

# Regulære udtryk

DM573

Rolf Fagerberg

# Mål

Målet for disse slides er at beskrive **regulære udtryk**. Disse er endnu en måde at beskrive formelle sprog på. De bruges i praksis bl.a. til at lave søgninger i tekst.

Dette emne er et uddrag af kurset *DM553: Komplexitet og beregnelighed* (6. semester).

## Definition:

Et sprog = en mængde af strenge.

## Eksempler:

$\{ \text{agurk, gulerod, elephant} \}$

$\{ a, b, ab \}$

$\{ a, ab, abb, abbb, abbbb, \dots \}$

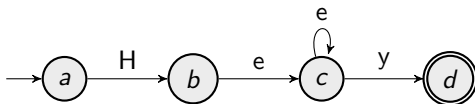
$\{ a, b, \varepsilon \}$

$\{ \}$

Her står  $\varepsilon$  for den tomme streng.

**Et sprog kan defineres via en DFA.** Sproget er mængden af strenge accepteret af DFA'en.

Eksempel fra Fabrizio's slides:

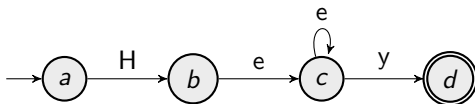


Hvad er sproget her?

# Sprog

**Et sprog kan defineres via en DFA.** Sproget er mængden af strenge accepteret af DFA'en.

Eksempel fra Fabrizio's slides:



Hvad er sproget her?

$\{ \text{Hey, Heey, Heeey, ...} \}$

# Sprog

**Et sprog kan defineres via en CFG.** Sproget er mængden af strenge, som kan udledes (can be derived) via CFG'en.

Eksempel fra Fabrizio's slides:

|     |               |                    |
|-----|---------------|--------------------|
| $S$ | $\rightarrow$ | Hello $\sqcup$ $T$ |
| $S$ | $\rightarrow$ | Hey $\sqcup$ $T$   |
| $T$ | $\rightarrow$ | there              |

Hvad er sproget her?

# Sprog

**Et sprog kan defineres via en CFG.** Sproget er mængden af strenge, som kan udledes (can be derived) via CFG'en.

Eksempel fra Fabrizio's slides:

$$\begin{array}{ll} S & \rightarrow \text{Hello\_}T \\ S & \rightarrow \text{Hey\_}T \\ T & \rightarrow \text{there} \end{array}$$

Hvad er sproget her?

$$\{ \text{Hello\_there}, \text{Hey\_there} \}$$

# Regulære udtryk

Regulære udtryk (regular expressions, regex'es) er **endnu en mekanisme til at definere sprog**.

De har vist sig meget anvendelige til at specificere søgninger i tekst (eksempler senere).

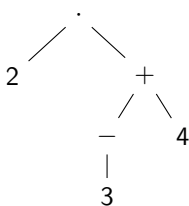
De er tilgængelige i de fleste programmeringssprog (Python, Java,...) samt på kommandolinjen.



# Udtrykstræer

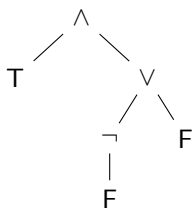
Ligesom normale algebraiske udtryk og boolske udtryk er regulære udtryk udtrykstræer, hvor beregning (nye værdier fra gamle) går nedefra og op.

Værdierne er denne gang *sprog*. Vi mangler at definere de mulige **startværdier** (i blade) samt **operatorerne** (i knuderne).



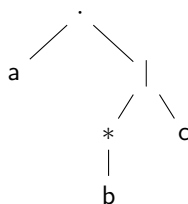
Input: tal  
Output: tal

$$2 \cdot ((-3) + 4)$$



Input: Booleans  
Output: Booleans

$$T \wedge ((\neg F) \vee F)$$



Input: Sprog  
Output: Sprog

$$a \cdot ((b^*) \mid c)$$

Parenteser bruges til en lineær beskrivelse af den hierarkiske struktur.

# Regulære udtryk

**Startværdierne** er **empty string** og **single char**.

**Operatorerne**, som bygger større regex'er og større tilhørende sprog ud fra mindre, er **concatenation**, **alternation** og **Kleene star**.

|                      | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|----------------------|-----------------|---------------------------------------------------------------------------------------------|
| <b>empty string</b>  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| <b>single char</b>   | $c$             | $\{ c \}$                                                                                   |
| <b>concatenation</b> | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| <b>alternation</b>   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| <b>Kleene star</b>   | $R^*$           | Alle strenge som kan fås via gentagen sammensætning af strenge i $L \cup \{ \varepsilon \}$ |

Her er  $R_1$ ,  $R_2$  og  $R$  allerede opbyggede regulære udtryk med tilhørende sprog  $L_1$ ,  $L_2$  og  $L$ . Strengen  $x \cdot y$  er tegnene i strengen  $x$  efterfulgt af tegnene i strengen  $y$ .

