Instanz d.V.
  - `{n}`: genau *n* Instanzen d.V.
  - `{n,}`: mindestens *n* Instanzen d.V.
  - `{n,m}`: mindestens *n* und höchstens *m* Instanzen d.V.
- Anker:
  - `/^/`: Zeilenbeginn
  - `/$/`: Zeilenende
  - `/\b/`: Wortgrenze

Regex: Reguläre Ausdrücke in Perl-Notation (2)

- Zähler:
  - `?`: Eine oder keine Instanz des Vorangegangenen (/*colou*?*r*/)
  - `*`: beliebig viele Instanzen d.V.
  - `+`: beliebig viele, aber mindestens eine Instanz d.V.
  - `{n}`: genau *n* Instanzen d.V.
  - `{n,}`: mindestens *n* Instanzen d.V.
  - `{n,m}`: mindestens *n* und höchstens *m* Instanzen d.V.
- Anker:
  - `/^/`: Zeilenbeginn
  - `/$/`: Zeilenende
  - `/\b/`: Wortgrenze
- sonstige:
  - `/./`: beliebiges Zeichen (wildcard)
  - `/\n/`: Zeilenumbruch
  - `/\s/`: whitespace (Leerzeichen, Tabulator, ...)
  - `/ /`: Leerzeichen
  - `/\t/`: Tabulator
  - geschützte Zeichen: `\.`,`\?`,`\$`,...

Präzendenzhierarchie

- Klammern: `( )`
- Zähler: `?, *, +, {n}`
- Strings und Anker: `tor, ^, $, \b`
- Disjunktion: `|`

Greedy

Perl-Regex Quantoren sind "greedy", das heißt, sie matchen immer den längstmöglichen String. (Beispiel: `[a-z]*` Computerlinguistik)

Disjunktion nicht kommutativ: eagerness

Disjunktionen in Perl-Regex werden von links nach rechts abgearbeitet (eagerness= der erste Match von links nacht rechts gewinnt). (Beispiel: `m(o?|a)` $\neq$ `m(a|o?)`)

Anfangsbeispiel

$\sqcup$Student(ϵ+en+in(ϵ+nen))$\sqcup$

Anfangsbeispiel

␣Student(ϵ+en+in(ϵ+nen))␣

```
/\s(Student((en)?|in(nen)?))\s/
```

Anfangsbeispiel

␣Student(ϵ+en+in(ϵ+nen))␣

`/\s(Student((en)?|in(nen)?))\s/`

Problem: Zeilenanfang, Zeilenende, Satzende

Anfangsbeispiel

␣Student(ϵ+en+in(ϵ+nen))␣

```
/\s(Student((en)?|in(nen)?))\s/
```

Problem: Zeilenanfang, Zeilenende, Satzende

```
/( |^)Student(en|in(nen)?)?(\.| |$)/
```

Anfangsbeispiel

␣Student(ϵ+en+in(ϵ+nen))␣

```
/\s(Student((en)?|in(nen)?))\s/
```

Problem: Zeilenanfang, Zeilenende, Satzende

```
/( |^)Student(en|in(nen)?)?(\.| |$)/
```

Problem: Aufeinanderfolgen zweier Treffer “Studentin Student”

Anfangsbeispiel

␣Student(ϵ+en+in(ϵ+nen))␣

```
/\s(Student((en)?|in(nen)?))\s/
```

Problem: Zeilenanfang, Zeilenende, Satzende

```
/( |^)Student(en|in(nen)?)?(\.| |$)/
```

Problem: Aufeinanderfolgen zweier Treffer “Studentin Student”

```
/\bStudent(en|in(nen)?)?\b/
```

Gruppenübung zur Suche mit regulären Ausdrücken

Bilden Sie Gruppen von 3-5 Personen. Pro Gruppe bilden Sie zwei "Teilgruppen". Aufgabe der Teilgruppen:

- 1. Teilgruppe erstellt einen Perl-Regex, der auf die zu suchenden Strings matcht (manchmal ist es nicht vermeidbar, dass Teile des Kontextes der zu suchenden Strings mitmatchen)
- 2. Teilgruppe überlegt sich Strings, die von dem Perl-Regex nicht erfaßt oder fälschlicherweise erfaßt werden.

Nach jeder Teilaufgabe wechseln Sie die Rollen (Aufgaben auf der kommenden Folie). Beachten Sie, dass sich die meisten der Aufgaben mit den behandelten Mitteln nicht vollständig lösen lassen. Suchen Sie eine optimale Lösung.

Überlegen Sie sich, warum sich jeder Perl-Regex ein einen regulären Ausdruck nach Kleene überführen läßt.

Gruppenübung zur Suche mit regulären Ausdrücken

Erstellen Sie Perl-Regexe für die folgenden Aufgaben:

1. Suche aller Vorkommnisse der Wortform "alt"
2. Suche aller Vorkommnisse von Wortformen des Wortes "alt"
3. Suche aller Nomina in einem deutschen Text
4. Suche aller Abkürzungen in einem deutschen Text
5. Suche wohlgeformter Email-Adressen (`http://de.wikipedia.org/wiki/E-Mail-Adresse`)
6. Überpfüfung, ob ein Password die folgende Bedingung erfüllt: enthält mindestens 2 Buchstaben, wovon mindestens einer ein Kleinbuchstabe ist.[a]

Testen Sie Ihre Perl-Regexe mit einem der folgenden Tools:
`http://Regexpal.com/`
`http://www.myRegextester.com/`
`http://regex.larsolavtorvik.com/`

[a] Geänderte Aufgabe, da die Ursprungsaufgabe nur mit einem extrem langen Regex gelöst werden konnte (Danke für den Hinweis!).

Hausaufgabe (Abgabe: 7.6.2010) (für BN: 3 und entweder 1.1 oder 1.2)

1. Erstellen Sie einen Perl-Regex, der folgendes findet:
   1. Superlative in einem deutschen Text
   2. Abkürzungen in einem deutschen Text

   Beschreiben Sie genau, in welchen Fällen Ihr Perl-Regex zu falsch positiven oder falsch negativen Treffern führt.
2. Lösen Sie Aufgabe 6 von der vorangegangenen Folie
3. Geben Sie zu den folgenden Perl-Regexen formale reguläre Ausdrücke nach der Definition auf Folie 8 aus `EinfCl_3.pdf` an:
   1. `/(ab){2}/`
   2. `/[aA]+/`
   3. `/[Ee]in(e(n|r|m|s)?)?/`
4. Wiederholung: Geben Sie einen endlichen Automaten an, der die durch den Regex `/[aA]+/` beschriebene Sprache akzeptiert. Nennen Sie ein Wort, das akzeptiert wird und begründen Sie, warum es von dem Automaten akzeptiert wird. Geben Sie die Übergangstabelle für den Automaten an.

Nützliches Material

- **Übungstutorium:**
  `http://www.regenechsen.de/phpwcms/index.php?Regex_allg`
- **Überblick:**
  `http://Regexlib.com/CheatSheet.aspx`
- **Literatur:**
  Jeffrey E. F. Friedl: *Reguläre Ausdrücke*. O'Reilly, 2. Auflage 2007.