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: Kompleksitet og beregnelighed* (6. semester).

Sprog

Definition:

Et sprog = en mængde af strenge.

Eksempler:

$\{$ agurk, gulerod, elefant $\}$

$\{$ a, b, ab $\}$

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

$\{$ a, b, ε $\}$

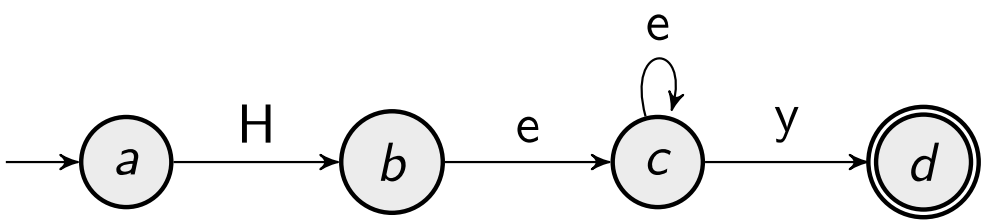
$\{$ $\}$

Her står ε for den tomme streng.

Sprog

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

Eksempel fra Kevins slides:

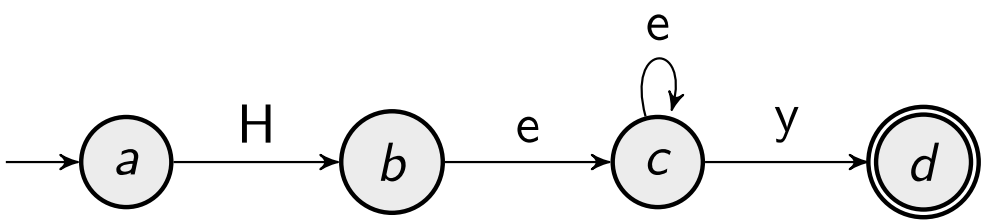


Hvad er sproget her?

Sprog

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

Eksempel fra Kevins 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 Kevins slides:

$$
\begin{aligned}
S &\to \text{Hello}\,T \\
S &\to \text{Hey}\,T \\
T &\to \text{there}
\end{aligned}
$$

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 Kevins slides:

$$\begin{aligned} S &\to \text{Hello}\,T \\ S &\to \text{Hey}\,T \\ T &\to \text{there} \end{aligned}$$

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 er meget anvendelige til at specificere søgninger i strenge (tekst).

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

Regulære udtryk

Definition:

Et regulært udtryk er en af følgende ting:

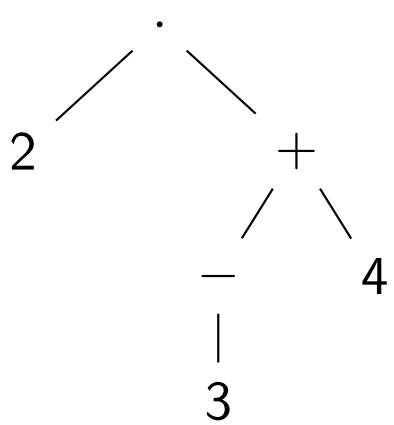
	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
Single char	c	$\{ \text{c} \}$
Concatenation	$R_1 \cdot R_2$	$\{x \cdot y \mid x \in L_1, y \in L_2\}$
Alternation	$R_1 \mid R_2$	$L_1 \cup L_2$
Kleene star	R^*	Alle strenge som kan fås via gentagen concatenation af strenge i $L \cup \{\varepsilon\}$

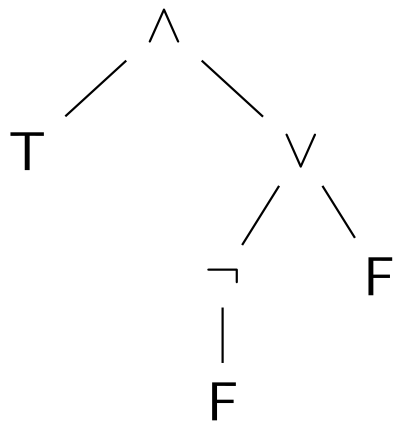
Her er R_1, R_2 og R allerede opbyggede regulære udtryk med tilhørende sprog L_1, L_2 og L. Strengen $x \cdot y$ er tegnene i strengen x efterfulgt af tegnene i strengen y.