[Teknisk set er det tomme sprog  $\emptyset = \{ \}$  også en startværdi. At tilføje dette giver ikke nye muligheder for at lave sprog udover  $\{ \}$  selv, så vi udelader det i vores version her.]

# Færre parenteser

Vi kan spare parenteser ved at have en standard rækkefølge af operationerne (kaldet bindingsstyrke, prioritet eller precedence).

**For tal:** potens før gange før plus.

Så  $4^2 \cdot 3 + 5$  forstås som  $((4^2) \cdot 3) + 5$ .

**For regulære udtryk:** Kleene star før concatenation før alternation.

Så  $a^* \cdot b \mid c$  forstås som  $((a^*) \cdot b) \mid c$ .

# Færre parenteser

Vi kan spare parenteser ved at have en standard rækkefølge af operationerne (kaldet bindingsstyrke, prioritet eller precedence).

**For tal:** potens før gange før plus.

Så  $4^2 \cdot 3 + 5$  forstås som  $((4^2) \cdot 3) + 5$ .

**For regulære udtryk:** Kleene star før concatenation før alternation.

Så  $a^* \cdot b \mid c$  forstås som  $((a^*) \cdot b) \mid c$ .

Vi bliver stadig nødt til at bruge parenteser, når vi mener noget andet end denne standard rækkefølge. Eksempel med tal, hvor parenteser er nødvendige:  $4^2 \cdot (3 + 5)$

## Yderligere simplificering af notation

Vi kan også spare notation ved at udelade “.”.

**For tal (gange):**  $2 \cdot x$  skrives blot som  $2x$ .

**For regulære udtryk (concatenation):**  $a \cdot b$  skrives blot som  $ab$ .

# Yderligere simplificering af notation

Vi kan også spare notation ved at udelade “ $\cdot$ ”.

**For tal (gange):**  $2 \cdot x$  skrives blot som  $2x$ .

**For regulære udtryk (concatenation):**  $a \cdot b$  skrives blot som  $ab$ .

Vi kan også spare parenteser ved at bruge associativitet af operationer.

**For tal:**  $2 + (3 + 4) = (2 + 3) + 4$ , da begge er lig 9. Udtrykket skrives derfor også bare som  $2 + 3 + 4$ .

**For regulære udtryk:**  $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ , da begge har det tilhørende sprog  $\{ abc \}$ . Udtrykket skrives derfor også bare som  $abc$ .

# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

Eksempler:

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i> |
|------------------------|-------------------------|
| $abc xy$               | $\{ abc, xy \}$         |



# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

Eksempler:

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i> |
|------------------------|-------------------------|
| $abc xy$               | $\{ abc, xy \}$         |
| $(\varepsilon a)bc$    | $\{ bc, abc \}$         |

# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

Eksempler:

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>  |
|------------------------|--------------------------|
| $abc xy$               | $\{ abc, xy \}$          |
| $(\varepsilon a)bc$    | $\{ bc, abc \}$          |
| $\varepsilon abc$      | $\{ \varepsilon, abc \}$ |

# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

Eksempler:

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>                |
|------------------------|----------------------------------------|
| $abc xy$               | $\{ abc, xy \}$                        |
| $(\varepsilon a)bc$    | $\{ bc, abc \}$                        |
| $\varepsilon abc$      | $\{ \varepsilon, abc \}$               |
| $a^*$                  | $\{ \varepsilon, a, aa, aaa, \dots \}$ |

# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

Eksempler:

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>                |
|------------------------|----------------------------------------|
| $abc xy$               | $\{ abc, xy \}$                        |
| $(\varepsilon a)bc$    | $\{ bc, abc \}$                        |
| $\varepsilon abc$      | $\{ \varepsilon, abc \}$               |
| $a^*$                  | $\{ \varepsilon, a, aa, aaa, \dots \}$ |
| $ba^*c$                | $\{ bc, bac, baac, baaac, \dots \}$    |

# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

Eksempler:

