DD1360 Programmeringsparadigm: Artificiella språk och syntaxanalys, föreläsning 4

Karl Palmskog

KTH

27 mars, 2023

Förra föreläsningen

- Lexikal analys
  - tecken och tokens
- Härledningar och syntaxträd
  - härledningskedjor och trädkonstruktion
  - semantik genom trädtraversering
- Parsning genom rekursiv medåkning
  - rättfram strategi för att implementera vissa grammatiker
  - används ofta för att parsa programspråk
- Eliminering av tvetydighet
  - faktorisering av grammatiker
  - anpassning av rekursiv medåkning för associativitet

Dagens föreläsning

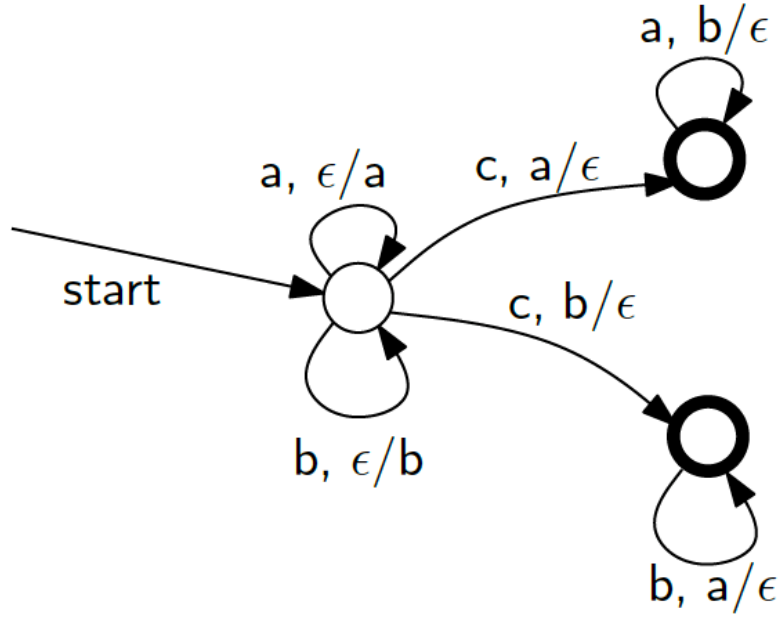
1. Stackautomater
2. Klasser av språk och grammatiker
3. Parsergeneratorer
4. Sammanfattning av kursens syntaxdel
5. Exempeluppgifter

1. Stackautomater
2. Klasser av språk och grammatiker
3. Parsergeneratorer
4. Sammanfattning av kursens syntaxdel
5. Exempeluppgifter

Stackautomater

En **stackautomat** (DPDA, Deterministic Push-Down Automaton) är en automat med **obegränsat minne** i form av en stack.

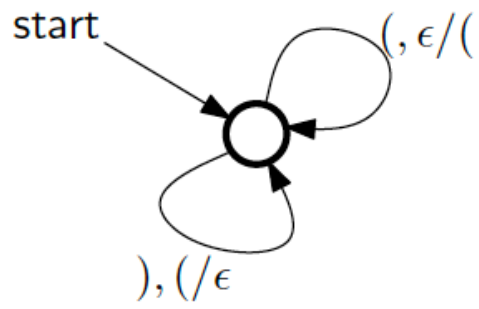
- Övergångar märkta med x, y/z: läs x, poppa y, pusha z



- Börja med att läsa a eller b och lägg på stacken
- Vid läsning av c, gå till tillstånd beroende på stacken
- Fortsätt läsa a (eller b) och poppa b (eller a) från stacken
- Stacken måste vara tom när indata är slut för att acceptera

Stackautomat för balanserade parenteser

- Grammatik för balanserade parenteser:
  $\text{Expr} \rightarrow \epsilon \mid (\text{Expr})\,\text{Expr}$
- DPDA:n som känner igen språket har ett enda tillstånd (förutom implicit feltillstånd)



Intuition:

- När vi ser en vänsterparentes, pusha den på stacken
- När vi ser en högerparentes, poppa en vänsterparentes