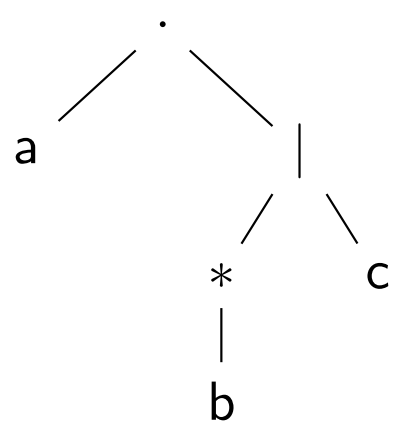
Definitionen er rekursiv. Base cases er **empty string** og **single char**. Rekursive cases, som bygger større regex'er og større tilhørende sprog ud fra mindre, er **concatenation**, **alternation** og **Kleene star**.

Udtrykstræer

Bemærk at der, lige som for normale algebraiske udtryk og for boolske udtryk, er tale om udtrykstræer, hvor beregningen går nedefra og op.







Input: tal	Input: Booleans	Input: Sprog
Output: tal	Output: Booleans	Output: Sprog
$2 \cdot ((-3) + 4)$	$T \wedge ((\neg F) \vee F)$	$a \cdot ((b^*) \mid c)$

Vi bruger parenteser til at lave en lineær beskrivelse af den hierarkiske struktur.

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 "$\cdot$".

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

Eksempler på regulære udtryk

Recap af definition:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
Single char	c	$\{ \text{c} \}$
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:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
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:

Regulært udtryk	*Tilhørende sprog*
abc|xy	{ abc, xy }

Eksempler på regulære udtryk

Recap af definition:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
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:

Regulært udtryk	*Tilhørende sprog*
abc|xy	{ abc, xy }
(ε|a)bc	{ bc, abc }

Eksempler på regulære udtryk

Recap af definition:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
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:

Regulært udtryk	*Tilhørende sprog*
abc|xy	{ abc, xy }
(ε|a)bc	{ bc, abc }
ε|abc	{ ε, abc }

Eksempler på regulære udtryk

Recap af definition:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
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:

Regulært udtryk	*Tilhørende sprog*
abc|xy	$\{$ abc, xy $\}$
(ε|a)bc	$\{$ bc, abc $\}$
ε|abc	$\{ \varepsilon$, abc $\}$
a*	$\{ \varepsilon$, a, aa, aaa, ...$\}$

Eksempler på regulære udtryk

Recap af definition:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
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:

Regulært udtryk	*Tilhørende sprog*
abc|xy	{ abc, xy }
(ε|a)bc	{ bc, abc }
ε|abc	{ ε, abc }
a*	{ ε, a, aa, aaa, ...}
ba$^*c$	{ bc, bac, baac, baaac, ...}

Eksempler på regulære udtryk

Recap af definition:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
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:

Regulært udtryk	*Tilhørende sprog*
abc|xy	{ abc, xy }
(ε|a)bc	{ bc, abc }
ε|abc	{ ε, abc }
a*	{ ε, a, aa, aaa, ...}
ba*c	{ bc, bac, baac, baaac, ...}
ac|b*	{ ac, ε, b, bb, bbb, bbbb,...}

Eksempler på regulære udtryk

Recap af definition:

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
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:

Regulært udtryk	*Tilhørende sprog*
abc|xy	{ abc, xy }
(ε|a)bc	{ bc, abc }
ε|abc	{ ε, abc }
a*	{ ε, a, aa, aaa, ...}
ba*c	{ bc, bac, baac, baaac, ...}
ac|b*	{ ac, ε, b, bb, bbb, bbbb,...}
(ac|b)*	{ ε, ac, b, acb, bac, acacb, bacb, acacacbbbacb,...}

Flere eksempler

Regulært udtryk	*Tilhørende sprog*
grey|gray	{ grey, gray }

Flere eksempler

Regulært udtryk	*Tilhørende sprog*
grey|gray	{ grey, gray }
gr(e|a)y	{ grey, gray }

Flere eksempler

Regulært udtryk	*Tilhørende sprog*
grey|gray	$\{$ grey, gray $\}$
gr(e|a)y	$\{$ grey, gray $\}$
$1(0\|1)^*$	alle binære heltal ≥ 1