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>                          |
|------------------------|--------------------------------------------------|
| $abc xy$               | $\{ abc, xy \}$                                  |
| $(\varepsilon a)bc$    | $\{ bc, abc \}$                                  |
| $\varepsilon abc$      | $\{ \varepsilon, abc \}$                         |
| $a^*$                  | $\{ \varepsilon, a, aa, aaa, \dots \}$           |
| $ba^*c$                | $\{ bc, bac, baac, baaac, \dots \}$              |
| $ac b^*$               | $\{ ac, \varepsilon, b, bb, bbb, bbbb, \dots \}$ |

# Eksempler på regulære udtryk

Recap af definition:

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------|
| empty string  | $\varepsilon$   | $\{ \varepsilon \}$                                                                         |
| single char   | $c$             | $\{ c \}$                                                                                   |
| concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                                 |
| alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                              |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \varepsilon \}$ |

Eksempler:

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>                                                 |
|------------------------|-------------------------------------------------------------------------|
| $abc xy$               | $\{ abc, xy \}$                                                         |
| $(\varepsilon a)bc$    | $\{ bc, abc \}$                                                         |
| $\varepsilon abc$      | $\{ \varepsilon, abc \}$                                                |
| $a^*$                  | $\{ \varepsilon, a, aa, aaa, \dots \}$                                  |
| $ba^*c$                | $\{ bc, bac, baac, baaac, \dots \}$                                     |
| $ac b^*$               | $\{ ac, \varepsilon, b, bb, bbb, bbbb, \dots \}$                        |
| $(ac b)^*$             | $\{ \varepsilon, ac, b, acb, bac, acacb, bacb, acacacbbbbacb, \dots \}$ |

# Flere eksempler

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i> |
|------------------------|-------------------------|
| grey gray              | { grey, gray }          |

# Flere eksempler

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i> |
|------------------------|-------------------------|
| grey gray              | { grey, gray }          |
| gr(e a)y               | { grey, gray }          |



## Flere eksempler

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>     |
|------------------------|-----------------------------|
| grey gray              | { grey, gray }              |
| gr(e a)y               | { grey, gray }              |
| 1(0 1)*                | alle binære heltal $\geq 1$ |

# Flere eksempler

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>                            |
|------------------------|----------------------------------------------------|
| <code>grey gray</code> | $\{ \text{grey, gray} \}$                          |
| <code>gr(e a)y</code>  | $\{ \text{grey, gray} \}$                          |
| <code>1(0 1)*</code>   | alle binære heltal $\geq 1$                        |
| <code>␣␣␣*</code>      | to eller flere spaces i træk (overflødigt i tekst) |

# Flere eksempler

| <i>Regulært udtryk</i> | <i>Tilhørende sprog</i>                            |
|------------------------|----------------------------------------------------|
| grey gray              | { grey, gray }                                     |
| gr(e a)y               | { grey, gray }                                     |
| 1(0 1)*                | alle binære heltal $\geq 1$                        |
| $\cup\cup\cup^*$       | to eller flere spaces i træk (overflødigt i tekst) |
| 199(0 1 2 3 4)         | første halvdel af 1990'erne                        |

# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)

# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)  
Svar:  $(A|C|G|T)^*$

# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)  
Svar:  $(A|C|G|T)^*$
- ▶ Alle aldre, hvor man er teenager

# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)  
Svar:  $(A|C|G|T)^*$
- ▶ Alle aldre, hvor man er teenager  
Svar:  $1(3|4|5|6|7|8|9)$

# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)  
Svar:  $(A|C|G|T)^*$
- ▶ Alle aldre, hvor man er teenager  
Svar:  $1(3|4|5|6|7|8|9)$
- ▶ Alle binære tal  $\geq 4$



# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)

Svar:  $(A|C|G|T)^*$

- ▶ Alle aldre, hvor man er teenager

Svar:  $1(3|4|5|6|7|8|9)$

- ▶ Alle binære tal  $\geq 4$

Svar:  $1(0|1)(0|1)(0|1)^*$

# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)  
Svar:  $(A|C|G|T)^*$
- ▶ Alle aldre, hvor man er teenager  
Svar:  $1(3|4|5|6|7|8|9)$
- ▶ Alle binære tal  $\geq 4$   
Svar:  $1(0|1)(0|1)(0|1)^*$
- ▶ Alle datoer i fjerde kvartal i år, i formatet 2025.11.03

# Opgaver

Find et regulært udtryk for:

- ▶ Alle DNA strenge (her er et eksempel: AACTCGA)  
Svar:  $(A|C|G|T)^*$
- ▶ Alle aldre, hvor man er teenager  
Svar:  $1(3|4|5|6|7|8|9)$
- ▶ Alle binære tal  $\geq 4$   
Svar:  $1(0|1)(0|1)(0|1)^*$
- ▶ Alle datoer i fjerde kvartal i år, i formatet 2025.11.03  
Svar:  $2025.1((0|1|2).((0|1|2)(1|2|3|4|5|6|7|8|9)|(1|2|3)0)|(0|2).31)$

Hvorfor hedder det “Kleene star”?

