DD1360 Programmeringsparadigm: Artificiella språk och syntaxanalys, föreläsning 1

Karl Palmskog

KTH

Måndag 27 Februari, 2023

1. Introduktion
2. Artificiella språk
3. Reguljära uttryck
4. Ändliga automater

Om mig

- Tekn. dr., KTH, 2014; postdoktor/forskare 2015-2021
- Adjunkt, KTH, 2021-
- Forskar inom:
  - programverifiering och bevisteknik
  - distribuerade system (t ex blockkedjor)
- Undervisar även inom:
  - parallella och distribuerade system
  - programvarukonstruktion
- Kontaktinformation finns i kurskatalogen, men använd gärna forumet på Canvas för generella frågor

Syntaxdelen i kursen

- Teoretiskt verktygslåda för strängmatchning
  - Reguljära uttryck och reguljära språk
  - Grammatiker
  - Språkklasser, leder till algoritmteori
- Praktisk tillämpning av verktygslådan
  - Lexikal analys för programspråk
  - Syntaktisk analys för programspråk
  - Leder till konstruktion av parsers

Kursbok för syntaxdelen

- **Introduction to Automata Theory, Languages, and Computation**
  - Hopcroft, Motwani, Ullman
  - 3rd Edition, Pearson, 2006
  - http://www-db.stanford.edu/~ullman/ialc.html
- Slides är fristående från boken, men följer bokens ansatser
- Syntaxordlista och läsanvisningar för boken finns på Canvas
  - bara vissa delar av boken ingår (t ex inte algoritmer och bevis)
  - vissa räkneuppgifter i boken ej relevanta för kursen ...
  - ... men kan hjälpa förståelse
- Andra liknande böcker går att använda
  - ex **Compilers: Principles, Techniques, and Tools**, 2nd Ed
  - tänk på avvikelser i terminologi och definitioner

Mästarprovet i syntax

- Mästarprovet följer andra mästarprov i upplägg
  - skriftliga lösningar som redovisas muntligt
  - ett eller fler allvarliga fel ger underkänt
- Kursboken användbar som uppslagsverk/referens
- Tänk på att dubbelkolla svarsförsök
- Vissa exempeluppgifter och referenslösningar gås igenom under föreläsningarna

Språk

- Ett **ord** är en ändlig sekvens av element från ändlig mängd Σ
- Vi kallar oftast Σ för ett **alfabet**
- Vi tillåter det **tomma ordet** ϵ
- Ett **språk** är en delmängd av Σ^*
- uv är sammanslagningen (konkateneringen) av orden u och v

$$L_1 L_2 = \{u_1 u_2 \,|\, u_1 \in L_1, u_2 \in L_2\}$$
$$L^0 = \{\epsilon\},\ L^{k+1} = L L^k$$
$$L^* = \bigcup_k L^k \text{ (Kleenestjärna)}$$

- Kleenestjärna: “det som stod framför, valfritt antal gånger”

Exempel på språk

$$\Sigma = \{a, b\}$$
$$\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, ...\}$$

Exempel på två språk (delmängder av Σ^*):

$$L_1 = \{a, bb, ab\} \text{ (ändligt språk, tre ord)}$$
$$L_2 = \{ab, abab, ababab, ...\}$$
$$= \{(ab)^n | n > 0\} \text{ (oändligt språk)}$$

Exempel på språk, fortsättning

- 8-bitars binära tal: $\{0,1\}^8$
- Satisfierbara Booleska formler
- Binära strängar med udda paritet:
  $\{w \in \{0,1\}^* \mid \text{udda antal ettor i } w\}$
- Mersenne-primtal i bas 10
- Syntaktiskt korrekta Javaprogram som terminerar

Exempel på operationer på språk

$$
\begin{aligned}
L &= \{a, ab\} \\
LL &= \{aa, aab, aba, abab\} \\
L^* &= \{\epsilon, a, ab, aa, aab, aba, abab, aaa, ...\} \\
&= \{u \,|\, \text{omedelbart före varje } b \text{ i } u \text{ finns ett } a\}
\end{aligned}
$$

Hur bevisa att $bb \notin L^*$?

