INF 2820 V2016: Obligatorisk innleverinsoppgave 1

OBS – Korrigert eksemplene oppgave 2, 8.2

- **Besvarelsene skal leveres i devilry innen torsdag 18.2 kl 18.00**
- **Filene det vises til finner du på /projects/nlp/inf2820/fsa på IFIs linuxmaskiner**
- **Les reglementet for obligatoriske oppgaver:**
  http://www.mn.uio.no/ifi/studier/admin/obliger/index.html
  Dette er en individuell oppgave. Det skal ikke leveres felles besvarelser.
- **Poengene antyder arbeidsmengden på hvert punkt. Det er 100 poeng i alt.**

Oppgave 1: Endelige tilstandsmaskiner (20 poeng)

Denne oppgaven kan gjøres i JFLAP. Du anbefales likevel å løse den med papir og penn først for å få eksamenstrening. Så kan du bruke JFLAP til å kontrollere løsningen din.

a. Lag en ikke-deterministisk endelig tilstandsmaskin (NFA) som beskriver språket L1=L(*a*b(a+c)* + ac(b+a)*), der alfabetet er A={*a, b, c*}. (Symbolet + er her disjunksjon).
b. Lag en deterministisk maskin (DFA) som beskriver det samme språket.
c. Lag en tilstandsmaskin som beskriver komplementspråket til L1.
d. Hvilke av følgende uttrykk er i L1?
   i. abc
   ii. acb
   iii. bac
   iv. bbc
   v. aaaa
   vi. aaab
   vii. aaba
   viii. abaa
   ix. abab
   x. baaa

Innlevering: Svar på de fire punktene. Hvis du løser oppgaven med papir og penn og tegner diagrammer, er det tilstrekkelig å ta et tydelig bilde av hvert diagram og levere disse. Bruker du JFLAP, kan du lagre diagrammene som JPEG-filer og levere.

Oppgave 2: Gjenkjenning med DFA (15 poeng)

a) Se på filen *dfa_recog.py*, som du finner i mappen /projects/nlp/inf2820/fsa/ på IFIs linux-maskiner. Skriv inn maskinen fra oppgave 1b på dette formatet. Prøv den ut på noen strenger med trace=1. Overbevis deg selv om at DFA-en din gjør det den skal og at du forstår programmet *dfa_recog.py*.

Innlevering: Python-representasjon av automaten.

b) La L2 være språket med alfabet A = {a, b} med følgende egenskaper:

- Et velformet uttrykk skal inneholde et odde antall *a*-er, altså 1, 3, 5, 7, eller …
- *b*-ene skal komme i par, dvs. enhver *b* skal ha en annen *b* rett til venstre for seg, eller rett til høyre for seg, men ikke på begge sider

For eksempel skal følgende uttrykk være i språket

i. a
ii. bba
iii. abbaa
iv. abbabbabbaaaa
v. bbaaa

Mens følgende uttrykk ikke skal være i språket

i. b
ii. aa
iii. ababa
iv. abbbaa
v. abbaabbaaa

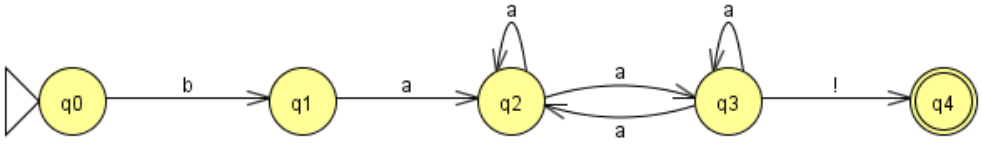
Lag en DFA for dette språket, skriv den inn på formatet for *dfa_recog.py* og test den på eksemplene over.
Innlevering: Python-representasjon av automaten og kjøring med de 10 eksempelsetningene.

c) Lag et regulært uttrykk for L2. Skriv det som et Python regulært uttrykk og test det på eksemplene over som beskrevet i oppgave 5 på ukesoppgavesett 1.) (Denne kan være litt vanskelig. Du anbefales å gjøre oppgavene 3-5 først. Det er viktigere at du arbeider med dem enn at du blir stående fast på denne.)

Innlevering: Det regulære uttrykket på Python-format og resultatet av kjøring på de 10 eksempelsetningene.