# Hvorfor hedder det “Kleene star”?

Stephen C. Kleene (1909–1994)

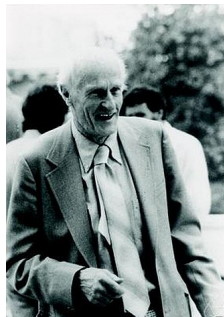


Foto: Konrad Jacobs

Matematiker med fokus på logik, beregnelighed, beregningsmodeller og deres styrker. Arbejdede fra 1930'erne. Definerede begrebet regulære udtryk i 1951.

## Søgning i tekst

**Søgning i tekst:** find alle steder, hvor en bestemt streng optræder.

**Eksempel:** find alle steder, hvor “dog” optræder i flg. tekst:

Giv mig dog en doggybag, så min bulldog  
kan få resten af min hotdog.

## Søgning i tekst

**Søgning i tekst:** find alle steder, hvor en bestemt streng optræder.

**Eksempel:** find alle steder, hvor “dog” optræder i flg. tekst:

Giv mig **dog** en **doggy**bag, så min bulld**dog**  
kan få resten af min hot**dog**.

# Søgning i tekst

**Søgning i tekst:** find alle steder, hvor en bestemt streng optræder.

**Eksempel:** find alle steder, hvor “dog” optræder i flg. tekst:

Giv mig dog en doggybag, så min bulldog  
kan få resten af min hotdog.

Når vi søger i tekst efter strenge, ønsker vi ofte at finde én ud af flere beslægtede strenge. Dvs. vores søgningen er specificeret ved en **mængde** af strenge (dvs. ved et sprog).



# Søgning i tekst

**Søgning i tekst:** find alle steder, hvor en bestemt streng optræder.

**Eksempel:** find alle steder, hvor “dog” optræder i flg. tekst:

Giv mig dog en doggybag, så min bulldog  
kan få resten af min hotdog.

Når vi søger i tekst efter strenge, ønsker vi ofte at finde én ud af flere beslægtede strenge. Dvs. vores søgningen er specificeret ved en **mængde** af strenge (dvs. ved et sprog).

To eksempler på sådanne søgninger:

- ▶ Alle steder, hvor strengen “gray” eller strengen “grey” optræder.
- ▶ Alle i steder i teksten, som angiver datoer i fjerde kvartal af 2024.

# Søgning i tekst

**Søgning i tekst:** find alle steder, hvor en bestemt streng optræder.

**Eksempel:** find alle steder, hvor “dog” optræder i flg. tekst:

Giv mig dog en doggybag, så min bulldog  
kan få resten af min hotdog.

Når vi søger i tekst efter strenge, ønsker vi ofte at finde én ud af flere beslægtede strenge. Dvs. vores søgningen er specificeret ved en **mængde** af strenge (dvs. ved et sprog).

To eksempler på sådanne søgninger:

- ▶ Alle steder, hvor strengen “gray” eller strengen “grey” optræder.
- ▶ Alle i steder i teksten, som angiver datoer i fjerde kvartal af 2024.

Illustration af den første af disse søgninger:

Stingrays like to drink earl grey on a gray day.

# Søgning i tekst

**Søgning i tekst:** find alle steder, hvor en bestemt streng optræder.

**Eksempel:** find alle steder, hvor “dog” optræder i flg. tekst:

Giv mig dog en doggybag, så min bulldog  
kan få resten af min hotdog.

Når vi søger i tekst efter strenge, ønsker vi ofte at finde én ud af flere beslægtede strenge. Dvs. vores søgningen er specificeret ved en **mængde** af strenge (dvs. ved et sprog).

To eksempler på sådanne søgninger:

- ▶ Alle steder, hvor strengen “gray” eller strengen “grey” optræder.
- ▶ Alle i steder i teksten, som angiver datoer i fjerde kvartal af 2024.

Illustration af den første af disse søgninger:

Sting**gray**s like to drink earl **grey** on a **gray** day.

# Regulære udtryk og søgning i tekst

Recall: per definition specificerer et regulært udtryk en mængde af strenge (dvs. et sprog).