Flere eksempler

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

Flere eksempler

Regulært udtryk	*Tilhørende sprog*
grey|gray	$\{$ grey, gray $\}$
gr(e|a)y	$\{$ grey, gray $\}$
1(0|1)*	alle binære heltal ≥ 1
␣␣␣*	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 ≥ 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 ≥ 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 ≥ 4
  Svar: $1(0|1)(0|1)(0|1)^*$
- Alle datoer i fjerde kvartal i år, i formatet 2022.11.03

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 ≥ 4
  Svar: $1(0|1)(0|1)(0|1)^*$
- Alle datoer i fjerde kvartal i år, i formatet 2022.11.03
  Svar: $2022.1((0|1|2).((0|1|2)(1|2|3|4|5|6|7|8|9)|(1|2|3)0)|(0|2).31)$

Hvorfor hedder det “Kleene star”?

Hvorfor hedder det “Kleene star”?

Stephen C. Kleene (1909-1994)

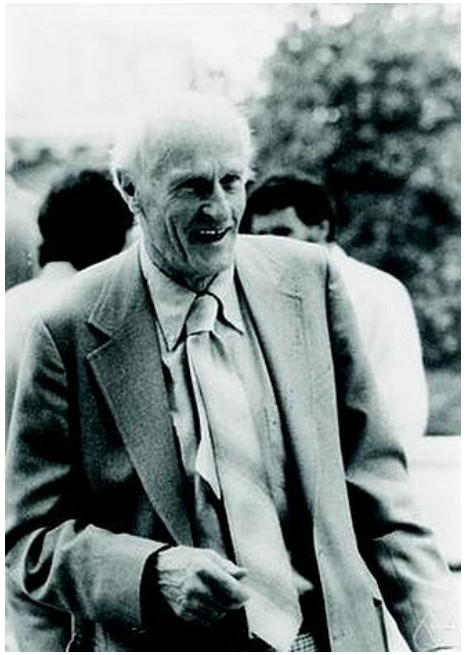


Foto: Konrad Jacobs

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 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 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 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.

Søgning i tekst

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

Eksempel: find alle steder, hvor "dog" optræder 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.

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 2022.

Søgning i tekst

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

Eksempel: find alle steder, hvor "dog" optræder 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.

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 2022.

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 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.

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 2022.

Illustration af den første af disse søgninger:

Stingrays 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.

Regulære udtryk og søgning i tekst

Recall: per definition specificerer et regulært udtryk en mængde af strenge.

Søgning med 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.

Søgning med 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

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

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

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

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,...).

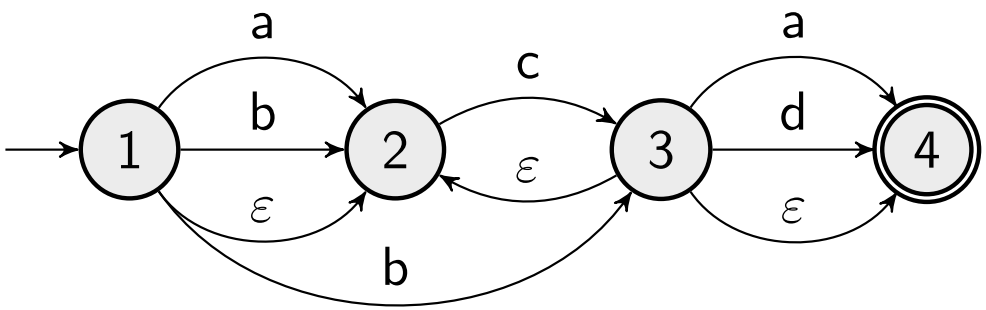
Spørgsmål:

Hvordan virker disse programmer?

Non-deterministic Finite Automaton with ε-moves

En ε-NFA er som en DFA, men tillader desuden:

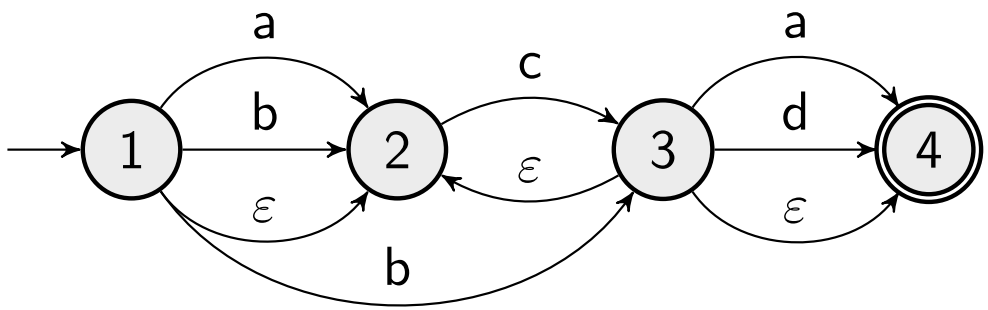
- Skridt ud af en state uden forbrug af et tegn fra strengen (ε-moves).
- Flere forskellige skridt med samme label (incl. label ε) ud af en state ("non-determinisme").



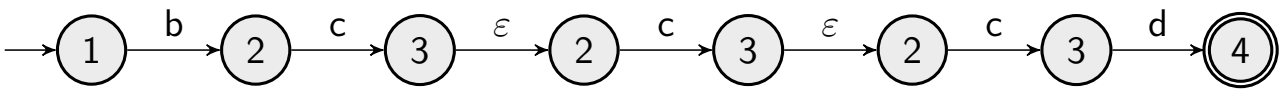
Non-deterministic Finite Automaton with ε-moves

En ε-NFA er som en DFA, men tillader desuden:

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



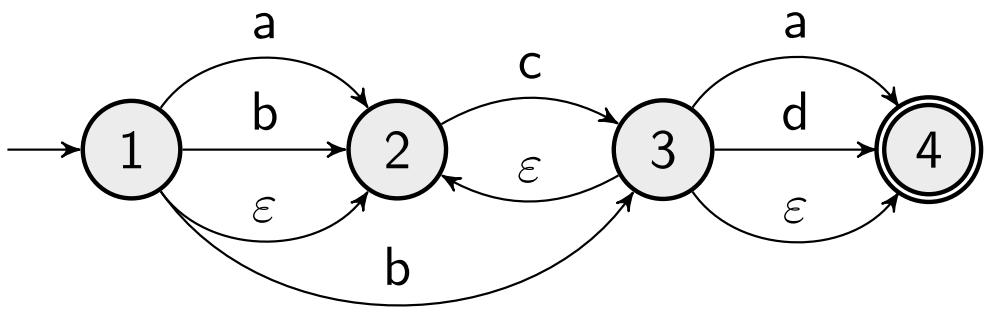
Strengen "bcccd" accepteres af ovenstående ε-NFA. Én mulig sti:



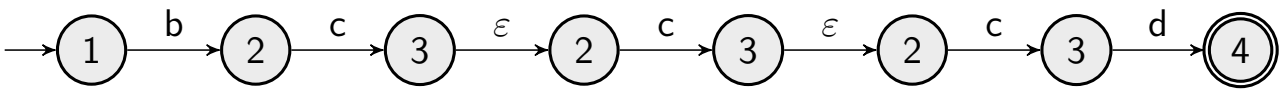
Non-deterministic Finite Automaton with ε-moves

En ε-NFA er som en DFA, men tillader desuden:

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



Strengen "bcccd" accepteres af ovenstående ε-NFA. Én mulig sti:

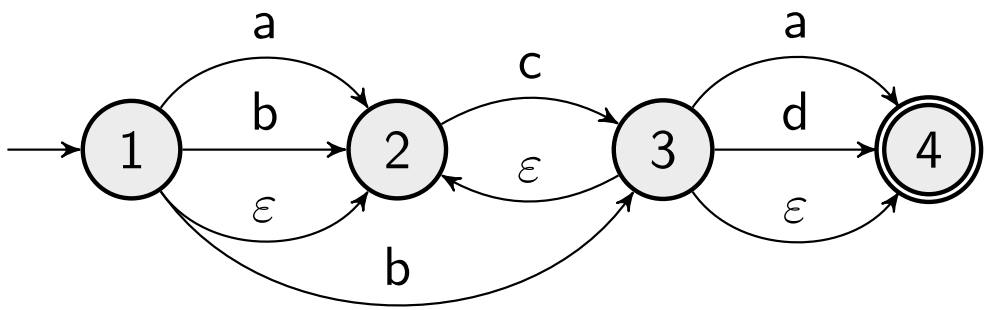


Hvilke af disse strenge accepteres: "aca", "bcc", "ad", "c" og "acdd"?

