DD1361
Programmeringsparadigm

Formella Språk & Syntaxanalys
Föreläsning 4

Per Austrin

2015-11-20

Idag

Rekursiv medåkning, fortsättning

Olika klasser av språk och grammatiker

Parsergeneratorer

Sammanfattning / Inför KS

Idag

Rekursiv medåkning, fortsättning

Olika klasser av språk och grammatiker

Parsergeneratorer

Sammanfattning / Inför KS

Begränsningar för rekursiv medåkning

Från förra föreläsningen:
Den här grammatiken för aritmetiska uttryck kan inte parsas med rekursiv medåkning

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term} \mid \\
&\quad\ \text{Expr} + \text{Term} \mid \\
&\quad\ \text{Expr} - \text{Term} \\
\text{Term} &\rightarrow \text{Factor} \mid \\
&\quad\ \text{Term} * \text{Factor} \mid \\
&\quad\ \text{Term} / \text{Factor} \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

Begränsningar för rekursiv medåkning

Från förra föreläsningen:
Den här grammatiken för aritmetiska uttryck kan inte parsas med rekursiv medåkning

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term} \mid \\
&\quad\ \text{Expr} + \text{Term} \mid \\
&\quad\ \text{Expr} - \text{Term} \\
\text{Term} &\rightarrow \text{Factor} \mid \\
&\quad\ \text{Term} * \text{Factor} \mid \\
&\quad\ \text{Term} / \text{Factor} \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

Det största problemet är *vänster-rekursion*:
ett Expr kan börja med ett Expr

Grammatik-notation för listor

Vi gör ytterligare ett försök, och formulerar en ny grammatik:

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term}\{\text{PM Term}\} \\
\text{Term} &\rightarrow \text{Factor}\{\text{TD Factor}\} \\
\text{PM} &\rightarrow + \mid - \\
\text{TD} &\rightarrow * \mid / \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term} \mid \\
&\quad\ \text{Expr} + \text{Term} \mid \\
&\quad\ \text{Expr} - \text{Term} \\
\text{Term} &\rightarrow \text{Factor} \mid \\
&\quad\ \text{Term} * \text{Factor} \mid \\
&\quad\ \text{Term}/\text{Factor} \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

Grammatik-notation för listor

Vi gör ytterligare ett försök, och formulerar en ny grammatik:

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term}\{\text{PM Term}\} \\
\text{Term} &\rightarrow \text{Factor}\{\text{TD Factor}\} \\
\text{PM} &\rightarrow + \mid - \\
\text{TD} &\rightarrow * \mid / \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term} \mid \\
&\quad\ \text{Expr} + \text{Term} \mid \\
&\quad\ \text{Expr} - \text{Term} \\
\text{Term} &\rightarrow \text{Factor} \mid \\
&\quad\ \text{Term} * \text{Factor} \mid \\
&\quad\ \text{Term}/\text{Factor} \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

Notationen $\{$ole dole doff$\}$ betyder
"ole dole doff upprepat 0 eller fler gånger"

Grammatik-notation för listor

Vi gör ytterligare ett försök, och formulerar en ny grammatik:

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term}\{\text{PM Term}\} \\
\text{Term} &\rightarrow \text{Factor}\{\text{TD Factor}\} \\
\text{PM} &\rightarrow + \mid - \\
\text{TD} &\rightarrow * \mid / \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term} \mid \\
&\quad\ \text{Expr} + \text{Term} \mid \\
&\quad\ \text{Expr} - \text{Term} \\
\text{Term} &\rightarrow \text{Factor} \mid \\
&\quad\ \text{Term} * \text{Factor} \mid \\
&\quad\ \text{Term}/\text{Factor} \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

Betyder alltså:

$$\text{Expr} \rightarrow \text{Term} \mid \text{Term PM Term} \mid \text{Term PM Term PM Term} \mid \text{Term}$$

Rekursiv medåkning för aritmetiska uttryck

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term}\{\text{PM Term}\} \\
\text{Term} &\rightarrow \text{Factor}\{\text{TD Factor}\} \\
\text{PM} &\rightarrow + \mid - \\
\text{TD} &\rightarrow * \mid / \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

Rekursiv medåkning för aritmetiska uttryck

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Term}\{\text{PM Term}\} \\
\text{Term} &\rightarrow \text{Factor}\{\text{TD Factor}\} \\
\text{PM} &\rightarrow + \mid - \\
\text{TD} &\rightarrow * \mid / \\
\text{Factor} &\rightarrow \text{Num} \mid (\text{Expr})
\end{aligned}
$$

Java-kod för att parsa ett Expr:

```
ParseTree Expr() {
   ParseTree result = Term();
   while (peekToken() == PLUS or
          peekToken() == MINUS) {
      Token operator = nextToken();
      ParseTree next = Term();
      result = new BinaryOperation(operator, result, next);
   }
   return result;
}
```

Rekursiv medåkning för aritmetiska uttryck

```
ParseTree Expr() {
   ParseTree result = Term();
   while (peekToken() == PLUS or
          peekToken() == MINUS) {
      Token op = nextToken();
      ParseTree next = Term();
      result = new BinOp(op,
                         result,
                         next);
   }
   return result;
}
```

```
ParseTree Term() {
   ParseTree result = Factor();
   while (peekToken() == TIMES or
          peekToken() == DIVIDE) {
      Token op = nextToken();
      ParseTree next = Factor();
      result = new BinOp(op,
                         result,
                         next);
   }
   return result;
}
```

```
ParseTree Factor() {
   Token t = nextToken();
   if (t == NUM) return new Number(t.value);
   else if (t == LPAREN) {
      ParseTree result = Expr();
      if (nextToken() != RPAREN)
         throw new SyntaxError();
      return result;
   }
   throw new SyntaxError();
}
```

Rekursiv medåkning för aritmetiska uttryck

```
ParseTree Expr() {
   ParseTree result = Term();
   while (peekToken() == PLUS or
          peekToken() == MINUS) {
      Token op = nextToken();
      ParseTree next = Term();
      result = new BinOp(op,
                         result,
                         next);
   }
   return result;
}
```

```
ParseTree Term() {
   ParseTree result = Factor();
   while (peekToken() == TIMES or
          peekToken() == DIVIDE) {
      Token op = nextToken();
      ParseTree next = Factor();
      result = new BinOp(op,
                         result,
                         next);
   }
   return result;
}
```

```
ParseTree Factor() {
   Token t = nextToken();
   if (t == NUM) return new Number(t.value);
   else if (t == LPAREN) {
      ParseTree result = Expr();
      if (nextToken() != RPAREN)
         throw new SyntaxError();
      return result;
   }
   throw new SyntaxError();
}
```

Exempel på tavlan:

5−2−3

(3+1)∗7

Rekursiv medåkning för aritmetiska uttryck

```
ParseTree Expr() {
   ParseTree result = Term();
   while (peekToken() == PLUS or
          peekToken() == MINUS) {
      Token op = nextToken();
      ParseTree next = Term();
      result = new BinOp(op,
                         result,
                         next);
   }
   return result;
}
```

```
ParseTree Term() {
   ParseTree result = Factor();
   while (peekToken() == TIMES or
          peekToken() == DIVIDE) {
      Token op = nextToken();
      ParseTree next = Factor();
      result = new BinOp(op,
                         result,
                         next);
   }
   return result;
}
```

```
ParseTree Factor() {
   Token t = nextToken();
   if (t == NUM) return new Number(t.value);
   else if (t == LPAREN) {
      ParseTree result = Expr();
      if (nextToken() != RPAREN)
         throw new SyntaxError();
      return result;
   }
   throw new SyntaxError();
}
```