Från grammatiker till stackautomater

- Kan vi alltid konvertera en grammatik till en stackautomat?
- Nej, det går inte alltid!
- Exempel: mängden av palindrom över $\{a, b\}$
  - exempelsträng: $abba$
  - grammatik: $\mathrm{Palin} \to \epsilon \mid a \mid b \mid a\,\mathrm{Palin}\,a \mid b\,\mathrm{Palin}\,b$
  - det finns ingen DPDA som känner igen detta språk
- **Intuition**: om DPDA:n har läst precis hälften av indatan, så skulle den behöva matcha den återstående indata mot alla tecken den har sett. Men det finns inget sätt för automaten att veta när den läst precis hälften av indatan.
- Ett mer formellt bevis kan använda ett så kallat **pumpningslemma** för DPDA:er

1. Stackautomater
2. Klasser av språk och grammatiker
3. Parsergeneratorer
4. Sammanfattning av kursens syntaxdel
5. Exempeluppgifter

Klasser av språk och grammatiker

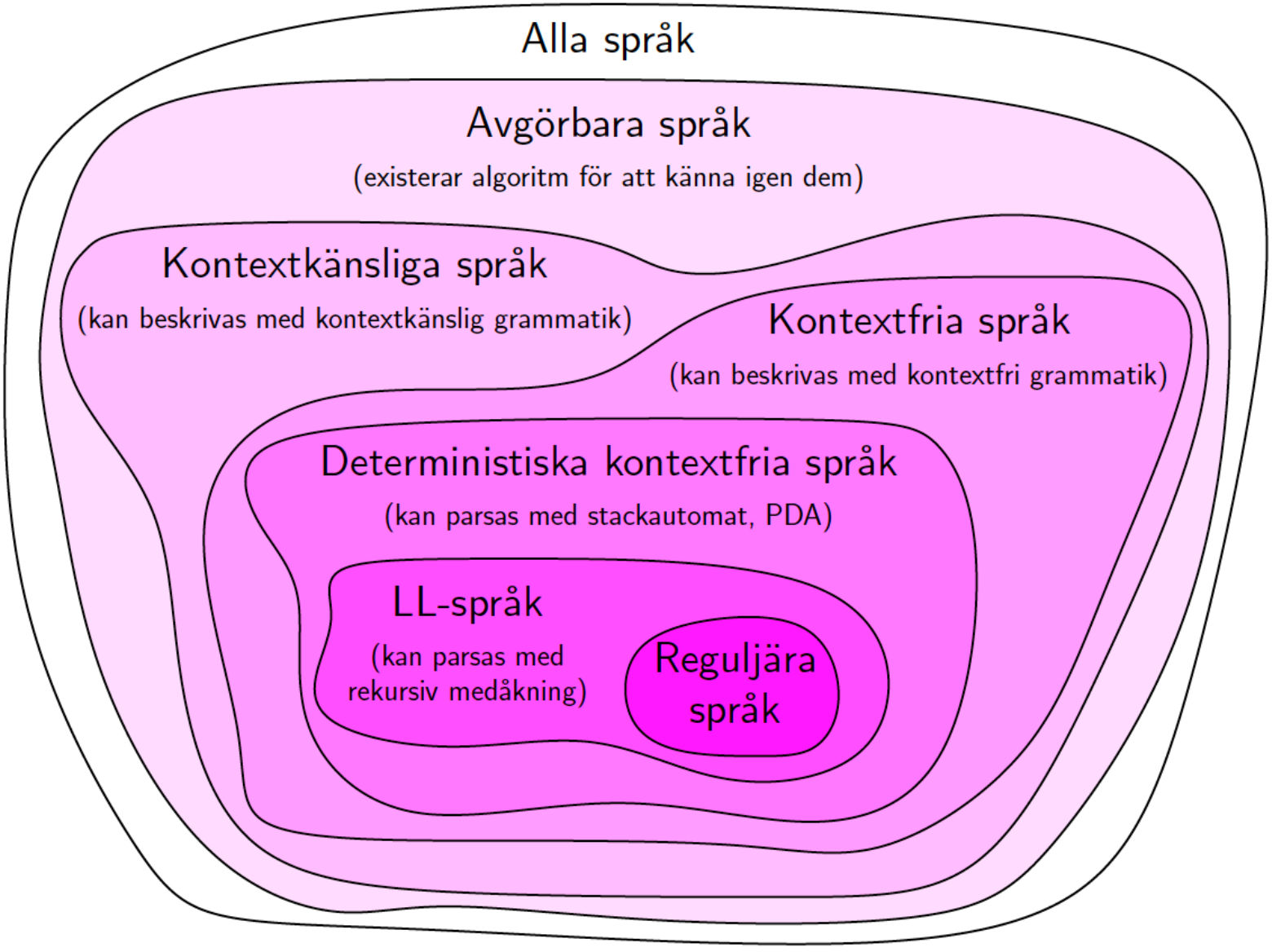
- Grammatikerna vi har använt i den här kursdelen är så kallade **kontextfria grammatiker**
  - det finns även *kontextkänsliga* grammatiker
- Vi har sett två metoder att parsa kontextfria grammatiker:
  - rekursiv medåkning: LL-grammatiker/språk
  - stackautomater: deterministiska kontextfria grammatiker/språk
- Ingen av metoderna hanterar *alla* kontextfria grammatiker, men klarar av många datalogiskt relevanta grammatiker