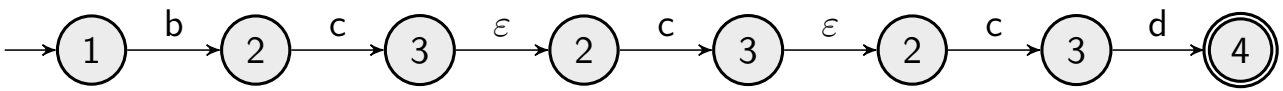
Non-deterministic Finite Automaton with ε-moves

En ε-NFA er som en DFA, men tillader desuden:

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



Strengen "bcccd" accepteres af ovenstående ε-NFA. Én mulig sti:

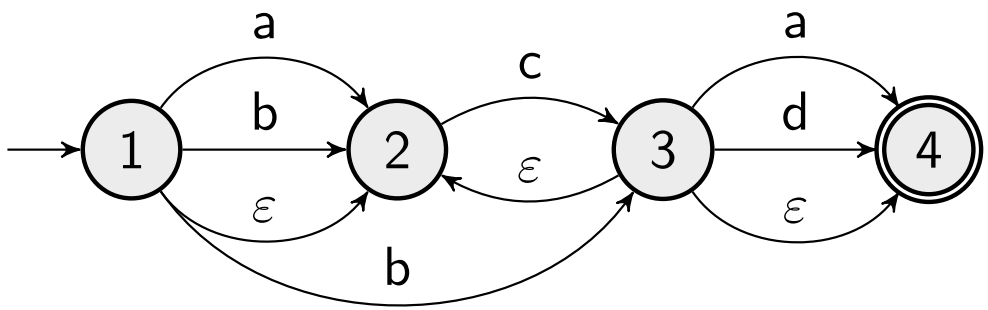


Hvilke af disse strenge accepteres: "aca", "bcc", "ad", "c" og "acdd"?

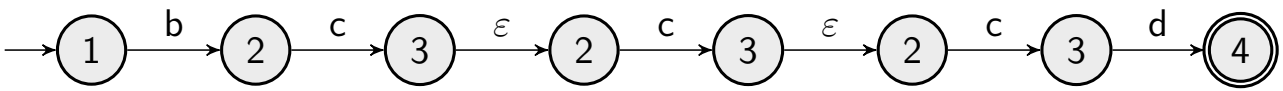
Non-deterministic Finite Automaton with ε-moves

En ε-NFA er som en DFA, men tillader desuden:

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



Strengen "bcccd" accepteres af ovenstående ε-NFA. Én mulig sti:



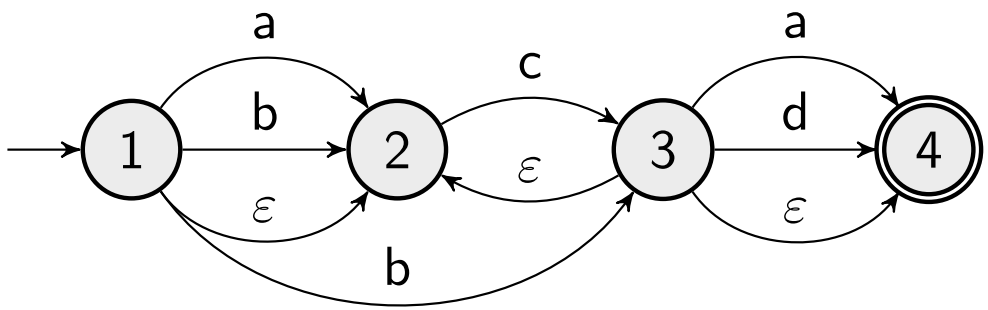
Hvilke af disse strenge accepteres: "aca", "bcc", "ad", "c" og "acdd"?

Et regulært udtryk som beskriver hele sproget?