Språk och problem

- Vi pratar ofta om (datalogiska) problem att avgöra om ett ord är i ett språk L
  - givet $w \in \Sigma^*$, avgör om w är i L eller inte
  - t ex för binärt tal w, avgör om w är primt eller inte
- Vi säger också ekivalent att vi letar efter metoder för att **känna igen** ord i L
  - en korrekt metod måste förkasta alla ord som **inte är** i L
  - en korrekt metod måste godkänna alla ord som **är** i L
- Att definiera ett problem kan vara, men är inte alltid, en implicit uppmaning att hitta en metod att känna igen språket

Artificiella språk och kompilatorer

- **Lexikal analys** i en kompilator känner igen beståndsdelar (*tokens*) i ett programspråk:
  - nyckelord som `class`, `while`, `if`
  - variabelnamn, parameternamn, metodnamn, klassnamn, etc.
  - operatorer och avgränsare: `+`, `-`, `*`, `/`, `%`, `;`, etc.
  - alfabet Σ i lexikal analys: tecken, t ex ASCII, UTF-8
- **Syntaktisk analys** i en kompilator känner igen syntaktiska konstruktioner i ett programspråk:
  - uttryck som `34 + x`
  - variabeldeklarationer som `int x = 3`
  - alfabet Σ i syntaktisk analys: tokens från lexikal analys

Reguljära uttryck

- Strukturerat sätt att beskriva språk entydigt
  - ofta språk med oändligt antal ord
- Reguljära uttryck byggs med:
  - tomma språket $\emptyset$
  - $\{\epsilon\}$, skrivs oftast ϵ
  - $\{a\}$, där $a \in \Sigma$, skrivs oftast a
  - union, skrivs oftast | (ibland +)
  - konkatenering, skrivs oftast som multiplikation (punkt/tom)
  - repetition med Kleenestjärna

Reguljära uttryck: exempel 1

- Namn på labbar i DD1360:
  - `F1`, `F2`, `F3`, `S1`, `S2`, `S3`, `X1`, `X2`
- Vi kan beskriva labbarna med följande reguljära uttryck:
  - `F1 | F2 | F3 | S1 | S2 | S3 | X1 | X2`
- Förklaring:
  - `F` står för språket $\{F\}$, där $F \in \Sigma$
  - `F1` står för språket $\{F1\}$, där $F \in \Sigma$ och $1 \in \Sigma$
  - `F1 | F2` står för språket $\{F1, F2\}$, där $F, 1, 2 \in \Sigma$
  - etc.

Reguljära uttryck: exempel 1, fortsättning

- Namn på labbar i DD1360:
  - `F1`, `F2`, `F3`, `S1`, `S2`, `S3`, `X1`, `X2`
- Namnen följer ett **mönster**:
  - börjar med `F`, `S` eller `X` följt av `1`, eller
  - börjar med `F`, `S` eller `X` följt av 2, eller
  - börjar med `F` eller `S` följt av 3
- Detta mönster kan beskrivas med följande reguljära uttryck:
  - `(F | S | X)1 | (F | S | X)2 | (F | S)3`
- Kan vi använda andra alfabet än ASCII?

Reguljära uttryck: exempel 2

- Alla binära strängar:
  - `""`, `"0"`, `"1"`, `"00"`, `"01"`, `"10"`, `"11"`, `"000"`, ...
- Fundamental skillnad från tidigare exempel?
  - det finns ett **obegränsat** antal binära strängar
  - vi kan inte lista dem alla med `|`
- Använd Kleenestjärna: `(0 | 1)*`

Syntaktiskt socker

- parenteser för gruppering
- `[a-z]` = `a | b | ... | z` (använder ASCII-ordning)
- `e?` (en eller ingen gång)
- `e+` (en eller flera gånger)
- `!e` (komplement/matcha inte, ibland `[^a-z]`)
- `e1 & e2` = `!(!e1|!e2)` (snitt, matcha båda)

Fler reguljära uttryck

- Decimaltal, “`digit`”: `0 | 1 | 2 | ... | 9`
- Heltalskonstanter: `digit digit*`
- Tecken i alfabetet, “`letter`”: `[a-z] | [A-Z]`
- Identifierare: `letter (letter | digit)*`
  - variabelnamn i programspråk
  - metodnamn
  - etc.

Reguljära uttryck: exempel 3

- Förenklade domännamn består av minst två punktseparerade icke-tomma strängar med bara små bokstäver och siffror
- Reguljärt uttryck för "icke-tom sträng med bara små bokstäver och siffror": `[a-z0-9]+`
- Försök till reguljärt uttryck för domännamn: `[a-z0-9]+(\.[a-z0-9]+)*`
- Problem(?): domännamn med bara en del (utan punkt)
- Slutligt reguljärt uttryck: `[a-z0-9]+(\.[a-z0-9]+)+`

Reguljärt uttryck för epostadress?

