

DD1360 Programmeringsparadigm: Artificiella språk och syntaxanalys, föreläsning 2

Karl Palmskog

KTH

1 Mars, 2023

Förra föreläsningen

- Artificiella språk (i kontrast till *naturliga*)
 - alfabet Σ , alltid ändligt
 - språk är delmängder av Σ^*
- Reguljära uttryck
 - exempel: `letter (letter | digit)*`
 - språkigenkänning genom “matchning”
 - tolka uttrycket som mängd, kontrollera om ord är i mängden
- Ändliga automater
 - grafer med kanter som är märkta med tecken
 - språkigenkänning genom “maskinexekvering”
 - gå igenom indata, kontrollera om vi till slut hamnar i ett accepterande tillstånd

Dagens föreläsning

- ① Mer formellt om automater
- ② Reguljära språk
- ③ Grammatiker
- ④ Exempeluppgifter

① Mer formellt om automater

② Reguljära språk

③ Grammatiker

④ Exempeluppgifter

Mer formellt om automater

En ändlig automat är en tupel som består av:

- Q : ändlig mängd av tillstånd (hörn i grafen)
- Σ : alfabet (indatasymboler)
- $\delta \subseteq Q \times \Sigma \times Q$: transitioner (märkta kanter i grafen)
- $q_0 \in Q$: initialt tillstånd (startpil)
- $F \subseteq Q$: mängd av slutliga tillstånd (dubbla cirklar)

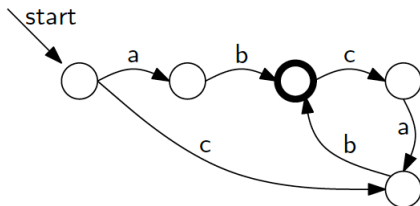
Typer av ändliga automater

- Deterministisk ändlig automat (DFA)
 - δ är en funktion
 - om $(q, a, q_1) \in \delta$ och $(q, a, q_2) \in \delta$ så $q_1 = q_2$
 - fokus i den här kursdelen
- Icke-deterministisk ändlig automat (NFA)
 - δ är en icke-funktionsrelation
 - behandlas inte i den här kursen, men viktig i andra kurser

Reguljära uttryck och ändliga automater

Om L är ett språk, så finns ett reguljärt uttryck som precis beskriver L **om och endast om** det finns en deterministisk ändlig automat som accepterar precis alla ord in L .

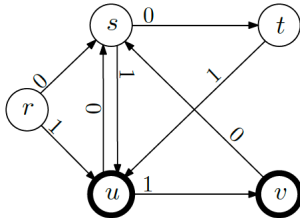
Exempel på automat och reguljärt uttryck



Ord vi ska fånga med ett reguljärt uttryck:

- börjar med ab eller cb (accepterande tillstånd) ...
- ... sedan valfritt antal cab (loop)
- översättning: $(ab|cb)(cab)^*$
- kompakt med syntaktiskt socker: $[ac]b(cab)^*$

Exempel på automat och reguljärt uttryck



Ord vi ska fånga med reguljärt uttryck:

- mer än två nollor/ettor i rad ..
- .. och slutar i en etta
- första översättningsförsök: $0?(10)^*1$
- fullständig översättning: $0?0?(1?10^?0)^*1?1$

1 Mer formellt om automater

2 Reguljära språk

3 Grammatiker

4 Exempeluppgifter

Språk och språkbeskrivningar

- Språk är matematiska mängder som inte har någon förvald representation i en formalism
- Språk kan dock indelas i abstrakta klasser
 - språket av Booleska värden: trivialt
 - språket av Mersenneprimtal i bas 10: definierbart men okänt
- Ett reguljärt uttryck eller en DFA är **en möjlig** entydig representation av ett språk
 - går också att använda t ex Javaprogram som korrekt svarar "ja" eller "nej" för givna indatasträngar
- Ibland ger existensen av en viss representation oss information om språket, men inte alltid

Reguljära språk

- Reguljära uttryck och ändliga automater har samma uttrycksfullhet (beskriver samma klass av språk)
- Språk som kan beskrivas av ett reguljärt uttryck eller en ändlig automat kallas för **reguljära språk**
- Exempel på reguljära språk:
 - namn på labbar i DD1360
 - alla binära strängar
 - giltiga identifierare i de flesta programspråk

Reguljära språks egenskaper

Antag att A och B är reguljära språk över Σ . Då gäller följande:

- $\bar{A} = \Sigma^* - A$ är ett reguljärt språk
 - bevisidé: byt accepterande och icke-accepterande tillstånd i DFA
- $A \cup B$ är ett reguljärt språk
 - bevisidé: beskriv med reguljära uttrycket $R_A | R_B$
- $A \cap B$ är ett reguljärt språk
 - bevisidé: använd att $A \cap B = \overline{\bar{A} \cup \bar{B}}$
- $A - B$ är ett reguljärt språk
 - bevisidé: använd att $A - B = \overline{\bar{A} \cap B}$