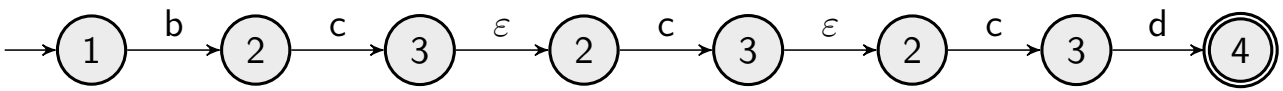
Non-deterministic Finite Automaton with ε-moves

En ε-NFA er som en DFA, men tillader desuden:

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



Strengen "bcccd" accepteres af ovenstående ε-NFA. Én mulig sti:



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

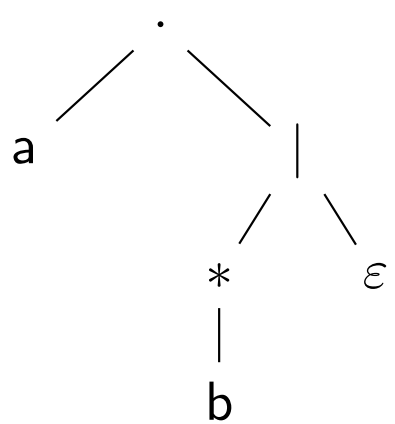
Et regulært udtryk som beskriver hele sproget? Fx. $((a|b|\varepsilon)c|b)c^*(a|d|\varepsilon)$

Thompsons algoritme: fra regulært udtryk til ε-NFA

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

	Notation	*Tilhørende sprog*
Empty string	ε	$\{ \varepsilon \}$
Single char	c	$\{ \text{c} \}$
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 $L \cup \{\varepsilon\}$

$\text{a}(\text{b}^*|\varepsilon)$ =



Thompsons algoritme: fra regulært udtryk til ε-NFA

Thompsons algoritme: Konvertér udtrykstræ til ε-NFA nedefra og op.

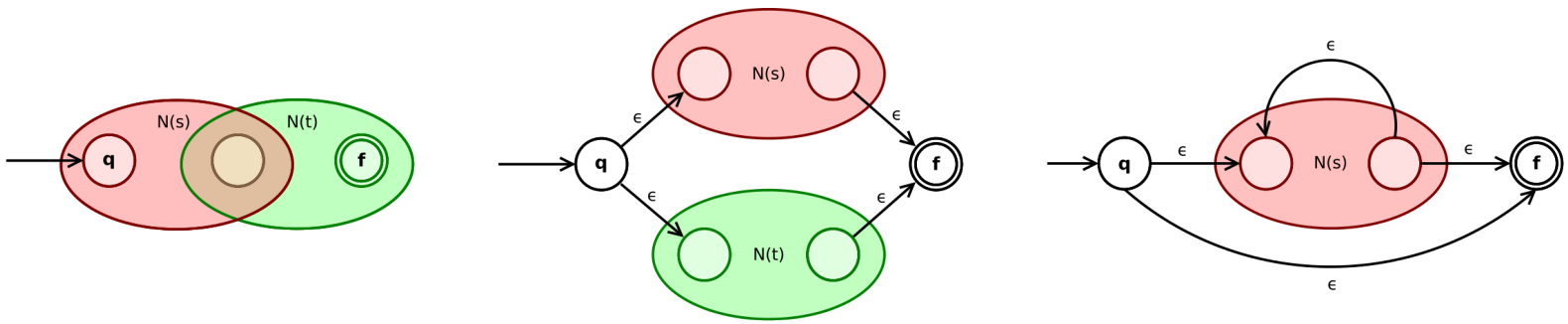
Thompsons algoritme: fra regulært udtryk til ε-NFA

Thompsons algoritme: Konvertér udtrykstræ til ε-NFA nedefra og op.

En knude i udtrykstræet konverteres således:

1) Base cases (empty string ε, single char a):

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



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

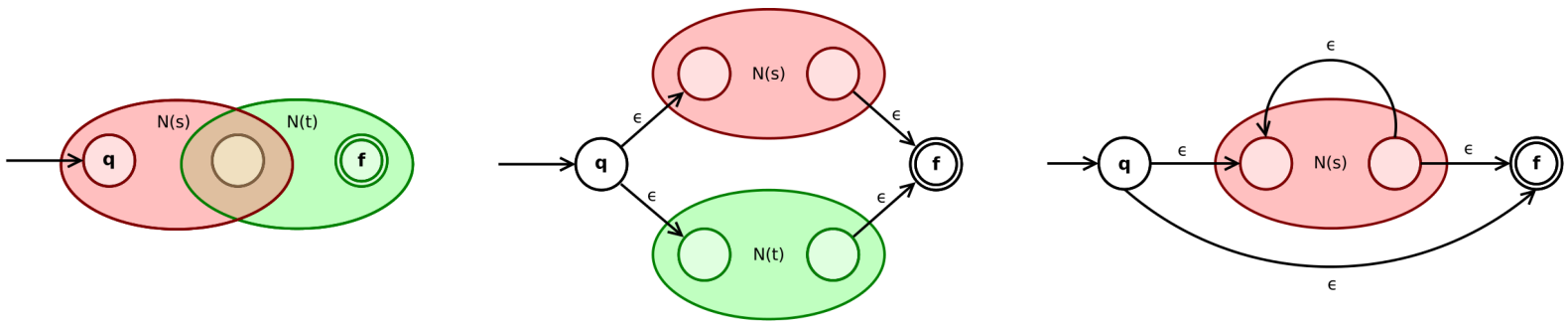
Thompsons algoritme: fra regulært udtryk til ε-NFA