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:
\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t]
)+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:
[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:
\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*
(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(
?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[
\t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?
[ \t])*(?:[^()<>@,;:\\".\[\] \000-\0
31]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@
,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\
```

Se även RFC 5321 och 5322.

Reguljära uttryck i praktiken: regex

- Reguljära uttryck används implicit för textbehandling
  - sök och ersätt i IDE:er
  - del av informationssökningsgränssnitt
- Många verktyg/programspråk implementerar egna reguljära uttryck (regex)
  - många olika inkompatibla variationer i syntax
  - utvidgningar (backtracking, negativ lookahead, …)
  - läs dokumentationen innan du matchar
  - se upp för regex som inte är reguljära uttryck

Reguljära uttryck i Unixverktyg

- `grep '<regex>' <file>`
  - matar ut rader i `<file>` där text som matchar `<regex>` finns
- `sed 's/<regex>/<replacement>/g' < <file>`
  - matar ut innehållet i `<file>` med matchningar av `<regex>` ersatta med `<replacement>`

Exempel med grep och sed

```
$ grep '..ing' wikipedia
grep is a command-line utility for searching plain-text
has the same effect: doing a global search with the
and printing all matching lines.
$ sed 's/Bell/Whistle/g' < wikipedia > wikipedia_funny
```

Reguljära uttryck i Java

- Paketet `java.util.regex` har klasser för att matcha Javas strängar mot mönster uttryckta som reguljära uttryck
- En instans av klassen Pattern representerar ett reguljärt uttryck som uttryckts i form av en sträng

```
import java.util.regex.*;

Pattern p = Pattern.compile("cat");
Matcher m = p.matcher("one cat, two cats in the yard");
String s = m.replaceAll("dog");
// --> s = "one dog, two dogs in the yard"
```

Exempeluppgift om reguljära uttryck

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

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

Definiera ett reguljärt uttryck som precis beskriver L.

Rimliga tolkningar och svar

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

- Tolka "tom eller innehåller bara a och b" som: (ab|ba|ϵ)
  - kan även skrivas (ab|ba)?
  - vi får reguljära uttrycket (ab|ba)?cd(e)+
  - (e)+ kan även skrivas e(e)*
- Tolka "tom eller innehåller bara a och b" som: (a|b)*
  - vi får reguljära uttrycket (a|b)*cd(e)+
- Inget tolkningsutrymme för entydig ordning mellan a och b
  - fel att tolka "tom eller innehåller bara a och b" som: (ab)?
  - fel att tolka "tom eller innehåller bara a och b" som: (ba)?

Vad är en ändlig automat?

En ändlig automat består av:

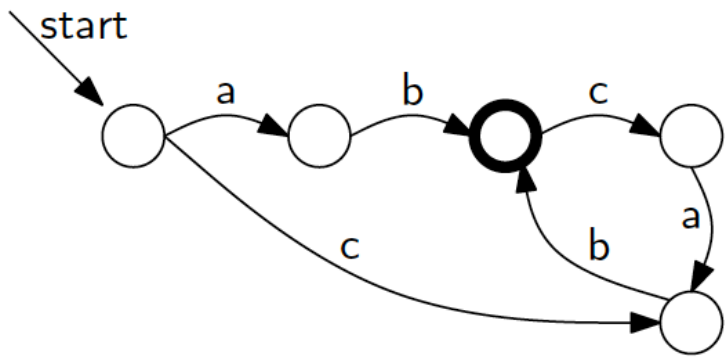
- En ändlig mängd av tillstånd Q
- Ett ändligt alfabet med indatasymboler Σ
- En mängd av övergångar δ mellan tillstånd märkta med element i Σ
- Ett initialt tillstånd $q \in Q$
- En mängd av accepterande tillstånd F (ibland *slutliga* tillstånd)

Ändliga automater och determinism

- Om alla övergångar ut från ett tillstånd har olika märkning är den ändliga automaten **deterministisk**
  - DFA, Deterministic Finite Automaton
- Om det finns flera övergångar ut från ett tillstånd med samma märkning är automaten **indeterministisk**
  - NFA, Nondeterministic Finite Automaton
- Vi fokuserar på **deterministiska** ändliga automater här (mängden av övergångar är en **funktion**)

Ändliga automater och matchning

- Ändliga automater kan ses som abstrakta maskiner för matchning av ord/strängar
- Tillstånd är kanter i maskinens graf
- Övergångar är hörn i maskinens graf
- Tecken från indatasträng matchas mot märkningar för övergångar och leder till tillståndsbyte
- Om vi slutar i ett accepterande tillstånd accepteras strängen
- Alternativ till reguljära uttryck för språkigenkänning
- Mer om automater i nästa föreläsning



Bonus: exempeluppgift reguljära uttryck

På matematisk nivå definierar vi språket L med hjälp av ordvariabeln w och tecknen a, b, och c:

$L = \{\ w\text{c} \mid w$ är tom eller består av ett jämnt positivt antal a följt av ett jämnt positivt antal b $\}$

Rimliga tolkningar och svar

$L = \{\ w\text{c} \mid w$ är tom eller består av ett jämnt positivt antal a följt av ett jämnt positivt antal b $\}$

- Föreslaget reguljärt uttryck: ((aa)+(bb)+)?c
- Orimligt med (a)+ eller (b)+, då är det inte nödvändigtvis jämnt antal
- Orimligt med (aa)* eller (bb)*, då är det inte positivt antal
- Glöm inte c på slutet