Oppgave 3: Algoritmer og implementasjoner av NFA-er (5 poeng)

Vi skal studere forskjellige algoritmer for NFA-er, og se på Python-implementasjonene *nfa_naive_recog.py* og *nfa_smart_recog.py*. Filen *sheep2.nfa* inneholder en representasjon av automaten (som er laget svært flertydig for å gjøre poenget tydelig).



a) Bruk et interaktivt python-vindu (fra idle eller ipython eller lignende) og kjør *nfa_naive_recog.py*. Skriv inn automaten som svarer til nettverket eller les den inn ved

- *nfa2 = NFAFromFile('sheep2.nfa').*

(For at kommandoen skal få denne formen, må *sheep2.nfa* ligge i samme mappe som programmet. Ligger den i en annen mappe, må vi skrive stien til filen. I idle og ipython får du autofullføringshjelp til å lete frem stier.)

Bli kjent med representasjonen av automaten ved å gi en del kommandoer som

- *nfa2.start*
- *nfa2.edges*
- *nfa2.finals*

Bli kjent med *naiveenrec*. Prøv automaten på en del strenger, både noen som skal aksepteres og noen som ikke skal aksepteres, for eksempel

- *naiveenrec("baaaaaab", nfa2, 1)*

og overbevis deg om at du skjønner hva programmet gjør. (Du behøver ikke studere innlesningsrutinen NFAFromFile.)

b) Gjør det samme med *nfa_smart_recog.py*.

Innlevering: Kjøringseksempel med de to programmene med "trace" av om "baaaaab" anerkjennes.

Oppgave 4: Forkortelser i NFA (20 poeng)

I bruk av regex ser en store besparelser i forkortelser. For eksempel kan vi skrive [a-z] for alle små engelske bokstaver. Også i NFA-er er det mye å spare på forkortelser. La oss si at vi skal gjenkjenne stavelsesstruktur og vil skille mellom konsonanter og vokaler. I en gitt tilstand, f.eks. q3, skal alle konsonanter føre til samme tilstand q4, mens vokalene skal føre til en annen tilstand q5. Uten forkortelser kan vi lage tjue kanter fra q3 til q4, en for hver konsonant. Hvis det også er slik at alle konsonanter fører fra q6 til q7 og fra q8 til q9, må vi også lage tjue kanter fra q6 til q7 og fra q8 til q9. Her kan vi spare på å bruke forkortelser.

- Vi innfører en passende betegnelse, f.eks. CONS
- Vi gjør det til en del av spesifikasjonen av automaten at CONS står for b, c, d, f, g, …osv, z
- Der alle konsonanter fører fra en tilstand til samme tilstand, som fra q3 til q4, innfører vi ikke 20 kanter, vi innfører bare én kant og merker den med CONS.
- Tolkningen av dette er at hvis en kant er merket med en forkortelse, som CONS, så skal alle symboler som er i mengden navngitt av forkortelsen føre oss fra den første til den andre tilstanden.

I programmet *nfa_smart_recog.py* har vi inkludert en måte for å representere forkortelser – kalt *abrs* i automatene – og i rutinene for å lese inn automater. Se på fila *template_abrs.nfa* for å se et eksempel på hvordan vi vil representere en automat med forkortelser i en fil. Prøv så

- nfa3=NFAFromFile('automata/template_abrs.nfa')
- >>> nfa3.edges
- >>> nfa3.abrs

for å se hvordan forkortelser representeres i programmet. (Du behøver ikke bry deg om innlesningsrutinen utover dette.)

a) Anerkjenningsprosedyren *nrec()* virker fint for automater uten forkortelser, men den kan ikke behandle forkortelser. Modifiser *nrec()* til også å kunne bruke automater med forkortelser.

Innlevering: Modifisert *nfa_smart_recog.py* med kommentarer der det er gjort endringer. Kjøringseksempel *med template_abrs.nfa*.

b) For å se litt mer av gevinsten av forkortelser skal vi se på automaten fra figur 2.16 fra Jurafsky og Martin som beskriver uttrykk i dollar og cents. Lag en alternativ representasjon av det samme språket der du bruker forkortelser.

Innlevering: Den modifiserte automaten som en fil som kan leses av NFA-programmet.

Oppgave 5: NFA for norske setninger (40 poeng)

Vi skal lage en NFA for noen norske setninger. Vi skal passe på at den ikke får med for mye. Alt som anerkjennes skal være grammatiske norske setninger. Vi kan selvsagt ikke forvente å beskrive alle mulige norske setninger.

