

Programmering för språkteknologer II

OH-serie:

- ändliga automater
- reguljära uttryck i Java



UPPSALA
UNIVERSITET

Mats Dahllöf

Institutionen för lingvistik och filologi
September 2009

1

Ändliga automater

- ”Abstrakt maskin”, ”tillståndsmaskin”, ”transitionssystem”. (Den enklaste typ man brukar ta upp.)
- Automaten befinner sig i varje givet läge i ett visst **tillstånd**. Det finns ett **ändligt antal** tillstånd (men olika för olika automater).
- Automaten läser symboler och byter tillstånd utifrån symbol och aktuellt tillstånd (**övergång/transition**).

2

Deterministiska ändliga automater

- Om det nya tillståndet är entydigt bestämt, alltså att en viss symbol i ett visst tillstånd leder till maximalt ETT bestämt tillstånd, så sägs automaten vara **deterministisk**. (Det blir enklare, och är den variant vi primärt skall arbeta med.)
- En automat antingen godkänner (accepterar) eller underkänner en sekvens av symboler (sträng).
- Dessa automater är alltså effektiva för att känna igen vissa typer av teckensekvenser. (Linjär tidsåtgång.)

3

Deterministiska ändliga automater

En automat antingen godkänner (accepterar) eller underkänner en sekvens av symboler (sträng):

- Automaten befinner sig först i **initialtillståndet**.
- Automaten läser en symbol i taget och följer övergångarna Tillstånd och symbol bestämmer nästa tillstånd.
- Automaten har godkänt en sträng om den står i ett tillstånd som räknas som **finaltillstånd** då den läst fram till slutet av strängen.

4

Ändliga automater (deterministiska)

- **Övergångarna** i automaten säger alltså att givet ett tillstånd och en symbol så går den till ett nytt tillstånd (eller samma som tidigare).
- Om det saknas en övergång för ett tillstånd och en symbol så underkänner automaten aktuell sträng.
- En automat kan komma tillbaka till ett tidigare tillstånd, eller stå kvar i samma tillstånd vid en övergång.
- Ändliga automater är effektiva och användbara mekanismer för stränganalys.

5

Exempeltillämpning

- Sannolika datum (i vissa texter): 090131, 090228, 090917, 090930, 091231.
- Felaktiga som datum: 090100, 090132, 090230, 090431, 091301.
- Kolla datum kan vara intressant i t.ex. talsyntes, sökning, extraktion och elektroniska formulär.
- Problemet kan lösas med en ändlig automat (vilket vi gör som del av labben).

6

Gör automater för...

- Alla strängar av (enbart) a och b som innehåller minst tre b .
- Alla strängar av (enbart) a och b som inte innehåller tre a i rad.
- Alla strängar av (enbart) a och b som innehåller ett jämt antal a och ett udda antal b .
- Alla strängar av (enbart) a och b som om de innehåller minst två a i rad också innehåller minst två b i rad.

7

Varför inte dessa?

Varför kan vi inte göra ändliga automater för dessa?

- Alla strängar av (enbart) a och b som innehåller lika många a som b .
- Alla strängar av (enbart) a och b som är palindrom (likadana lästa framifrån som bakifrån).

8

Skriva ned en automat (antaganden för labb.)

- Symbolerna i strängarna (automatens alfabet) – vanliga tecken (utom blanktecken och asterisk)
- Tillstånd – vilka ord som helst (t.ex. s1, s2, etc.)
- Sjelva definitionen av automaten:
 - ett **initialtillstånd**
 - godtyckligt antal **finaltillstånd**
 - godtyckligt antal **övergångar** (tillstånd – symbol – tillstånd)

9

Skriva ned en automat (syntax)

Automaten skrivs ned i en textfil med följande sorts rader. Symbolerna skiljs åt av blankslag.

- en rad för **initialtillstånd** (precis en)


```
initial tillstånd
t.ex. initial s1
```
- en rad för **finaltillstånd** (flera tänkbart, men samma rad)


```
final tillstånd
t.ex. final s3 s4 s5
```

10

Skriva ned en automat (syntax)

- en rad för varje **övergång** (normalt många)


```
tillstånd symbol tillstånd
t.ex. s1 a s2
t.ex. s2 b s5
```
- Andra rader avvisas.

11

Reguljära uttryck

- Sätt att karaktärisera teckensekvenser
- En teckensekvens machar eller matchar inte ett reguljärt uttryck
- Reguljära uttryck är nära besläktade med ändliga automater (de kan karaktärisera samma klasser av teckensekvenser)
- Mycket viktiga både i språkteknologin och datatekniken allmänt sett

12

Reguljära uttryck i Java

- Reguljära uttrycken är i sig strängar (som alltså tolkas på ett speciellt sätt)
- **Teckenklasser** är reguljära uttryck som avser enteckenssträngar.