Thompsons algoritme: Konvertér udtrykstræ til ε-NFA nedefra og op.

En knude i udtrykstræet konverteres således:

1) Base cases (empty string ε, single char a):

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



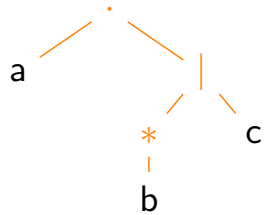
Her er N(s) og N(t) allerede konverterede ε-NFA'er for udtryk s og t.

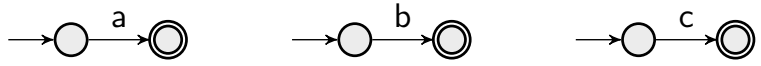
At ε-NFA og udtrykstræ specificerer samme sprog, kan argumenteres nedefra og op (formelt set: via induktion på højden af udtrykstræet).

Thompsons algoritme, et eksempel

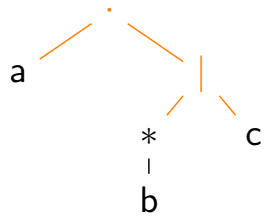
Konvertering af udtrykstræ til ε-NFA nedefra og op:

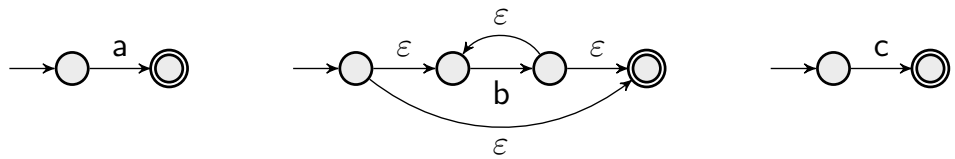
Alle deltræer med højde 0 er konverteret



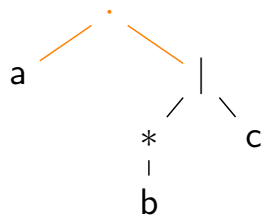


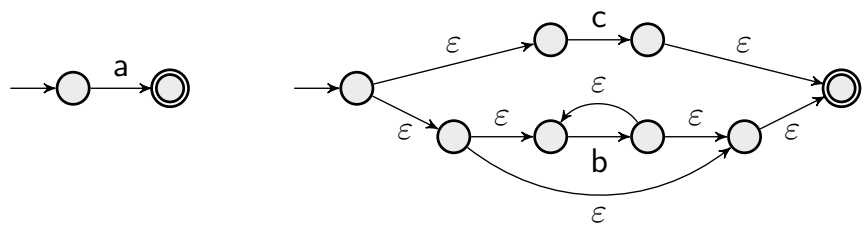
Alle deltræer med højde ≤ 1 er konverteret



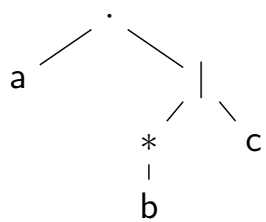


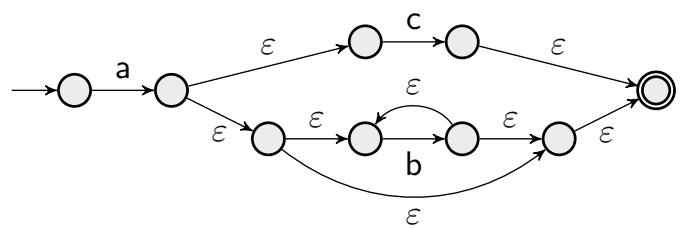
Alle deltræer med højde ≤ 2 er konverteret





Alle deltræer med højde ≤ 3 er konverteret





Søgning med et regulært udtryk

- Brug Thompsons algoritme til at konvertere udtrykket til en ε-NFA, som genkender samme sprog som udtrykket specificerer.
- Brug denne ε-NFA til at genkende strengene.