Översikt av språkklasser

Från mest begränsade klasser till mindre begränsade:

- **Reguljära språk**, t ex epostadresser
- **LL-språk**, t ex balanserade paranteser
- **Deterministiska kontextfria språk**, t ex $\{a^n b^m \mid n > m\}$
- **Kontextfria språk**, t ex palindrom över $\{a, b\}$
- Kontextkänsliga språk, t ex $\{a^n b^n c^n \mid n \geq 1\}$
- Avgörbara språk, t ex sanna satslogiska formler $P \rightarrow P$
- Alla språk, t ex Javaprogram som terminerar

Venndiagram för språkklasser



1 Stackautomater

2 Klasser av språk och grammatiker

3 Parsergeneratorer

4 Sammanfattning av kursens syntaxdel

5 Exempeluppgifter

Parsergeneratorer

- Att manuellt skriva en parser för ett helt stort programspråk är en resurskrävande och bräcklig process
- Att konstruera en parser från en grammatik är ofta mer mekaniskt och därmed lämpligt för en dator att göra
- En **parsergenerator** genererar en parser (inklusive lexikal analys) från en språkspecifikation

$$\text{språkspecifikation} \rightarrow \text{parser/lexergenerator} \rightarrow \text{parser}, \text{lexer}$$

Exempel på parsergeneratorer

- Lex/Flex och Yacc/Bison, Unixverktyg som ger kod i C/C++
  - hanterar de flesta kontextfria grammatiker: delmängd LALR
- JFlex och Java Cup, varianter av Flex och Yacc för Java
- Definite Clause Grammar-regler i Prolog, inbyggt i språket
  - hanterar alla kontextfria grammatiker
  - använder Prologs backtracking, som kan vara långsam
- ANTLR (ANother Tool for Language Recognition), genererar LL-parsers i flera språk (Java, C#, C++, JavaScript, ...)
- Menhir, genererar LR-parsers i OCaml

Grammatik för binära träd igen

```
<BinTree> ::= Leaf LPar Num RPar
  | Branch LPar <BinTree> Comma <BinTree> RPar
```

Slutsymboler:

- Leaf: `"leaf"`
- Branch: `"branch"`
- Num: `[0-9]+`
- LPar: `'('`, RPar: `')'`, Comma: `','`

Exempel: `branch(branch(leaf(17),leaf(42)),leaf(5))`

Resultat från lexikal analys:

```
Branch LPar Branch LPar Leaf LPar Num RPar Comma Leaf
LPar Num RPar RPar Comma Leaf LPar Num RPar RPar
```

Binärträd i Java Cup

```
import java_cup.runtime.*;

terminal BRANCH;
terminal LEAF;
terminal LPAREN;
terminal RPAREN;
terminal COMMA;
terminal Integer NUM;

non terminal ParseTree BinTree;

BinTree ::=
 LEAF LPAREN NUM:t RPAREN
   {: RESULT = new LeafNode(t); :}
| BRANCH LPAREN BinTree:left COMMA BinTree:right RPAREN
   {: RESULT = new BranchNode(left, right); :}
;
```

```
$ java -jar java-cup-11b.jar Parser.cup # erhåll parser.java, sym.java
```

Lexikal analys av binärträd i JFlex

```
import java.lang.System;
import java_cup.runtime.Symbol;

%%
%cup
%class Lexer

%%

branch { return new Symbol(sym.BRANCH); }
leaf { return new Symbol(sym.LEAF); }
"(" { return new Symbol(sym.LPAREN); }
")" { return new Symbol(sym.RPAREN); }
, { return new Symbol(sym.COMMA); }
[0-9]+ { return new Symbol(sym.NUM, new Integer(yytext())); }
[ \t\n] { }
```

```
$ jflex Lexer.lex # erhåll Lexer.java
```