Reguljära uttrycks begränsningar

- Finns det språk som inte kan beskrivas av reguljära uttryck (eller ändlig automat)?
- Om sådana språk finns, hur kan vi bevisa att vi **inte kan** konstruera ett reguljärt uttryck för språket?
- Exempelspråk: $L = \{a^n b^n \mid n \geq 0\}$
- Intuition för att en finit automat inte existerar (ej bevis):
 - efter att ha läst k stycken a och sedan endast b , måste automaten komma ihåg att bara k stycken b får läsas
 - men automaten kan bara komma ihåg ett fixt antal tillstånd, medan n kan vara godtyckligt stort

Automat som påstås känna igen

$$L = \{a^n b^n \mid n \geq 0\}$$

- Antag att det finns en automat som känner igen L
- Låt automaten ha k tillstånd, $|Q| = k$
- Ge automaten a , aa , aaa , ..., och låt q_i vara tillståndet efter läsning av a^i
- Betrakta följande sekvens av tillstånd: $q_0, q_1, q_2, \dots, q_k$
- Sekvensen är $k + 1$ tillstånd lång, så det måste finnas ett repeterat tillstånd, dvs i så att $q_i = q_{i+p}$ för ett $p > 0$
- Så automaten borde acceptera $a^{i+p}b^{i+p}$
- Men då måste den också acceptera $a^i b^{i+p}$, eftersom den är i samma tillstånd efter att ha läst a^i som efter a^{i+p}
- Således accepterar den inte precis det givna språket, motsägelse
- En sådan automat kan därför inte existera

Begränsningar för reguljära språk

- Automaten har begränsat minne, kan inte “räkna”
- Automat i givet tillstånd beter sig alltid likadant

Exempel på “enkla” språk som ej är reguljära:

- Balanserade parentesuttryck (t.ex. $()((()))$)
- Palindrom (t.ex. “abba”, “nitalarbralatin”)
- Strängar av formen xx för någon sträng x (t.ex. “aa”, “hejhej”, “01110010111001”)

1 Mer formellt om automater

2 Reguljära språk

3 Grammatiker

4 Exempeluppgifter

Grammatiker

- För att beskriva språk som inte kan beskrivas med reguljära uttryck krävs mer uttrycksfulla formalismer
- Grammatiker är en sådan formalism
- Betrakta återigen språket $L = \{a^n b^n \mid n \geq 0\}$ som inte kan beskrivas av ett reguljärt uttryck
- Hur kan vi beskriva L matematiskt?

Språkdefinition med induktion

- Definiera $L = \{a^n b^n \mid n \geq 0\}$ induktivt:
 - basfall, $i = 0$: det tomma ordet, $a^0 b^0 = \epsilon$ är ett ord i L
 - induktivt fall: om $a^i b^i$ är ett ord i L , så är $a^i a b b^i = a a^i b b^i = a^{i+1} b^{i+1}$ ett ord i L
- Definiera språket av balanserade parentesuttryck induktivt:
 - basfall: tomma strängen ϵ är ett balanserat parentesuttryck
 - induktivt fall: om x och y är balanserade parentesuttryck så är $(x)y$ ett balanserat parentesuttryck

Grammatik för språket L

- Grammatiker låter oss beskriva språk med **rekursion**
- Grammatik för L : $W \rightarrow \epsilon \mid aWb$
- Grammatiken kan förstås på följande sätt:

Ett giltigt ord W är antingen det tomma ordet ϵ eller ett a följt av ett giltigt ord följt av ett b .

Beståndsdelarna för en grammatik

Grammatik G för språket L :

- **Slutsymbolerna** är element i det underliggande alfabetet
- **Ickeslutsymbolerna** representerar delar av ord
- **Produktionsregler** beskriver hur ord byggs upp:
 - Första regeln för L : $W \rightarrow \epsilon$
 - Andra regeln för L : $W \rightarrow aWb$

Härledningar (derivations)

$$S \rightarrow B \mid AA$$

$$A \rightarrow cA \mid dB$$

$$B \rightarrow aSa \mid \epsilon$$

- hur hittar man strängar som är i språket som har S som startsymbol?
- utgå från S och använd reglerna för att få giltiga strängar
 - $S \rightarrow B \rightarrow aSa \rightarrow aBa \rightarrow aa$
 - $S \rightarrow B \rightarrow aSa \rightarrow aAAa \rightarrow acAAa \rightarrow acdBAa \rightarrow acdAa \rightarrow acddBa \rightarrow acdda$
 - i varje steg ersätter vi precis en ickeslutsymbol
- Vi är vanligtvis intresserade av omvända fallet: givet strängen för ett program, följer strängen grammatiken för ett programspråk som Java, Scala eller Go?