# Regulære udtryk og søgning i tekst

Recall: per definition specificerer et regulært udtryk en mængde af strenge (dvs. et sprog).

## Plan for søgning via regulære udtryk:

1. Find et regulært udtryk  $R$ , som specificerer den mængde af strenge, som vi leder efter.
2. Skriv et program, som givet et regulært udtryk  $R$  og en streng  $s$  kan afgøre, om  $s$  starter med en streng i mængden specificeret af  $R$ .

# Regulære udtryk og søgning i tekst

Recall: per definition specificerer et regulært udtryk en mængde af strenge (dvs. et sprog).

## Plan for søgning via regulære udtryk:

1. Find et regulært udtryk  $R$ , som specificerer den mængde af strenge, som vi leder efter.
2. Skriv et program, som givet et regulært udtryk  $R$  og en streng  $s$  kan afgøre, om  $s$  starter med en streng i mængden specificeret af  $R$ .

Vi kan derefter bruge programmet på  $R$  og endestrenge (suffixer)  $s$  af teksten således:

Stingrays like to...  
tingrays like to...  
ingrays like to...  
ngrays like to...  
grays like to...  
rays like to...  
⋮

# Søgning med regulære udtryk

## Recap af plan:

1. Find et regulært udtryk  $R$ , som specificerer den mængde af strenge, som vi leder efter.
2. Skriv et program, som givet et regulært udtryk  $R$  og en streng  $s$  kan afgøre, om  $s$  starter med en streng i mængden specificeret af  $R$ .

# Søgning med regulære udtryk

## Recap af plan:

1. Find et regulært udtryk  $R$ , som specificerer den mængde af strenge, som vi leder efter.
2. Skriv et program, som givet et regulært udtryk  $R$  og en streng  $s$  kan afgøre, om  $s$  starter med en streng i mængden specificeret af  $R$ .

Punkt 1) kræver analyse, erfaring og kreativitet (lige som anden programmering). Vi så nogle eksempler på side 12–13.



# Søgning med regulære udtryk

## Recap af plan:

1. Find et regulært udtryk  $R$ , som specificerer den mængde af strenge, som vi leder efter.
2. Skriv et program, som givet et regulært udtryk  $R$  og en streng  $s$  kan afgøre, om  $s$  starter med en streng i mængden specificeret af  $R$ .

Punkt 1) kræver analyse, erfaring og kreativitet (lige som anden programmering). Vi så nogle eksempler på side 12–13.

Programmer fra punkt 2) findes færdigskrevet. De er tilgængelige på kommandolinien, i editorer, og som libraries i generelle programmeringssprog (Python, Java, Perl,...).

# Søgning med regulære udtryk

## Recap af plan:

1. Find et regulært udtryk  $R$ , som specificerer den mængde af strenge, som vi leder efter.
2. Skriv et program, som givet et regulært udtryk  $R$  og en streng  $s$  kan afgøre, om  $s$  starter med en streng i mængden specificeret af  $R$ .

Punkt 1) kræver analyse, erfaring og kreativitet (lige som anden programmering). Vi så nogle eksempler på side 12–13.

Programmer fra punkt 2) findes færdigskrevet. De er tilgængelige på kommandolinien, i editorer, og som libraries i generelle programmeringssprog (Python, Java, Perl,...).

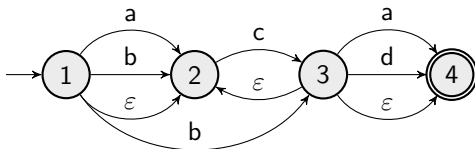
## Spørgsmål:

Hvordan virker disse programmer?

# Non-deterministic Finite Automaton with $\varepsilon$ -moves

En  $\varepsilon$ -NFA er som en DFA, men tillader desuden:

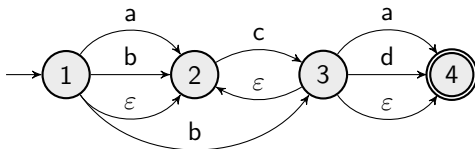
- ▶ Skridt ud af en state *uden* forbrug af et tegn fra strengen ( $\varepsilon$ -moves).
- ▶ *Flere* forskellige skridt med samme label (incl. label  $\varepsilon$ ) ud af en state ("non-determinisme").