Grammatik i Menhir

$$\text{Prop} \to \bot \mid \neg\text{Prop} \mid p \mid \text{Prop} \wedge \text{Prop} \mid \text{Prop} \vee \text{Prop} \mid \text{Prop} \Rightarrow \text{Prop} \mid (\text{Prop})$$

```
%token <string> IDENT
%token COMMA LPAREN RPAREN CONT NOT AND OR IMPLIES

%right IMPLIES
%left AND
%left OR
%right NOT

prop:
  CONT {} | NOT prop {} | IDENT {} | prop AND prop {}
| prop OR prop {} | prop IMPLIES prop {}
| LPAREN prop RPAREN {}
;
```

1 Stackautomater

2 Klasser av språk och grammatiker

3 Parsergeneratorer

4 Sammanfattning av kursens syntaxdel

5 Exempeluppgifter

Konkret och abstrakt syntax

- Vid parsning är vi primärt intresserade av **konkret** syntax
  - blanktecken kan spela roll
  - operatorprecedens viktig för betydelsen av strängar
  - avslutande hakparenteser eller semikolon spelar stor roll
- För många tillämpningar är endast **abstrakt** syntax viktig
  - översättningar mellan olika språk
  - transformationer av syntax till normalform
  - teoretisk analys av ett språk
- Denna kursdel fokuserar *mest* på konkret syntax
- Men bra att betänka abstrakta träd (räknarexemplet)

Sammanfattning

- Artificiella språk
  - språkklasser (t. ex. reguljära språk)
  - beskrivningar av språk (t. ex. automater, reguljära uttryck, grammatiker)
- Reguljära språk
  - ekvivalens mellan reguljära uttryck och finita automater
  - begränsningar: vissa språk är inte reguljära
- Grammatiker
  - kontextfria grammatiker
  - härledningar och parseträd
  - stackautomater (DPDA:er)
- Lexikal analys
- Parsning genom rekursiv medåkning

Förkortningar

- **DFA**: Deterministic Finite Automaton
  - den enklaste typen av finit automat
  - samma uttrycksfullhet som reguljära uttryck
- **DPDA**: Deterministic Push-Down Automaton
  - stackautomat, DFA men med obegränsad stack som minne
- **LL**: Left-to-right, Leftmost derivation
  - LL-parser: läs indata från vänster till höger och välj ickeslutsymbolen längst till vänster
  - LL-grammatik: grammatik som kan bli parsad av LL-parser
- **LR**: Left-to-right, Rightmost derivation
  - LR-parser: läs indata from vänster till höger och välj ickeslutsymbolen längst till höger
  - LR-grammatik: grammatik som kan bli parsad av LR-parser
- **LALR**: Look-Ahead LR
  - mindre kraftfull variant av LR som kan ge effektiv parsning

Typiska uppgiftstyper på mästarprov

- Tillämpa koncept från syntaxdelen på givna instanser, t ex språk och automater
- Beskriv relationerna mellan specifika tekniker och koncept
- Redogör för begränsingar av tekniker för givna exempel

Användbart material inför mästarprov

- Slides och kodexempel finns i kursöversikten i Canvas
- Inspelade föreläsningar från 2022 (inte likadana som i år)
- Delar av kursboken (IALC), enligt läsanvisningar
- Tidigare mästarprov med referenslösningar
  - referenslösningen ger endast ett rätt exempel eller centrala lösningssidén för varje uppgift
  - **mer utförliga** förklaringar krävs i nya mästarprovslösningar
- Kontrollskrivningar från tidigare kursomgångar (innan mästarprov infördes)

Allmänna tips inför mästarprov

- Det finns (ibland) flera sätt att svara rätt
  - detta tas hänsyn till under rättningen och presentationen
- Uppgifter ger ibland inte alla matematiska detaljer
  - var explicit med vilken tolkning av problemlydelsen som valts
  - vid stor osäkerhet, be om förtydligande
  - inga garantier om att få förtydliganden finns
- Ange källa för viktiga påståenden som används i lösningar
  - t ex från teorin om språk
- Wikipedia, Stack Overflow och liknande är inte giltiga källor
  - använd t ex kursboken (helst) eller slides
- Korrekta motiveringar till svar är inte nödvändigtvis långa
- Långa textsvar och handskriven text kan ofta öka risken för fel
- Åsidosätt tid att kontrollera lösningar
  - detaljer kan avgöra hela lösningen, t ex accepterande tillstånd
  - rättaren kan oftast inte skilja slarvfel från genuint fel
- Förklara egenkonstruerad notation och konventioner