Exempel på tavlan:

5−2−3

(3+1)∗7

(Fullständig Java-implementation upplagd på kurshemsidan)

Idag

Rekursiv medåkning, fortsättning

Olika klasser av språk och grammatiker

Parsergeneratorer

Sammanfattning / Inför KS

Olika kraftfulla grammatiker

De grammatiker vi använt oss av är vad som kallas för *kontextfria grammatiker*
(Det finns mer generella grammatiker som kallas för *kontextkänsliga*)

Olika kraftfulla grammatiker

De grammatiker vi använt oss av är vad som kallas för *kontextfria grammatiker*
(Det finns mer generella grammatiker som kallas för *kontextkänsliga*)

Vi har sett två verktyg för att parsa kontextfria grammatiker:

- Stackautomater (föreläsning 2)

- Rekursiv medåkning (föreläsning 3 och nyss)

Olika kraftfulla grammatiker

De grammatiker vi använt oss av är vad som kallas för *kontextfria grammatiker*
(Det finns mer generella grammatiker som kallas för *kontextkänsliga*)

Vi har sett två verktyg för att parsa kontextfria grammatiker:

- Stackautomater (föreläsning 2)

- Rekursiv medåkning (föreläsning 3 och nyss)

Inget av dem är tillräckligt kraftfullt för att kunna hantera alla kontextfria grammatiker, men de är tillräckliga för att hantera de flesta språk man faktiskt vill skriva en parser för.

Olika kraftfulla grammatiker

De grammatiker vi använt oss av är vad som kallas för
kontextfria grammatiker
(Det finns mer generella grammatiker som kallas för *kontextkänsliga*)

Vi har sett två verktyg för att parsa kontextfria grammatiker:

- Stackautomater (föreläsning 2)
  Deterministiska kontextfria grammatiker
  Deterministiska kontextfria språk
- Rekursiv medåkning (föreläsning 3 och nyss)
  LL-grammatiker
  LL-språk

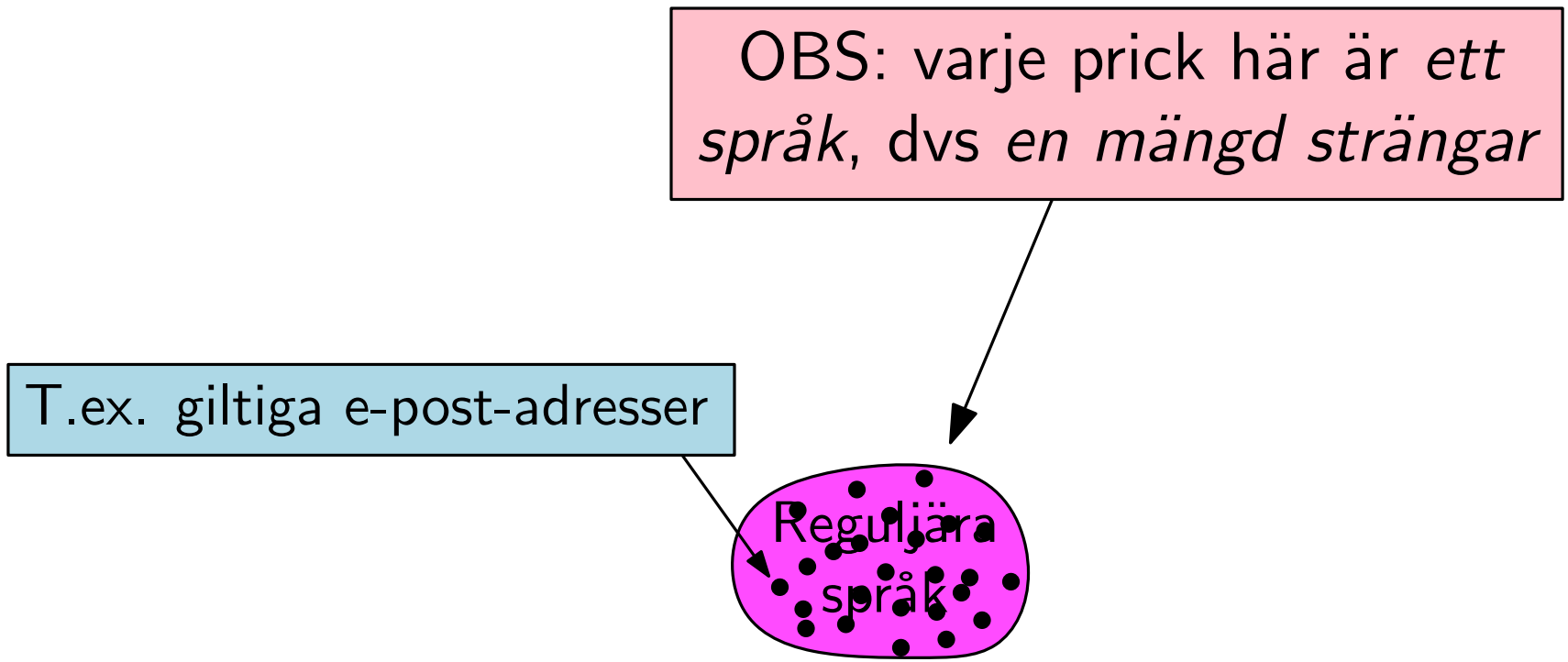
Inget av dem är tillräckligt kraftfullt för att kunna hantera alla kontextfria grammatiker, men de är tillräckliga för att hantera de flesta språk man faktiskt vill skriva en parser för.