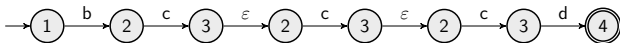
# Non-deterministic Finite Automaton with $\varepsilon$ -moves

En  $\varepsilon$ -NFA er som en DFA, men tillader desuden:

- ▶ Skridt ud af en state *uden* forbrug af et tegn fra strengen ( $\varepsilon$ -moves).
- ▶ *Flere* forskellige skridt med samme label (incl. label  $\varepsilon$ ) ud af en state (“non-determinisme”).



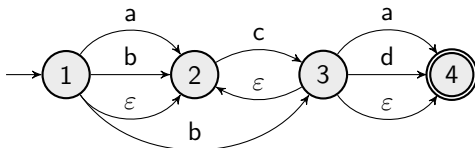
Strengen “bcccd” accepteres af ovenstående  $\varepsilon$ -NFA. Én mulig sti:



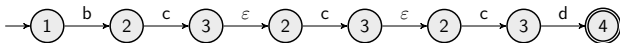
# Non-deterministic Finite Automaton with $\varepsilon$ -moves

En  $\varepsilon$ -NFA er som en DFA, men tillader desuden:

- ▶ Skridt ud af en state *uden* forbrug af et tegn fra strengen ( $\varepsilon$ -moves).
- ▶ *Flere* forskellige skridt med samme label (incl. label  $\varepsilon$ ) ud af en state (“non-determinisme”).



Strengen “bcccd” accepteres af ovenstående  $\varepsilon$ -NFA. Én mulig sti:

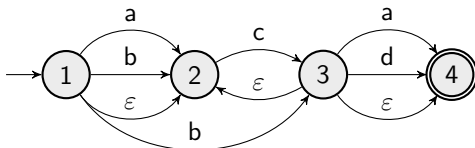


Hvilke af disse strenge accepteres: “aca”, “bcc”, “ad”, “c” og “acdd”?

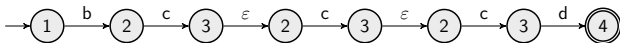
# Non-deterministic Finite Automaton with $\varepsilon$ -moves

En  $\varepsilon$ -NFA er som en DFA, men tillader desuden:

- ▶ Skridt ud af en state *uden* forbrug af et tegn fra strengen ( $\varepsilon$ -moves).
- ▶ *Flere* forskellige skridt med samme label (incl. label  $\varepsilon$ ) ud af en state ("non-determinisme").



Strengen "bcccd" accepteres af ovenstående  $\varepsilon$ -NFA. Én mulig sti:

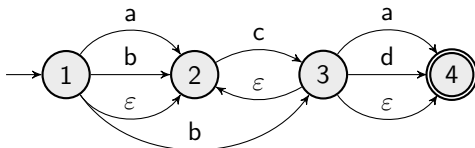


Hvilke af disse strenge accepteres: "aca", "bcc", "ad", "c" og "acdd"?

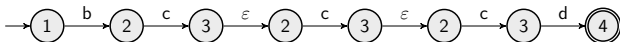
# Non-deterministic Finite Automaton with $\varepsilon$ -moves

En  $\varepsilon$ -NFA er som en DFA, men tillader desuden:

- ▶ Skridt ud af en state *uden* forbrug af et tegn fra strengen ( $\varepsilon$ -moves).
- ▶ *Flere* forskellige skridt med samme label (incl. label  $\varepsilon$ ) ud af en state (“non-determinisme”).



Strengen “bcccd” accepteres af ovenstående  $\varepsilon$ -NFA. Én mulig sti:



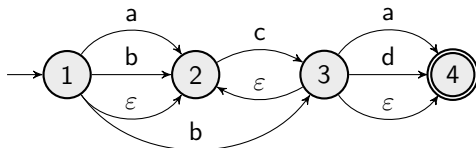
Hvilke af disse strenge accepteres: “aca”, “bcc”, “ad”, “c” og “acdd”?

Et regulært udtryk som beskriver hele sproget?

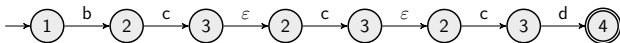
# Non-deterministic Finite Automaton with $\varepsilon$ -moves

En  $\varepsilon$ -NFA er som en DFA, men tillader desuden:

- ▶ Skridt ud af en state *uden* forbrug af et tegn fra strengen ( $\varepsilon$ -moves).
- ▶ *Flere* forskellige skridt med samme label (incl. label  $\varepsilon$ ) ud af en state (“non-determinisme”).



Strengen “bcccd” accepteres af ovenstående  $\varepsilon$ -NFA. Én mulig sti:



Hvilke af disse strenge accepteres: “aca”, “bcc”, “ad”, “c” og “acdd”?