Kurser som bygger vidare med syntax

- DD2350 Algoritmer, datastrukturer och komplexitet
  - mer om språkhierarkier
  - relation till komplexitet för datalogiska problem
- DD2373 Automatteori
  - djupstudie av automater
  - mer formellt om bevis och beräkningsbarhet
- DD2481 Principer för programspråk
  - formell semantik, typsystem
  - sundhet och verifiering av program
- ID2202 Kompilatorer och exekveringsmiljöer
  - tekniker för att implementera programspråk
- FDD3023 Interaktiv teorembevisning och programverifiering
  - doktorandkurs med verifierad matchning av reguljära uttryck

Klassisk drakbok som vidareläsning

Compilers: Principles, Techniques, and Tools

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman

https://suif.stanford.edu/dragonbook

1 Stackautomater

2 Klasser av språk och grammatiker

3 Parsergeneratorer

4 Sammanfattning av kursens syntaxdel

5 Exempeluppgifter

Talseriegrammatik igen

```
<Nums> ::= Num <NumsAux> | NoNums
<NumsAux> ::= <NumsAux> SemiColon Num | Epsilon
```

Startsymbol: `<Nums>`

- Num: `[0-9]+`
- SemiColon: `';'`
- NoNums: `'_'`
- Epsilon: tomma strängen

Kan grammatiken parsas med en LL-parser, och tillhör språket som definieras av grammatiken LL-språkklassen?

Talseriegrammatikanalys

```
<Nums> ::= Num <NumsAux> | NoNums
<NumsAux> ::= <NumsAux> SemiColon Num | Epsilon
```

- En LL-parser väljer alltid ickeslutsymbolen längst till vänster
- Detta gör att parsning enligt `NumsAux` aldrig terminerar, så grammatiken kan ej parsas med LL-parser
- En grammatik är **en möjlig** beskrivning av ett språk
- Det här språket kan beskrivas av ett reguljärt uttryck, så språket tillhör LL-språkklassen

```
NoNums|(Num(SemiColon Num)*)
```