Enklaste fall: Enstaka tecken matchar samma tecken:
Gäller bokstäver m.m. T.ex.: "a" matchar "a".
- (Sedan har vi reguljära uttryck som fångar längre strängar.)

13

Reguljära uttryck (teckenklasser) i Java

- Teckenklasser: T.ex.


```
"[abc]" matchar vilken som av "a", "b" och "c".
"[a-z]" matchar vilken som av "a", "b", o.s.v. t.o.m. "z".
"[a-zA-Z]" matchar vilken som av "a", "b", o.s.v. t.o.m. "z" och "A", "B", o.s.v. t.o.m. "Z".
"[a-zA-ZäöÄÖ]" matchar vilken bokstav som helst i svenska "standardalfabetet". (Men t.ex. "é" faller utanför.)
```

14

Reguljära uttryck (teckenklasser) i Java

- Negerade teckenklasser: T.ex.


```
"[^abc]" matchar alla (enstaka) tecken UTOM "a", "b" och "c".
"[^a-ms]" matchar alla (enstaka) tecken UTOM "a", "b", o.s.v. t.o.m. "m" och "s".
"[^a-zA-ZäöÄÖ]" matchar alla (enstaka) tecken som inte är svenska bokstäver.
```

15

Reguljära uttryck (teckenklasser) i Java

- Symboler för **fördefinierade klasser**:


```
"\d" matchar en siffra (synonym "[0-9]").
Obs! "\d" måste stå som "\\d" i koden. (Man måste backslasha backslashen för att få in den i strängen.)
"\D" matchar en icke-siffra (synonym "[^0-9]").
"\w" matchar ett (engelskt) ordtecken (synonym "[a-zA-Z_0-9]").
"\W" matchar ett icke-ordtecken (synonym "[^\w]").
```

16

Reguljära uttryck (teckenklasser) i Java

- Snitt (intersection) mellan teckenklasser:
`"[C0&&C1]"`
 C_0 och C_1 uttryck för teckenklasser.
 En sträng (enteckens) matchar `"[C0&&C1]"` om den matchar både C_0 och C_1 .
 T.ex.
`"[\d&&[~27]]"` matchar alla siffror utom "2" och "7".
`"[\w&&[~AEIOUY]]"` matchar alla bokstäver utom (vanliga engelska) vokaler som versal.

17

Reguljära uttryck (teckenklasser) i Java

- Många tecken faller utanför de fördefinierade klasserna.
 T.ex.
`áäääçéëëíííí`
`ñóòòöüüüÿÿæÇ`
`ÀÁÁÁÆÈÈÈÍÎÎÎÓÔÔÔÛÜÜÝŸ`
 Dessa måste tas med explicit i de reguljära uttrycken om man vill hantera dem.

18

Reguljära uttryck i Java: konkatenering

- Teckenklasser är, som sagt, reguljära uttryck som matchar ett tecken.
- Om R_0 och R_1 är reguljära uttryck, så matchar R_0R_1 en sträng som är sammansatt av två strängar där den första matchar R_0 och den andra R_1 .
- Exempel: `"[ab][cd]"` matchar dess fyra strängar:
`"ac"`, `"ad"`, `"bc"`, `"bd"`

19

Reguljära uttryck i Java: optionalitet

- Om R är ett reguljärt uttryck, så matchar `"R?"` tomma strängen och varje sträng som matchar R . (alltså R taget en eller ingen gång, s.a.s.)
- Exempel: `"a?"` matchar `""` och `"a"`.
- Exempel: `"a?b?c"` matchar `"c"`, `"ac"`, `"bc"` och `"abc"`. (Bara de fyra.)

20

Reguljära uttryck i Java: Kleeneslutning

- Om R är ett reguljärt uttryck, så matchar `"R*" (kallas "Kleene-stjärna")` en sträng som är sammansatt av noll eller fler (godtyckligt många) strängar som matchar R .
- Exempel: `"a*" matchar "", "a", "aa", "aaa", "aaaa", o.s.v. (oändligt många).`
- Exempel: `"[ab]*" matchar "", "a", "b", "aa", "ab", "ba", "bb", o.s.v.`

21

Reguljära uttryck i Java: plusoperatoren

- Om R är ett reguljärt uttryck, så matchar `"R+" en sträng som är sammansatt av en eller fler (godtyckligt många) strängar som matchar R . (Synonym: "RR*")`
- Exempel: `"a+" matchar "a", "aa", "aaa", "aaaa", o.s.v. (oändligt många).`
- Exempel: `"(ab)+" matchar "ab", "abab", "ababab", o.s.v. (oändligt många).`