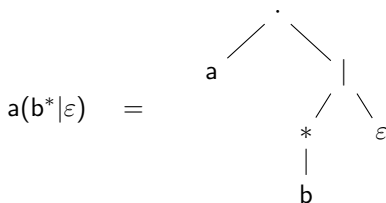
Et regulært udtryk som beskriver hele sproget? Fx.  $((a|b|\varepsilon)c|b)c^*(a|d|\varepsilon)$



# Thompsons algoritme: fra regulært udtryk til $\epsilon$ -NFA

Recall: regulære udtryk er bygge op hierarkisk via de basale udtryk empty string og single char, samt operatorerne concatenation, alternation og Kleene star.

|               | <i>Notation</i> | <i>Tilhørende sprog</i>                                                                  |
|---------------|-----------------|------------------------------------------------------------------------------------------|
| Empty string  | $\epsilon$      | $\{ \epsilon \}$                                                                         |
| Single char   | $c$             | $\{ c \}$                                                                                |
| Concatenation | $R_1 \cdot R_2$ | $\{ x \cdot y \mid x \in L_1, y \in L_2 \}$                                              |
| Alternation   | $R_1 \mid R_2$  | $L_1 \cup L_2$                                                                           |
| Kleene star   | $R^*$           | Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{ \epsilon \}$ |



# Thompsons algoritme: fra regulært udtryk til $\varepsilon$ -NFA

Thompsons algoritme: Konvertér udtrykstræ til  $\varepsilon$ -NFA nedefra og op.

# Thompsons algoritme: fra regulært udtryk til $\varepsilon$ -NFA

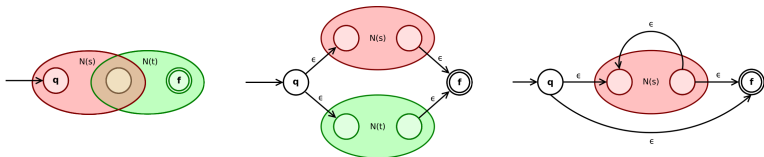
Thompsons algoritme: Konvertér udtrykstræ til  $\varepsilon$ -NFA nedefra og op.

En knude i udtrykstræet konverteres således (invariant: én accept state):

1) Base cases (empty string  $\varepsilon$ , single char  $a$ ):



2) Rekursive cases (concatenation  $s \cdot t$ , alternation  $s|t$ , Kleene star  $s^*$ ):



Her er  $N(s)$  og  $N(t)$  allerede konverterede  $\varepsilon$ -NFA'er for udtryk  $s$  og  $t$ .

# Thompsons algoritme: fra regulært udtryk til $\varepsilon$ -NFA

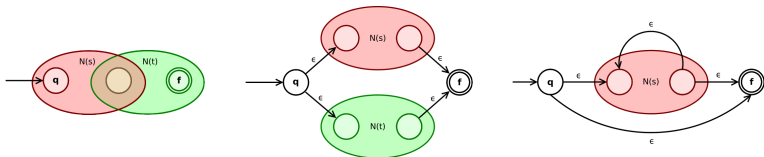
Thompsons algoritme: Konvertér udtrykstræ til  $\varepsilon$ -NFA nedefra og op.

En knude i udtrykstræet konverteres således (invariant: én accept state):

1) Base cases (empty string  $\varepsilon$ , single char a):



2) Rekursive cases (concatenation  $s \cdot t$ , alternation  $s|t$ , Kleene star  $s^*$ ):



Her er  $N(s)$  og  $N(t)$  allerede konverterede  $\varepsilon$ -NFA'er for udtryk  $s$  og  $t$ .

At  $\varepsilon$ -NFA og udtrykstræ specificerer samme sprog, kan argumenteres nedefra og op (formelt set: via induktion på højden af udtrykstræet).

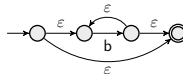
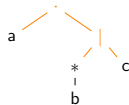
# Thompsons algoritme, et eksempel

Konvertering af **udtrykstræ** til  $\varepsilon$ -NFA nedefra og op:

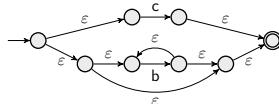
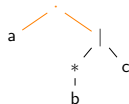
Alle deltræer med højde 0 er konverteret



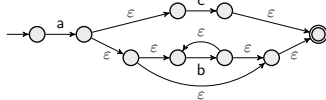
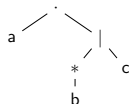
Alle deltræer med højde  $\leq 1$  er konverteret



Alle deltræer med højde  $\leq 2$  er konverteret



Alle deltræer med højde  $\leq 3$  er konverteret



## Søgning med et regulært udtryk

- ▶ Brug Thompsons algoritme til at konvertere udtrykket til en  $\varepsilon$ -NFA, der genkender samme sprog som udtrykket specificerer.
- ▶ Brug denne  $\varepsilon$ -NFA til at genkende strengene.