Perspektiv

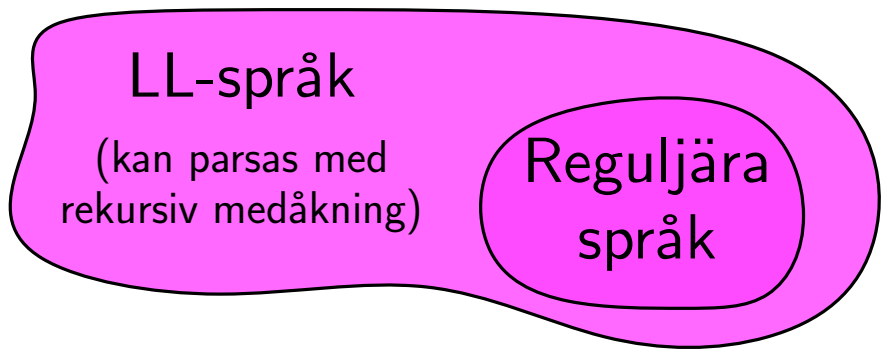
Perspektiv

T.ex. giltiga e-post-adresser

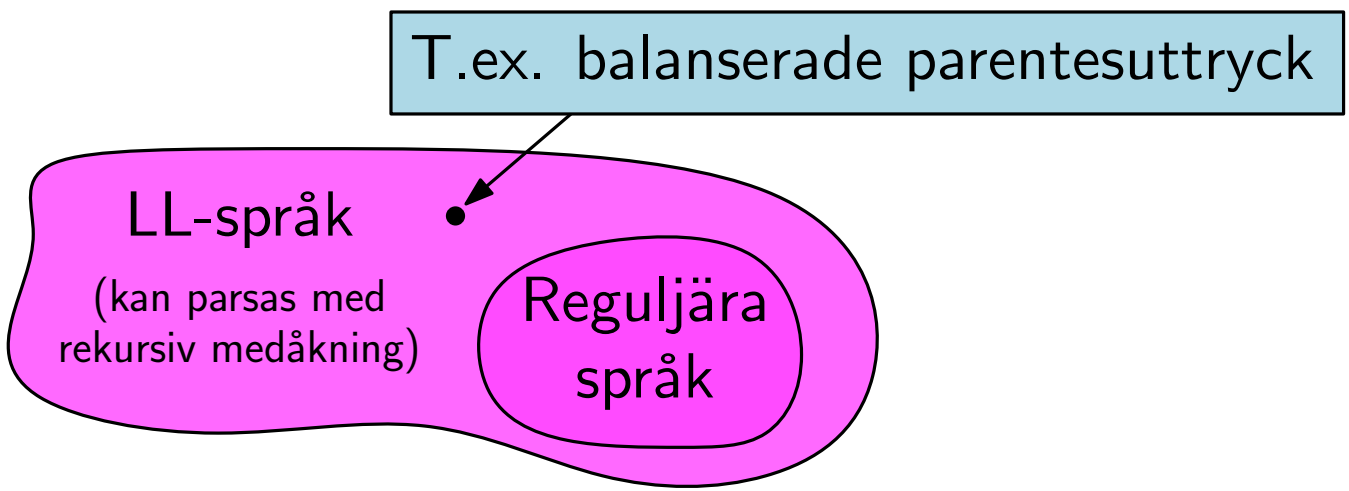
Perspektiv


OBS: varje prick här är *ett språk*, dvs *en mängd strängar*
T.ex. giltiga e-post-adresser
Reguljära språk

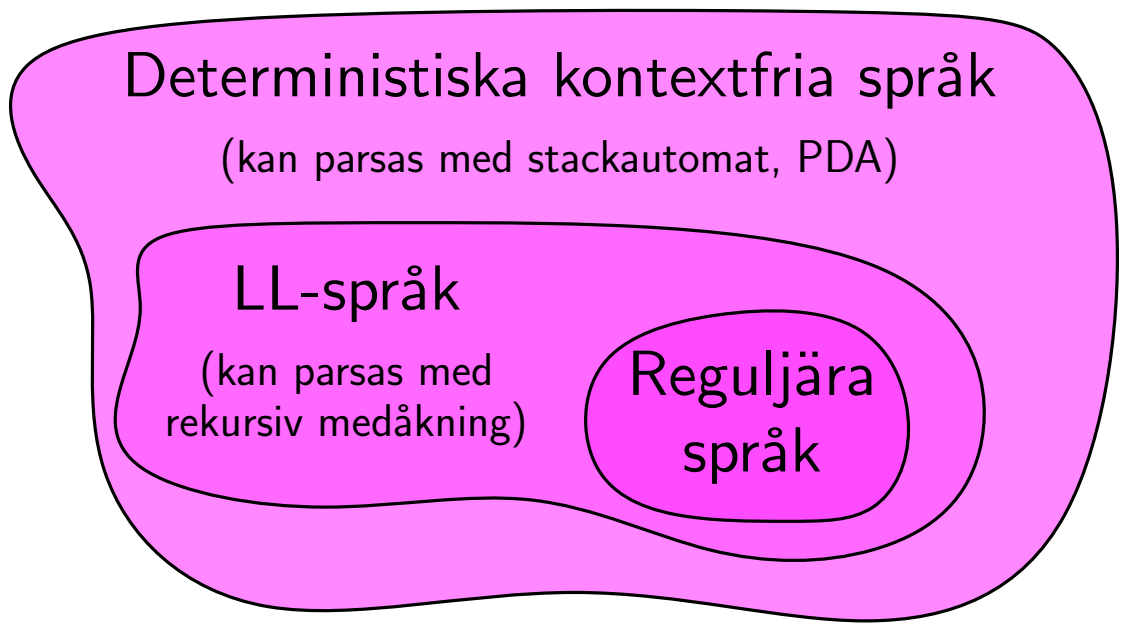
Perspektiv


LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk

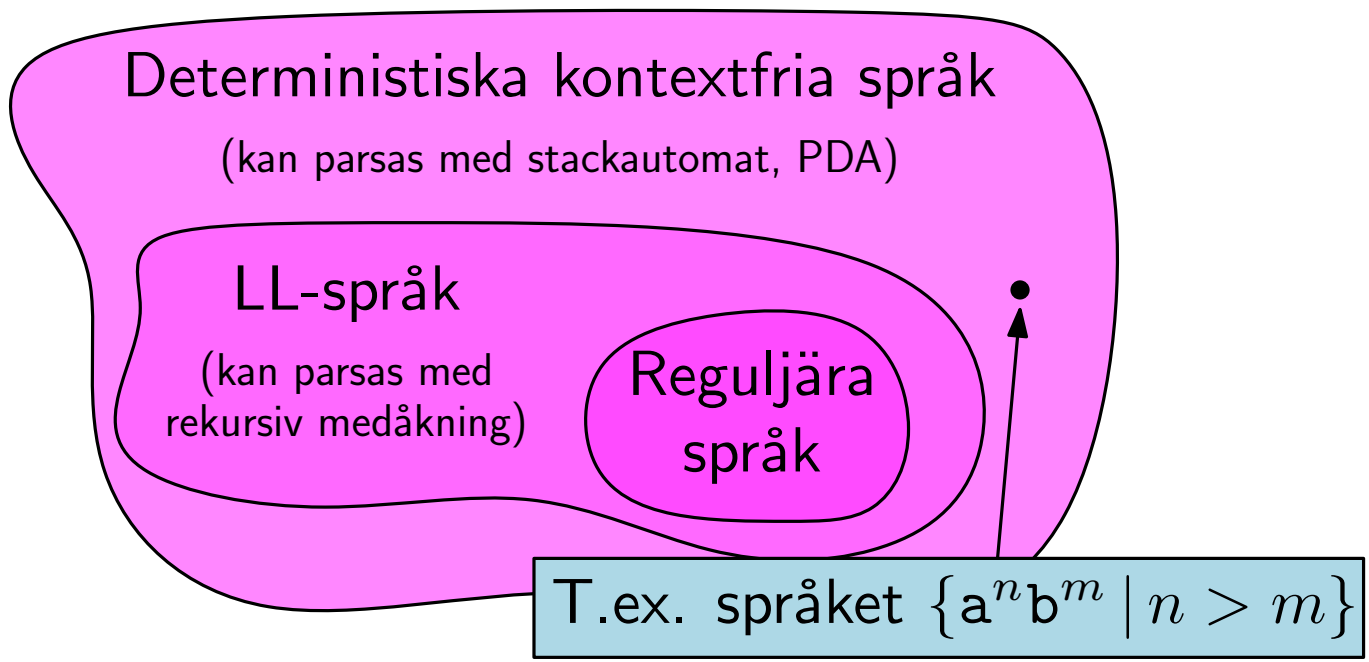
Perspektiv


T.ex. balanserade parentesuttryck
LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk

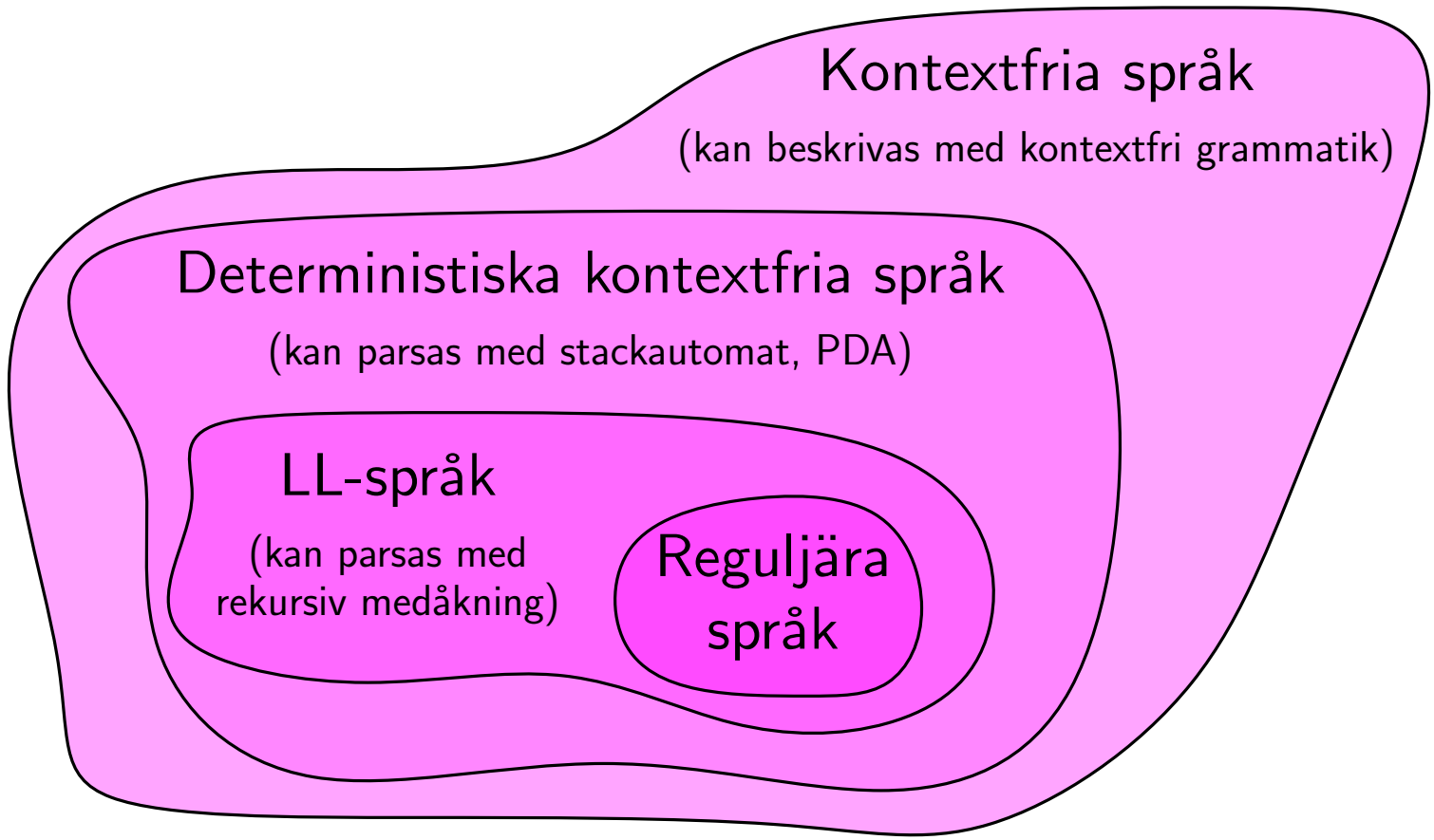
Perspektiv


Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk

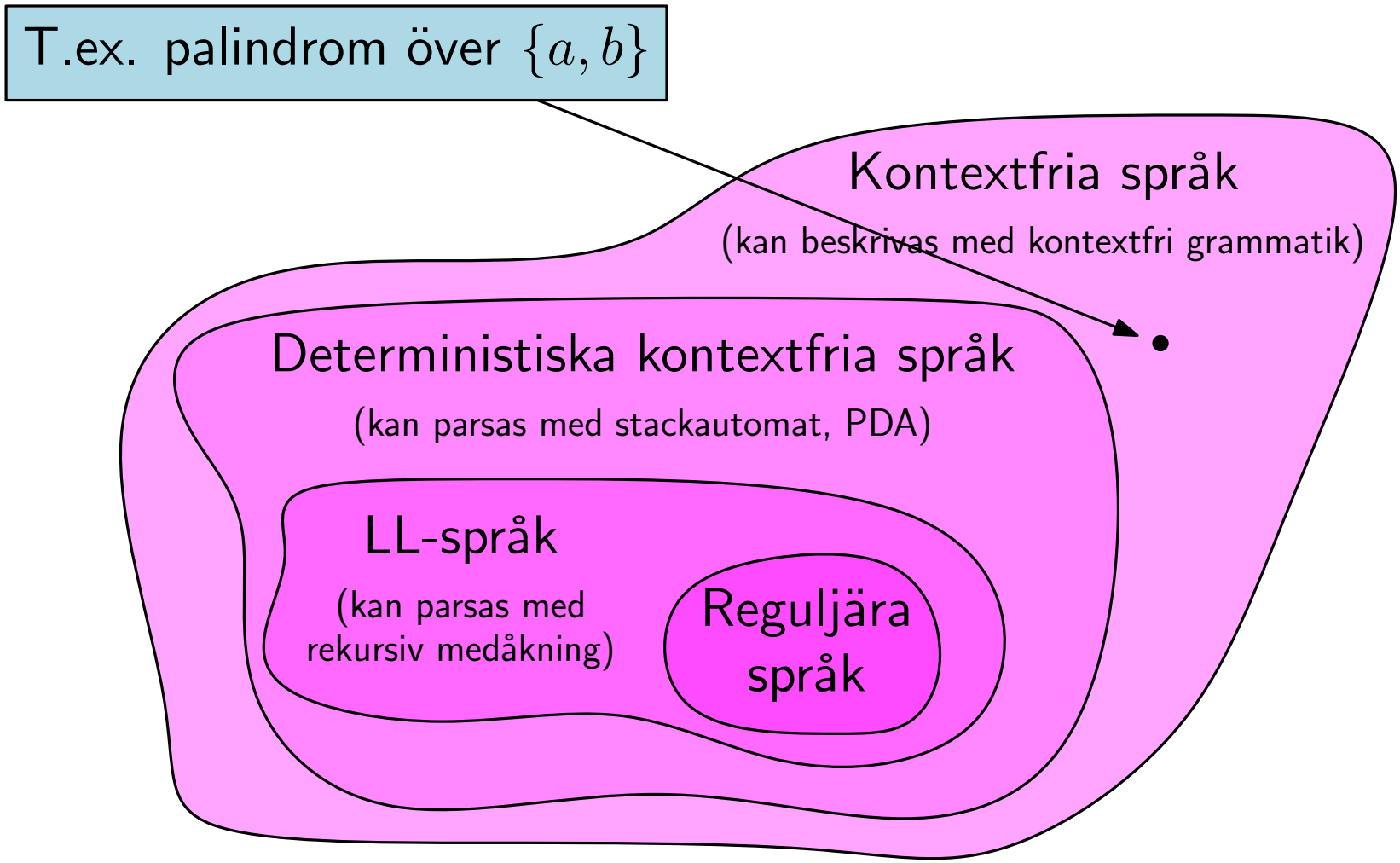
Perspektiv


Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk
T.ex. språket $\{\mathtt{a}^n\mathtt{b}^m \mid n > m\}$