Byggestenene vil være norske ord, som *ga*, *barnet, huset*, mao. vil disse utgjøre "alfabetet" for det formelle språket.

Ord kan deles inn i **ordklasser**. De største og viktigste er

- N (*substantiv* eller *nomen*): Kari, barn, barnet, …
- V (*verb*): ga, solgte, spiste, …
- A (*adjektiv*): stort, pent, underlig, …
- P (*preposisjoner*): fra, til, på,…

Innenfor ordklassene kan det være **underklasser**. For substantiv skiller vi mellom egennavn, *Kari*, og fellesnavn, *barn*. For verb skiller vi bl.a. mellom intransitive, *sov*, transitive, *anerkjenne*, og ditransitive, *ga*.

Ord kommer også i forskjellige **former**, som bestemt, *barnet* og ubestemt, *barn*.

Klasse, underklasse og form er bestemmende for hvor et ord kan forekomme. To ord som tilhører samme klasse, underklasse og form, f.eks. *barnet* og *huset* kan forekomme i de samme posisjonene. I en grammatisk setning kan vi skifte ut det ene med det andre og stadig få en grammatisk setning. To forskjellige former av et ord kan ikke alltid skiftes. Vi kan si *et barn*, men ikke *et barnet*. Det er også grunnen til å operere med underklasser. Både *bil* og *jente* er substantiv, men vi kan ikke si *et jente sov* selv om vi kan si *et barn sov*.

I denne oppgaven kan vi dele ord inn i grupper ut i fra ordklasse og evt. underklasse og form. Så kan vi bruke forkortelser som navn på slike grupper, og lage en NFA som beskriver setninger basert på forkortelser og ord.

Vi har laget et lite testsett med setninger som skal være med i språket, kalt *norsk_testset.txt*. Setningene merket med + skal være med i språket. For å teste at vi ikke får med for mye, har vi også tatt med negative eksempler som ikke er grammatiske setninger og merket dem med -.

Å lage en NFA som beskriver nøyaktig de 8 setningene er trivielt. Vi ønsker en NFA som beskriver en mer generell klasse. I det følgende beskriver vi i lingvistiske termer litt mer om hva som i hvert fall bør være med i språket.

Som strategi kan det være lurt å først lage nettverket for svært enkle setninger som

- Kari smilte.
- Barnet sov.

og så utvide det gradvis til å inkludere mere. Slike enkle setninger er bygget opp av et substantivledd (NP), *barnet*, og et verballedd (VP), *sov*.

NP

Vi vil ha med flere NP-ledd, altså uttrykk som kan ta samme plassen som *Kari* eller *barnet* i eksemplene. For NP vil vi begrense oss til (i) egennavn, (ii) substantiv i bestemt form entall, (iii) nøytrumsubstantiv i ubestemt form entall med en foranstilt bestemmer (*et*, *ethvert*, *noe*, …). For disse ubestemte vil vi også ta med et ubegrenset antall modifiserende adjektiv (*pent, stort, lite, …).*

Skal være med i språket	Skal ikke være med i språket
Kari sov.	
Barnet sov.	
Et barn sov.	
Et lite pent barn sov.	Et pene barn sov.
Kari sa at et lite barn sov.	

VP

Vi vil ha med intransitive (*sov*), transitive (*solgte)* og ditransitive (*overrakte*) verb. Noen verb kan være både f.eks. transitive og intransitive, andre kan bare være i en underklasse.

Skal være med i språket	Skal ikke være med i språket
Kari ga barnet huset	Kari ga
Kari overrakte dyret eplet	Kari ga dyret
Kari solgte huset	Kari solgte
Kari spiste eplet	Kari spiste barnet eplet
Kari spiste	Kari sov eplet
Kari smilte	

Vi skal også ta med verb som tar setningskomplement. Noen av disse tar både et nominalkomplement (*barnet*) og et setningskomplement (*at Ola sa at dyret smilte* i det siste eksemplet).

Skal være med i språket	Skal ikke være med i språket
Kari fortalte barnet at dyret smilte	Kari sa barnet at dyret smilte
Kari fortalte at dyret smilte	Kari solgte at dyret smilte.
Kari sa at dyret smilte	
Kari fortalte barnet at Ola sa at dyret smilte.	

Merk at denne konstruksjonen kan gjentas som i det siste eksempelet. Den kan gjentas et ubegrenset antall ganger.