Søgning med et regulært udtryk

- Brug Thompsons algoritme til at konvertere udtrykket til en ε-NFA, som genkender samme sprog som udtrykket specificerer.
- Brug denne ε-NFA til at genkende strengene.

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

```
def CHECK():
  CHECK_REC(-1,startknude)

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+1 < len(T) and char on edge is T[i+1]:
        CHECK_REC(i+1,u) # advance in T
```

Søgning med et regulært udtryk

- Brug Thompsons algoritme til at konvertere udtrykket til en ε-NFA, som genkender samme sprog som udtrykket specificerer.
- Brug denne ε-NFA til at genkende strengene.

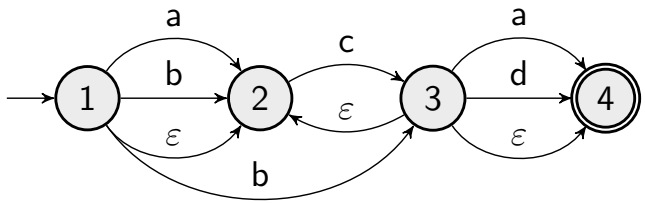
Rekursiv algoritme for ε-NFA til at genkende/acceptere strenge i starten af en tekst `T` (dvs. som i `T` = "grays like to..."):

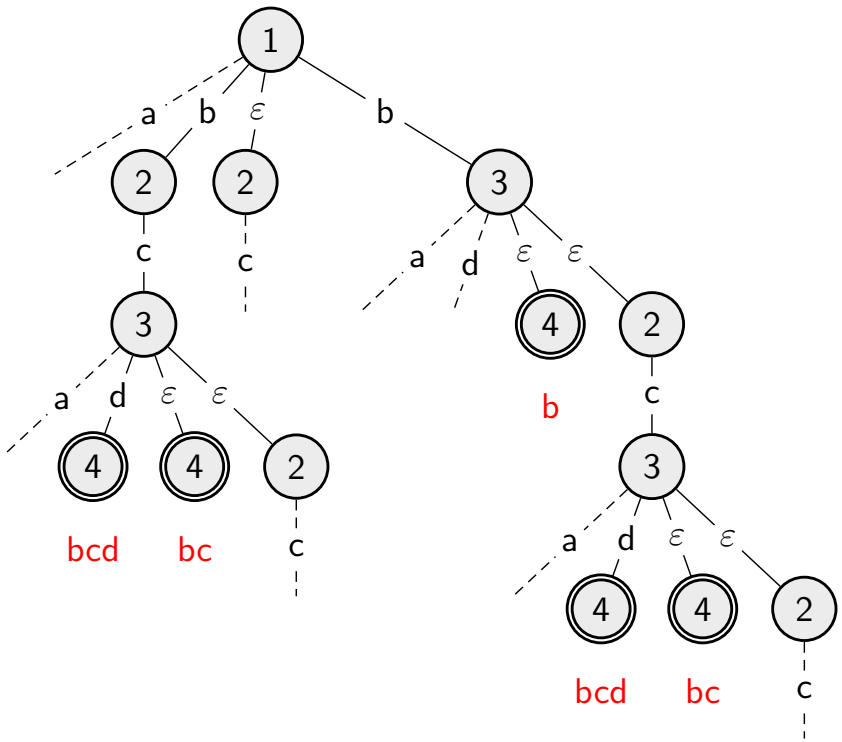
```
def CHECK():
  CHECK_REC(-1,startknude)

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+1 < len(T) and char on edge is T[i+1]:
        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`.

Eksempel

Hvis algoritmen bruges på strengen $\mathtt{T}$ = "bcde" samt automaten til højre, har rekursionstræet nedenstående facon. De matchede strenge er vist med rødt. Stiplede kanter følges ikke.





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:

- Brugerdefinerede klasser af tegn ([a-k], [0-9]).
- Pre-definerede klasser af tegn (\w, \d,...)
- Særlige positioner i strengen ($\hat{}$, $, \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`