22

Reguljära uttryck i Java: union

- Om R_0 och R_1 är reguljära uttryck, så matchar $R_0 | R_1$ en sträng som matchar R_0 ELLER R_1 .
- Exempel: `"((abc) | (cd)) . *e . *" matchar alla strängar som börjar med "abc" eller "cd" och dessutom innehåller ett "e"`
- Exempel: `"((abc) | (cd)) . *e . *" matchar alla strängar som börjar med "abc" eller "cd" och dessutom innehåller ett "e"`

23

Reguljär matchning är indeterministisk

- T.ex. `"\w*a\w*" (i koden "\w*a\w*") kan matcha "apanage" på tre sätt. (Vilka?)`
- T.ex. `"a\w*a\w*" (i koden "a\w*a\w*") kan matcha "apanage" på två sätt. (Vilka?)`
- Och (fråga): På hur många sätt matchar `"\w+a\w*" strängen "apanage"?`

24

Reguljära kvantifikatorer, tre typer

Greedy	Reluctant	Possessive
X?	X??	X?+
X*	X*?	X**
X+	X+?	X++

Dessa representerar tre olika matchningsprinciper.

Obs! Frågetecken och plustecken får plötsligt helt annan innebörd än tidigare, alltså "reluctant" och "possessive".

25

Tre typer av matchnings sätt

(Det här är lite klurigt!)

- **Greedy:** Matchar så lång delsträng som möjligt först; försöker med kortare därefter. (Default.)
- **Reluctant:** Matchar så kort delsträng som möjligt först; försöker med längre därefter.
- **Possessive:** Matchar som greedy, men eliminerar alternativa tidigare matchningsalternativ. Den lägger så att säga beslag på den delsträng som den matchat.

26

Greedy matchning, exempel

"\w*a\w*" ("\w*a\w*") matchar "apanage":

- (1) a p a n a g e
 \w* a \w*
- (2) a p a n a g e
 \w* a \w*
- (3) a p a n a g e
 \w*a \w*

27

Reluctant matchning, exempel

"\w*?a\w*" ("\w*?a\w*") matchar "apanage":

- (1) a p a n a g e
 \w*?a \w*
- (2) a p a n a g e
 \w*? a \w*
- (3) a p a n a g e
 \w*? a \w*

28

Possessive matchning, exempel

"\w**a\w**" (i koden "\w**a\w**") kan matchar INTE ALLS "apanage".

- (1) a p a n a g e
 \w** a \w**

 (lägger beslag på) (inget kvar)

29

Ersättningsmetoder i klassen String

Byter ut den först (i matchningsordning) matchande strängen mot en annan sträng.

- `replaceAll(String regex, String replacmnt)`
 returtyp String (avser alla förekomster i strängen)
- `replaceFirst(String regex, String replacmnt)`
 returtyp String (avser första förekomsten i strängen)

30

Exempel (Varför blir det så här?)

```

"aaa".replaceAll("a*", "b")
  returnerar "bb"
"aaa".replaceAll("a*?", "b")
  returnerar "bababab"
"aaa".replaceAll("a**", "b")
  returnerar "bb"
    
```

31

Exempel (Varför blir det så här?)

```

"aaa".replaceAll("a+", "b")
  returnerar "b"
"aaa".replaceAll("a+?", "b")
  returnerar "bbb"
"aaa".replaceAll("a++", "b")
  returnerar "b"
    
```

32

Exempel

```
"aabaa".replaceAll("\w*ba*", "c")
returnerar "c"
"aabaa".replaceAll("\w*?ba*", "c")
returnerar "c"
"aabaa".replaceAll("\w**ba*", "c")
returnerar "aabaa"
```

33

Exempel

```
"en <i>liten</i> hund".replaceAll("</?.*>", "")
returnerar "en hund"
"en <i>liten</i> hund".replaceAll("</?.*?>", "")
returnerar "en liten hund"
"en <i>liten</i> hund".replaceAll("</?.*+>", "")
returnerar "en <i>liten</i> hund"
```

34

Dela upp en sträng

- I klassen `String` finns även en metod `split(String regex)` med returtyp `String[]` som klipper upp en sträng vid varje matchande delsträng.
- `"en liten hund".split(" ")` ger ett fält med `"en", "liten", "hund"`.
- `"en <i>liten</i> hund".split(" *</?.*?> *")` ger samma resultat.

35