PP

Vi tar med preposisjonalledd, disse kan modifisere et nominalledd eller VP.

Skal være med	Type
Dyret med barnet sov.	NP-modifikasjon
Dyret sov i huset.	VP-modifikasjon
Dyret med barnet sov i huset.	Begge typer i denne setningen.

NFA-en skal skrives på samme format som *template_abrs.nfa*. Det skal være minst fem forskjellige ord i hver (under)ordklasse (Negennanvn, Nfellesnavn, P, V, A) og minst tre verb av hver type av verb (intransitiv, verb som tar NP-komplement, verb av typen "V at …", osv.) Vi vil ha flere ord i hver klasse for å få prinsipielle løsninger og unngå snarveier. Alle ordene i i *norsk_testset.txt* skal være med.

Sjekk at NFA-en virker med (din modifiserte) *nfa_smart_recog.py*. Test nfa-en din på eksemplene i *norsk_testset.txt*. Du kan bruke funksjonen *test()* fra *test.py.* Se i dokumentasjonen i fila hvordan du skal bruke den. Hvis du ikke vet hvordan du skal lese inn funksjoner fra flere filer, kan du sette sammen filene *nfa_smart_recog.py* og *test.py* til en fil.

Innlevering: Filen med NFA og resultatet av å teste med *test*().

Oppgave 6: E-transisjoner (0 poeng)

Denne oppgaven er til de som ønsker litt større utfordringer. Den gir ikke poeng, men den har læringsverdi og den er morsom.

Hverken *nfa_naive_recog.py* eller *nfa_smart_recog.py* kan bruke ε-transitioner (hoppekanter). Vi kan skrive en ε-transisjon som: 5 '#' 7. Innlesingsrutinen tar høyde for hoppekanter i automatene. Du kan se på *template_abrs_epsilon.nfa* og hvordan den representeres etter innlesning med *NFAFromFile*. Vi skal prøve å modifisere NFA-anerkjenningen til å takle slike kanter.

Her kan en ta to ulike tilnærminger. Så lenge det ikke er hoppekanter vil hvert trinn (i) flytte seg en kant og (ii) bevege seg en posisjon i input. Med hoppekanter kan vi velge mellom å tenke på et trinn som

i. Å flytte seg en kant (og ikke flytte seg i input hvis dette er en hoppekant), eller
ii. Å flytte seg en posisjon i input (og samtidig flytte seg flere kanter hvis det er hoppekanter)

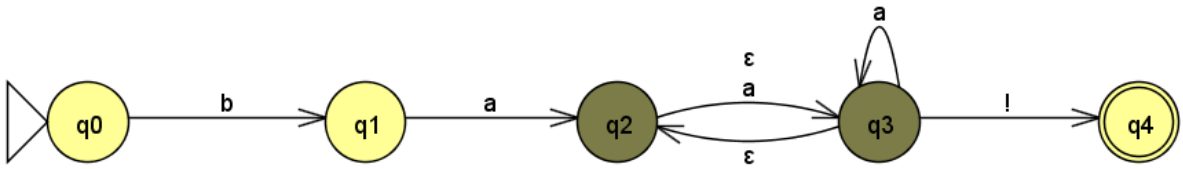
JFLAP illustrerer forskjellen mellom de to med "step by state" og "step with closure".

a) Vi ser først på den naive baktrackingstilnærmingen fra from *nfa_naive_recog.py*. Denne kan modifiseres forholdsvis enkelt til å takle hoppekanter med strategi (i) hvis vi antar at det ikke er noen løkker av hoppekanter hvor programmet kan gå i ring. Modifiser *nfa_naive_recog.py* til å takle hoppekanter så sant de ikke går i ring.

Innlevering: Modifisert *nfa_naive_recog.py* med kommentarer der det er gjort endringer.

b) **For de som er klar for en litt større utfordring**
For å kunne behandle også det tilfellet at hoppekanene går i ring, skal vi følge strategi (ii). Vi vil gjøre bruk av epsilontillukningen til en tilstand. Lag en prosedyre som tar en tilstand og en NFA som argumenter og beregner epsilontillukningen til tilstanden. Bruk dette til å modifisere *nfa_smart_recog.py* til å takle epsilontransisjoner.

Du kan teste resultatet på den litt ekstreme NFA-en



Innlevering: Modifisert *nfa_smart_recog.py* med kommentarer der det er gjort endringer.

Lykke til!