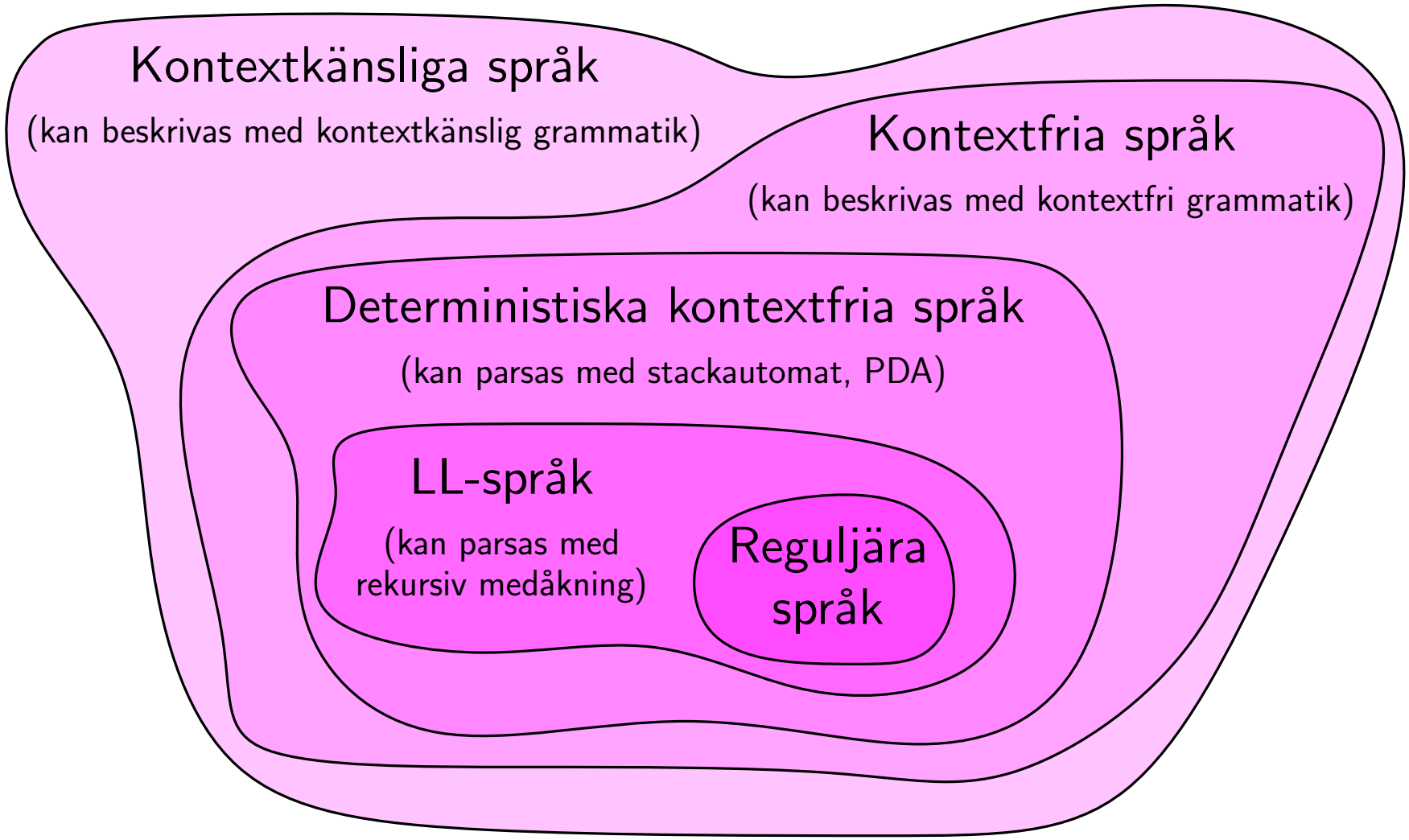
Perspektiv


Kontextfria språk
(kan beskrivas med kontextfri grammatik)
Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk

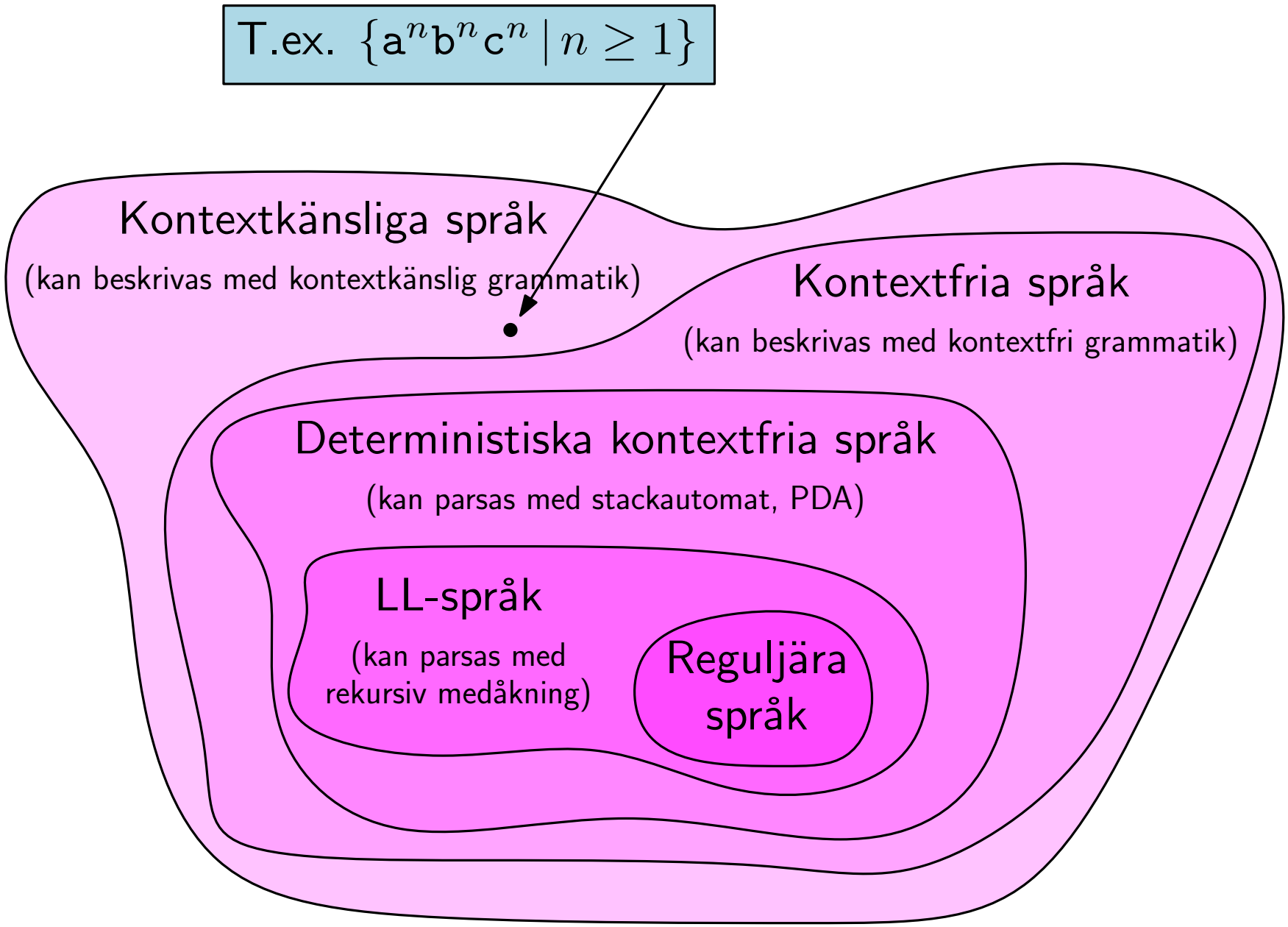
Perspektiv


T.ex. palindrom över $\{a, b\}$
Kontextfria språk
(kan beskrivas med kontextfri grammatik)
Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med rekursiv medåkning)
Reguljära språk

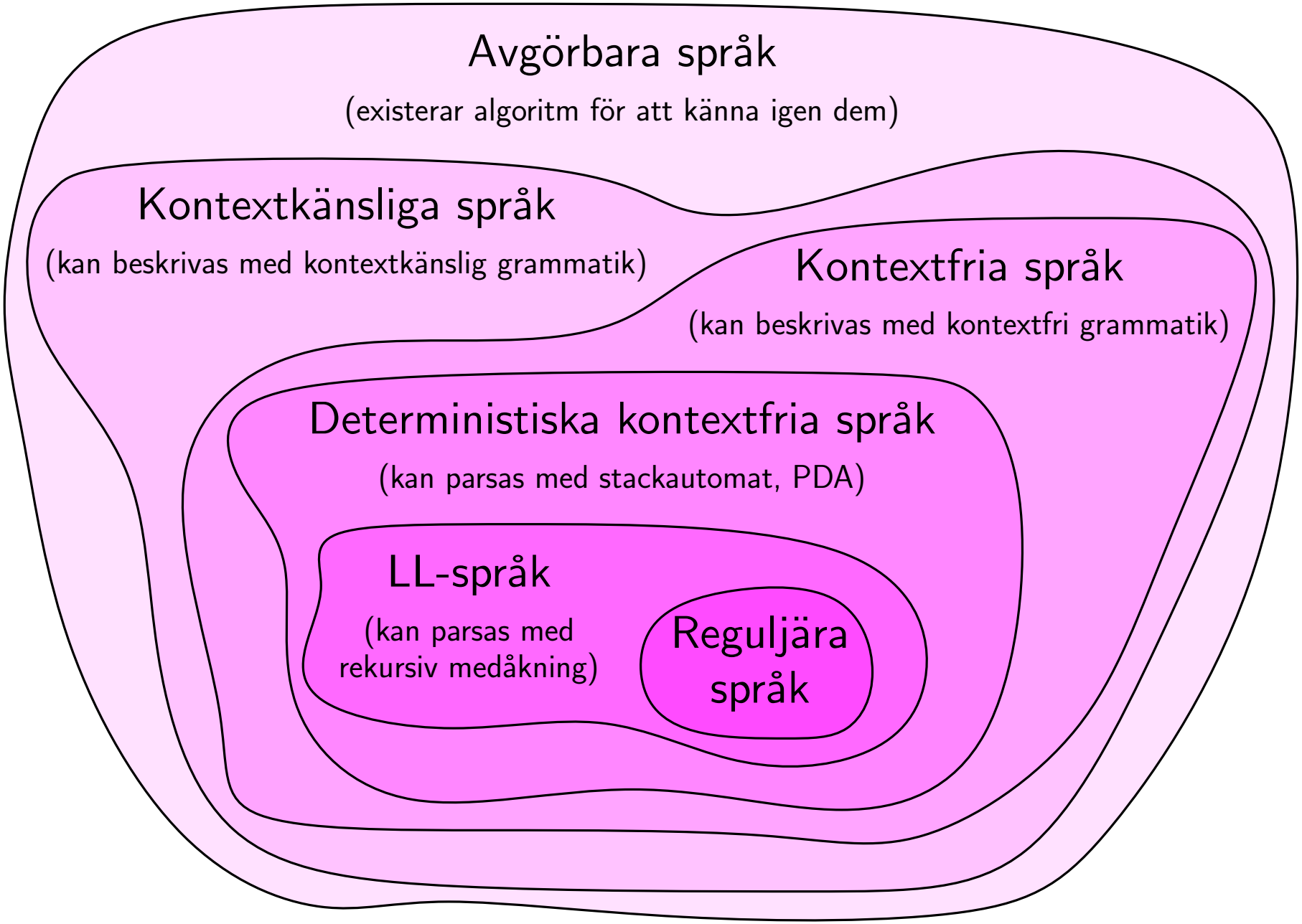
Perspektiv


Kontextkänsliga språk
(kan beskrivas med kontextkänslig grammatik)
Kontextfria språk
(kan beskrivas med kontextfri grammatik)
Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk

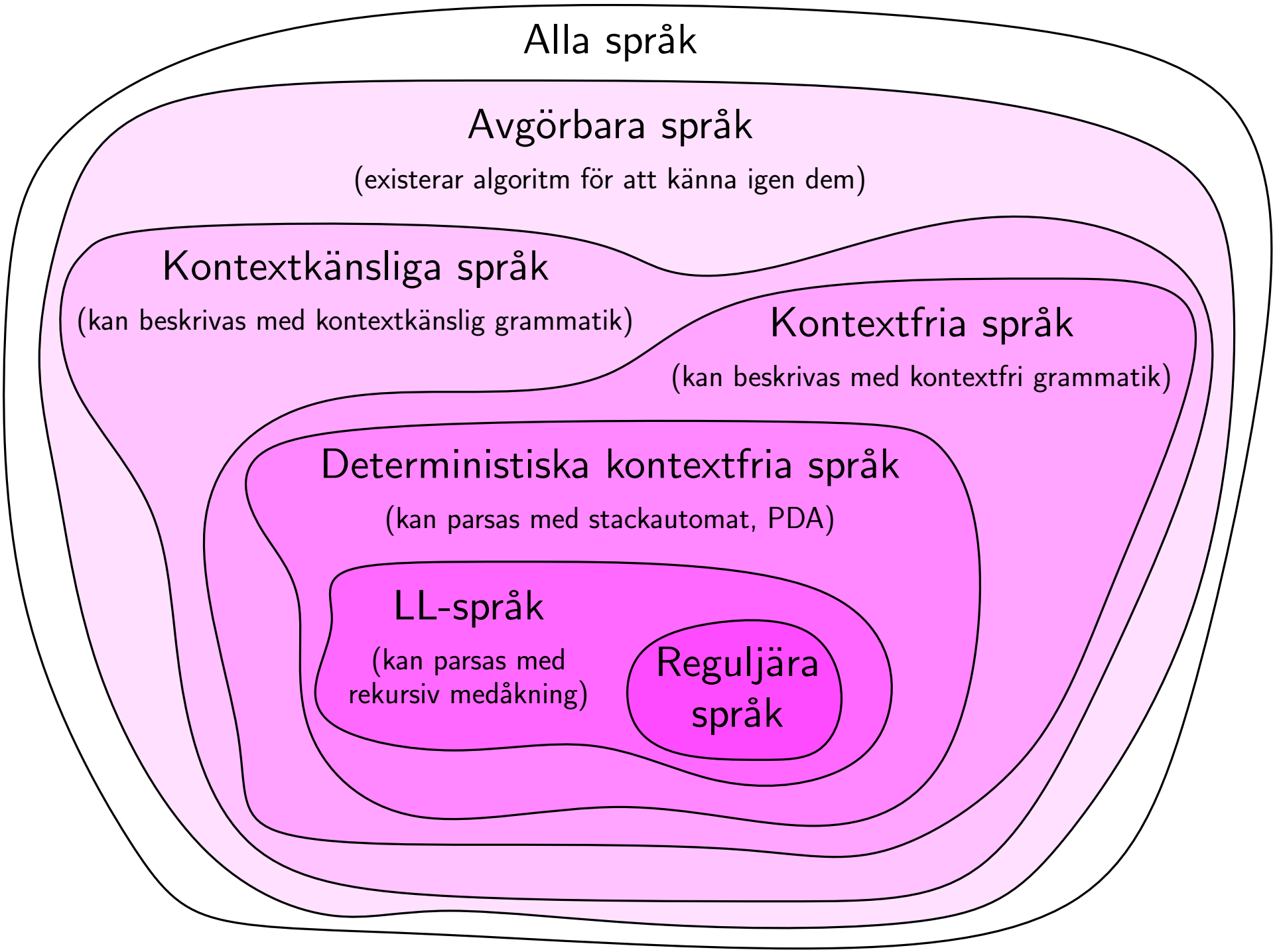
Perspektiv


T.ex. $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \geq 1\}$
Kontextkänsliga språk
(kan beskrivas med kontextkänslig grammatik)
Kontextfria språk
(kan beskrivas med kontextfri grammatik)
Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med rekursiv medåkning)
Reguljära språk

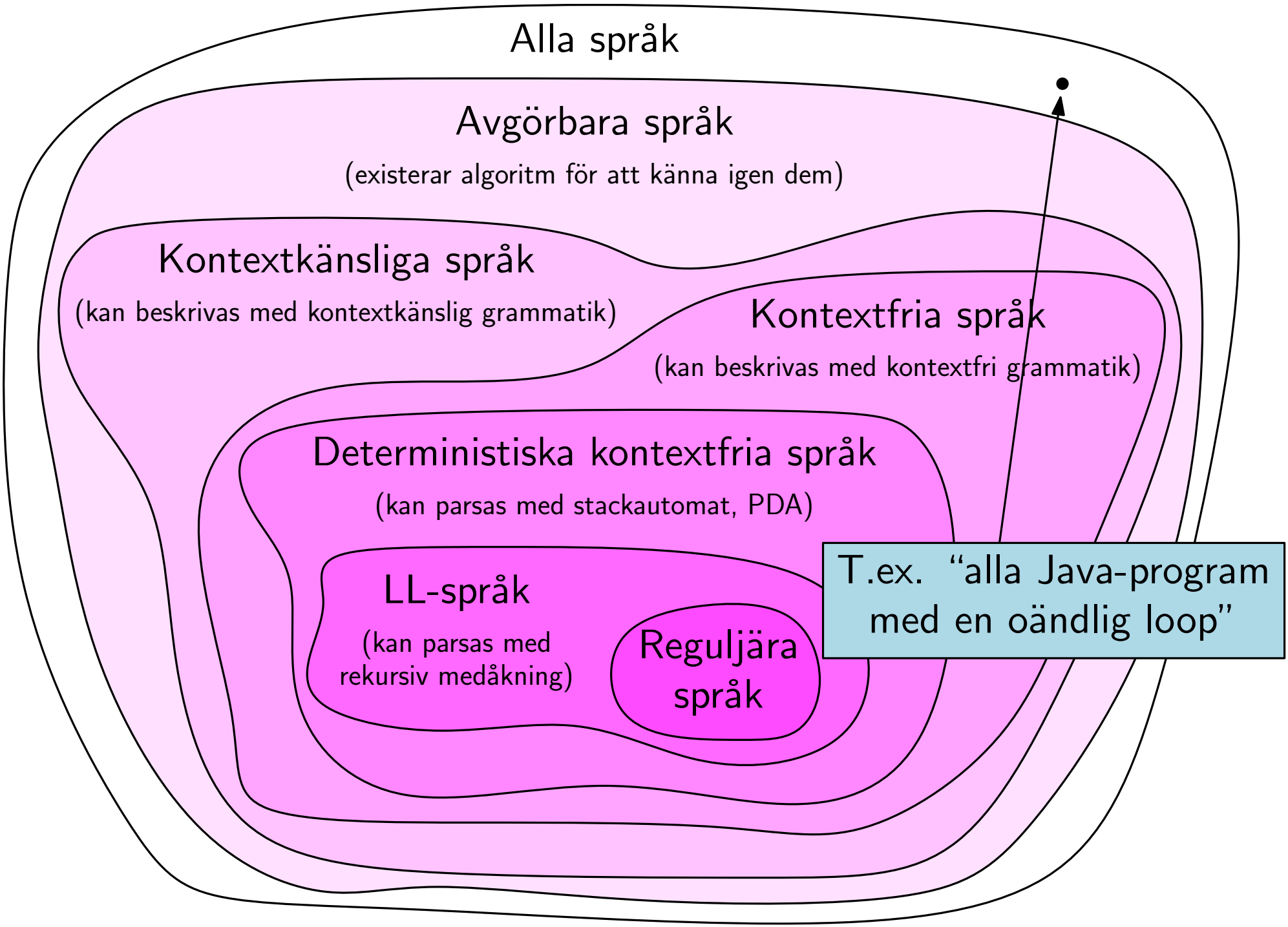
Perspektiv


Avgörbara språk
(existerar algoritm för att känna igen dem)
Kontextkänsliga språk
(kan beskrivas med kontextkänslig grammatik)
Kontextfria språk
(kan beskrivas med kontextfri grammatik)
Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk

Perspektiv


Alla språk
Avgörbara språk
(existerar algoritm för att känna igen dem)
Kontextkänsliga språk
(kan beskrivas med kontextkänslig grammatik)
Kontextfria språk
(kan beskrivas med kontextfri grammatik)
Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med
rekursiv medåkning)
Reguljära
språk

Perspektiv


Alla språk
Avgörbara språk
(existerar algoritm för att känna igen dem)
Kontextkänsliga språk
(kan beskrivas med kontextkänslig grammatik)
Kontextfria språk
(kan beskrivas med kontextfri grammatik)
Deterministiska kontextfria språk
(kan parsas med stackautomat, PDA)
LL-språk
(kan parsas med rekursiv medåkning)
Reguljära språk
T.ex. "alla Java-program med en oändlig loop"

Idag

Rekursiv medåkning, fortsättning

Olika klasser av språk och grammatiker

Parsergeneratorer

Sammanfattning / Inför KS

Parsergeneratorer

De flesta vill nog inte sätta sig ner och skriva en parser för ett helt programmeringsspråk för hand.