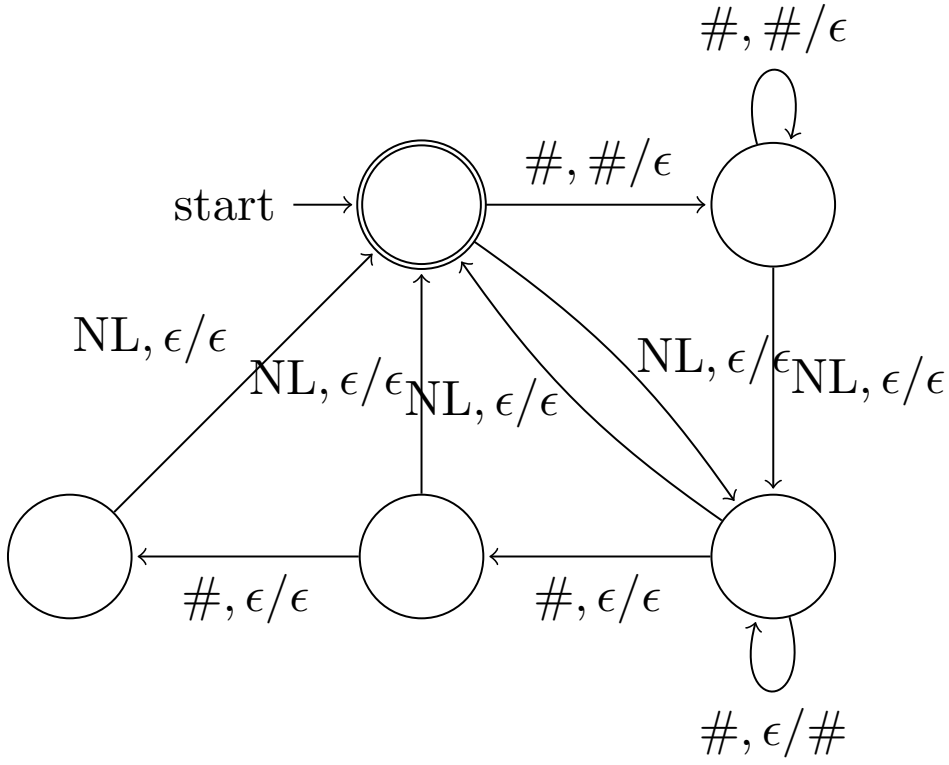
Nummerteckenspråk

- Betrakta ett språk som består av nummertecken (`'#'`, tokennamn Hash) separarade med nyradstecken (`'\n'`, tokennamn Newline).
- Varje rad av nummertecken ingår i **precis ett par av rader** räknat från början av ett ord i språket, så t ex ingår den första raden i ett par med den andra raden, och den femte raden i ett par med den sjätte raden, osv.
- För raderna rx och ry i varje par av rader gäller följande: om rx har n stycken förekomster av Hash så har ry antingen n stycken Hash eller n+1 stycken Hash eller n+2 stycken Hash.

Exempelsträng i språket:

```
#####
######
##
##
```

Nummerteckenspråk som DPDA



#####

######

##

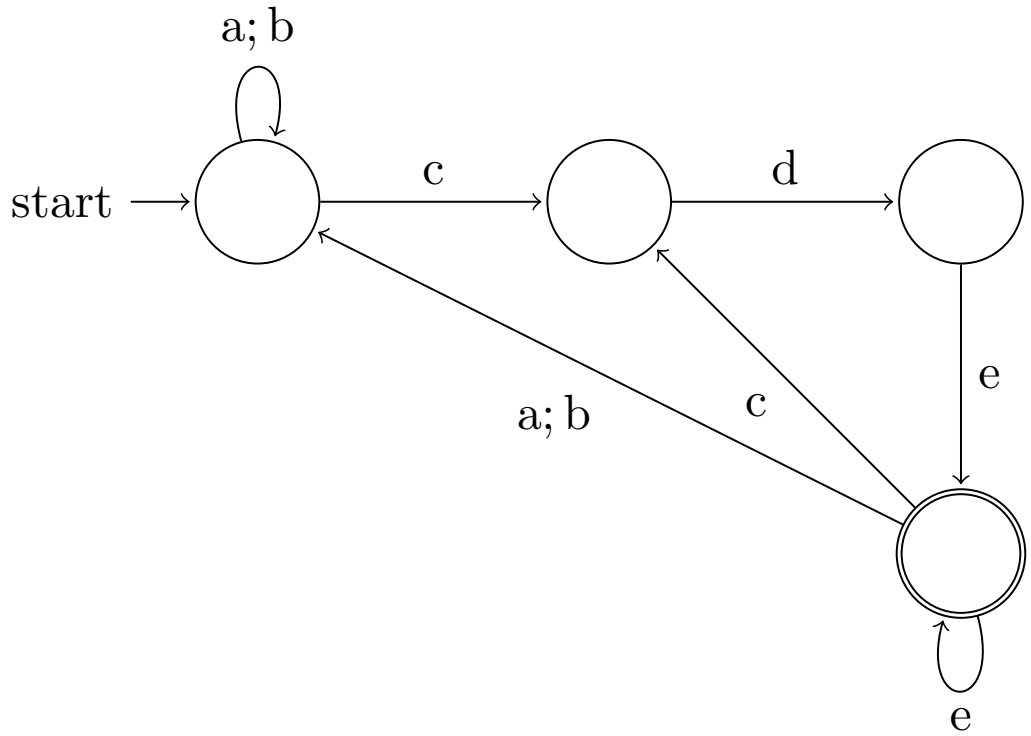
##

OK att ha rad som endast har nyradstecken!

Komplementspråk och DFA:er

$L = \{\ wcdv \mid w$ är tom eller innehåller bara a och b, och v består av ett eller flera e $\}$

DFA för $L+$:



Hur kan man bygga komplementautomaten? Inför explicit feltillstånd och växla accepterande och icke-accepterande tillstånd.

JFlex 1.5.1 och JavaCUP på modern Java

- `https://sourceforge.net/projects/jflex/files/jflex/1.5.1/jflex-1.5.1.tar.gz/download`
- `BinTreeWithJFlexAndCup.zip`

```
$ path/to/jflex-1.5.1/bin/jflex Lexer.lex

$ java -jar path/to/jflex-1.5.1/lib/java-cup-11a.jar \
  -parser Parser Parser.cup

$ javac -cp path/to/jflex-1.5.1/lib/java-cup-11a.jar:. *.java

$ java -cp path/to/jflex-1.5.1/lib/java-cup-11a.jar:. Main < test.in
[[3521;23];4145]
```