# Søgning med et regulært udtryk

- ▶ Brug Thompsons algoritme til at konvertere udtrykket til en  $\varepsilon$ -NFA, der genkender samme sprog som udtrykket specificerer.
- ▶ Brug denne  $\varepsilon$ -NFA til at genkende strengene.

Rekursiv algoritme for  $\varepsilon$ -NFA til at genkende/acceptere strenge i starten af en tekst T (dvs. som i T = “**gray**s like to...”):

```
def CHECK():  
    CHECK_REC(0, startknode)  
  
def CHECK_REC(i, v):  
    if v is an accept state:  
        report match with T[0:i]  
    for each edge (v, u) out of v:  
        if edge is an epsilon edge:  
            CHECK_REC(i, u) # don't advance in T  
        else:  
            if i < len(T) and char on edge is T[i]:  
                CHECK_REC(i+1, u) # advance in T
```

# Søgning med et regulært udtryk

- ▶ Brug Thompsons algoritme til at konvertere udtrykket til en  $\varepsilon$ -NFA, der genkender samme sprog som udtrykket specificerer.
- ▶ Brug denne  $\varepsilon$ -NFA til at genkende strengene.

Rekursiv algoritme for  $\varepsilon$ -NFA til at genkende/acceptere strenge i starten af en tekst T (dvs. som i T = "grays like to..."):

```
def CHECK():  
    CHECK_REC(0, startknode)  
  
def CHECK_REC(i, v):  
    if v is an accept state:  
        report match with T[0:i]  
    for each edge (v, u) out of v:  
        if edge is an epsilon edge:  
            CHECK_REC(i, u) # don't advance in T  
        else:  
            if i < len(T) and char on edge is T[i]:  
                CHECK_REC(i+1, u) # advance in T
```

**Invariant:** ved kald af CHECK\_REC(i, v) er der en sti gennem automaten som bruger bogstaverne i T[0:i] og som ender i knuden v. Så kun korrekte matches rapporteres.



# Søgning med et regulært udtryk

- ▶ Brug Thompsons algoritme til at konvertere udtrykket til en  $\varepsilon$ -NFA, der genkender samme sprog som udtrykket specificerer.
- ▶ Brug denne  $\varepsilon$ -NFA til at genkende strengene.

Rekursiv algoritme for  $\varepsilon$ -NFA til at genkende/acceptere strenge i starten af en tekst T (dvs. som i T = “**gray**s like to...”):

```
def CHECK():  
    CHECK_REC(0,startknode)  
  
def CHECK_REC(i,v):  
    if v is an accept state:  
        report match with T[0:i]  
    for each edge (v,u) out of v:  
        if edge is an epsilon edge:  
            CHECK_REC(i,u) # don't advance in T  
        else:  
            if i < len(T) and char on edge is T[i]:  
                CHECK_REC(i+1,u) # advance in T
```

**Invariant:** ved kald af CHECK\_REC(i,v) er der en sti gennem automaten som bruger bogstaverne i T[0:i] og som ender i knuden v. Så kun korrekte matches rapporteres.

Og omvendt: enhver matchende sti gennem automaten bliver fundet og rapporteret.



# Regulære udtryk i den virkelige verden (ikke pensum)

Mange *ekstra operatorer* er tilføjet til de klassiske (single char, concatenation, alternation, Kleene star) base cases og operatorer:

- ▶ Brugedefinerede klasser af tegn ([a-k], [0-9]).
- ▶ Pre-definerede klasser af tegn (. , \w, \d,...)
- ▶ Særlige positioner i strengen (^, \$, \b,...)
- ▶ Bounded repetition (?, +, {m,n},...)
- ▶ Grupper, som kan refereres til senere (( ), \1, \2,...)
- ▶ ...

Vær opmærksom på, at der er forskellige varianter (Perl, POSIX, GNU, mfl.). Læs altid på dokumentationen. Python bruger ca. Perl syntax.

- ▶ Python How-To:  
<https://docs.python.org/3/howto/regex.html>
- ▶ Python Reference:  
<https://docs.python.org/3/library/re.html>