Notation för grammatiker

- Matematisk notation:

$$S \rightarrow B \mid AA$$

$$A \rightarrow cA \mid dB$$

$$B \rightarrow aSa \mid \epsilon$$

- Backus-Naur-form (distinkt syntax för slutsymboler):

$$\langle S \rangle ::= \langle B \rangle \mid \langle A \rangle \langle A \rangle$$

$$\langle A \rangle ::= \text{"c"} \langle A \rangle \mid \text{"d"} \langle B \rangle$$

$$\langle B \rangle ::= \text{"a"} \langle S \rangle \text{"a"} \mid \epsilon$$

Grammatiker för programspråk

- Grammatiker är tillräckligt uttrycksfulla för att beskriva syntaxen i populära programspråk (Java, C#, Scala, ...)
- Specifikationer av programspråksyntax använder ofta grammatiker (ofta används domänspecifik notation)
- Sådana grammatiker används sedan grund för implementation av **parsers** för språken
- Dock har vissa programspråk ingen överenskommen grammatik (parserdefinierad syntax)

Exempel: uttryck i Scala

```
Expr      ::= (Bindings | ['implicit'] id | '_' ) '=>' Expr
            | Expr1
Expr1     ::= 'if' '(' Expr ')' {nl} Expr [[semi] 'else' Expr]
            | 'while' '(' Expr ')' {nl} Expr
            | 'try' Expr ['catch' Expr] ['finally' Expr]
            | 'do' Expr [semi] 'while' '(' Expr ')'
            | 'for' '(' Enumers ')' | '{' Enumers '}'
              {nl} ['yield'] Expr
            | 'throw' Expr
            | 'return' [Expr]
            | [SimpleExpr '.' ] id '=' Expr
            | SimpleExpr1 ArgumentExprs '=' Expr
            | PostfixExpr
            | PostfixExpr Ascription
            | PostfixExpr 'match' '{' CaseClauses '}'
PostfixExpr ::= InfixExpr [id [nl]]
InfixExpr   ::= PrefixExpr
            | InfixExpr id [nl] InfixExpr
PrefixExpr  ::= ['- ' | '+ ' | '~ ' | '!'] SimpleExpr
```

Exempel: grammatik för listor i Haskell

`[1, 2, 3]` `1:[2,3]` `[]` `1:2:3:[]`

Induktiv definition:

- Första basfallet: den tomma listan `[]` är en Haskell-lista
- Andra basfallet: om `L` är en kommaseparerad sekvens av element, så är `[L]` en Haskell-lista
- Induktivt fall: om `H` är ett listelement och `T` är en Haskell-lista, så är `H:T` en Haskell-lista

BNF:

`<List> ::=`

`"[]"` | `"[" <ListElems> "]"` | `<ListElem> ":" <List>`

Haskell-listor, fortsättning

```
<List> ::=  
  "[]" | "[" <ListElems> "]" | <ListElem> ":" <List>  
<ListElems> ::= <ListElem> | <ListElem> "," <ListElems>
```

Hur ska <ListElem> definieras?

- i riktiga Haskell kan element i listor vara godtyckliga Haskelluttryck
- för att göra exemplet enklare: begränsa element till 0 och 1

```
<ListElem> ::= "0" | "1"
```

① Mer formellt om automater

② Reguljära språk

③ Grammatiker

④ Exempeluppgifter

DFA-konstruktion

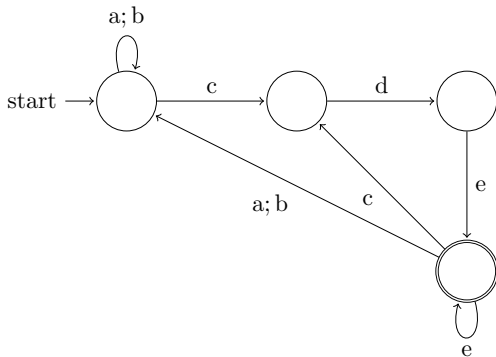
På matematisk nivå definierar vi språket L med hjälp av ordvariablerna w och v och tecknen a , b , c , d och e :

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

Betrakta språket L^+ som innehåller alla ord i L upprepade en eller flera gånger. Definiera grafiskt en DFA som precis beskriver L^+ .

DFA-konstruktion, fortsättning

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



Talseriegrammatik

Betrakta följande grammatik:

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

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

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

Vad kan vi säga om språket som definieras av grammatiken?

Talseriegrammatik, fortsättning

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

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

Exempelsträngar genererade av grammatiken:

- `-`
- `1;2;3`
- `52323`

Reguljärt uttryck som beskriver språket:

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

Språket är reguljärt.