Parsergeneratorer

De flesta vill nog inte sätta sig ner och skriva en parser för ett helt programmeringsspråk för hand.

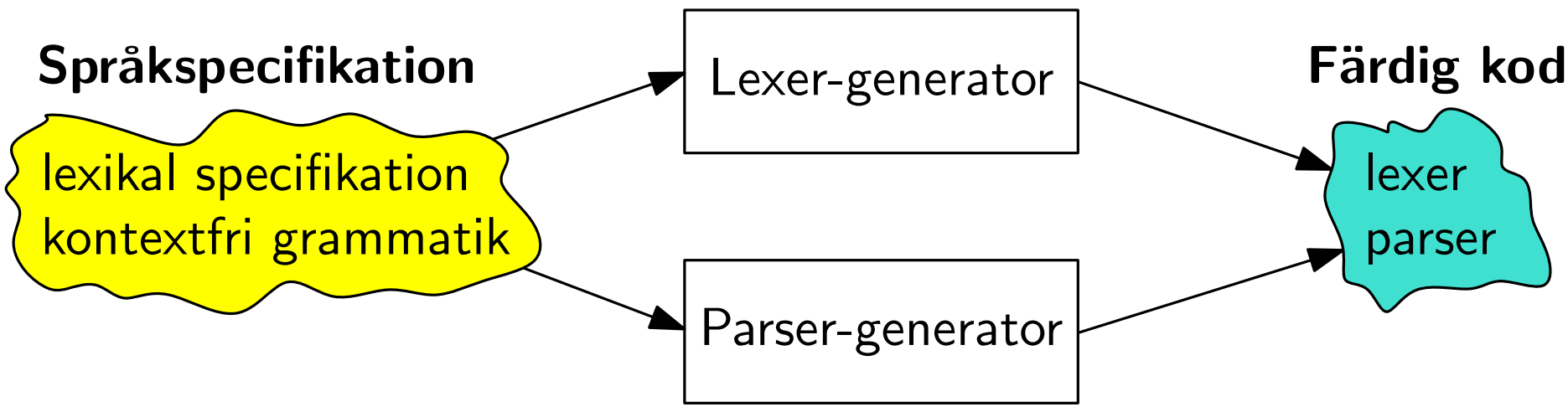
Ofta ganska mekaniskt men många smådetaljer man lätt kan göra fel på $\longrightarrow$ perfekt lämpat för en dator att göra istället!

Parsergeneratorer

De flesta vill nog inte sätta sig ner och skriva en parser för ett helt programmeringsspråk för hand.

Ofta ganska mekaniskt men många smådetaljer man lätt kan göra fel på $\longrightarrow$ perfekt lämpat för en dator att göra istället!

Det finns många verktyg som automatiskt genererar en lexer och en parser från en språkspecifikation.



Exempel på parsergeneratorer

- Lex/Flex och Yacc/Bison, klassiska unix-verktyg, producerar C/C++-kod
  (hanterar de flesta deterministiska kontextfria grammatiker – specifikt, en delmängd som kallas LALR-grammatiker)

- JFlex och Cup, Java-versioner av Flex och Yacc

- DCG i Prolog, inbyggt i språket
  (hanterar alla kontextfria grammatiker, men använder backtracking i Prolog så kan vara långsamt)

- ANTLR, kan generera LL-parser i många olika språk (C/C++, C#, Java, JavaScript, Python...)

- Parsec i Haskell

Exempel på parsergeneratorer

- Lex/Flex och Yacc/Bison, klassiska unix-verktyg, producerar C/C++-kod
  (hanterar de flesta deterministiska kontextfria grammatiker – specifikt, en delmängd som kallas LALR-grammatiker)

- JFlex och Cup, Java-versioner av Flex och Yacc

- DCG i Prolog, inbyggt i språket
  (hanterar alla kontextfria grammatiker, men använder backtracking i Prolog så kan vara långsamt)

- ANTLR, kan generera LL-parser i många olika språk (C/C++, C#, Java, JavaScript, Python...)

- Parsec i Haskell

Binära träd igen

Förra veckan skrev vi en rekursiv medåknings-parser för binära träd, definierade enligt följande grammatik.

BinTree →Leaf LParen Number RParen |
Branch LParen BinTree Comma BinTree RParen

Slutsymboler:
Leaf: strängen "`leaf`"
Branch: strängen "`branch`"
Number: `[0-9]+`
LParen, RParen, Comma: parenteser och kommatecken

Binära träd i Cup

Fil Parser.cup:

```
import java_cup.runtime.*;

// Deklarera slut-symboler
terminal BRANCH;
terminal LEAF;
terminal LPAREN;
terminal RPAREN;
terminal COMMA;
terminal Integer NUM;  // Ett tal har data av typen Integer

// Deklarera icke-slutsymboler, bara en i det här fallet
non terminal ParseTree BinTree; // Har data av typen ParseTree

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

Binära träd i Cup

Fil Parser.cup:

```
import java_cup.runtime.*;

// Deklarera slut-symboler
terminal BRANCH;
terminal LEAF;
terminal LPAREN;
terminal RPAREN;
terminal COMMA;
terminal Integer NUM;  // Ett tal har data av typen Integer

// Deklarera icke-slutsymboler, bara en i det här fallet
non terminal ParseTree BinTree; // Har data av typen ParseTree

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

Binära träd i Cup

Fil Parser.cup:

```
import java_cup.runtime.*;

// Deklarera slut-symboler
terminal BRANCH;
terminal LEAF;
terminal LPAREN;
terminal RPAREN;
terminal COMMA;
terminal Integer NUM;  // Ett tal har data av typen Integer

// Deklarera icke-slutsymboler, bara en i det här fallet
non terminal ParseTree BinTree; // Har data av typen ParseTree

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

Binära träd i Cup

Fil Parser.cup:

```
import java_cup.runtime.*;

// Deklarera slut-symboler
terminal BRANCH;
terminal LEAF;
terminal LPAREN;
terminal RPAREN;
terminal COMMA;
terminal Integer NUM;  // Ett tal har data av typen Integer

// Deklarera icke-slutsymboler, bara en i det här fallet
non terminal ParseTree BinTree; // Har data av typen ParseTree

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

Binära träd i Cup

Fil Parser.cup:

```
import java_cup.runtime.*;

// Deklarera slut-symboler
terminal BRANCH;
terminal LEAF;
terminal LPAREN;
terminal RPAREN;
terminal COMMA;
terminal Integer NUM;  // Ett tal har data av typen Integer

// Deklarera icke-slutsymboler, bara en i det här fallet
non terminal ParseTree BinTree; // Har data av typen ParseTree

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

Kör "`cup -parser Parser Parser.cup`", genererar `Parser.java` och `sym.java`

Lexikal analys för binära träd i JFlex

Fil Lexer.lex:

```
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, Integer.parseInt(yytext())); }
[ \t\n] { }
```

Lexikal analys för binära träd i JFlex

Fil Lexer.lex:

```
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, Integer.parseInt(yytext())); }
[ \t\n] { }
```

Lexikal analys för binära träd i JFlex

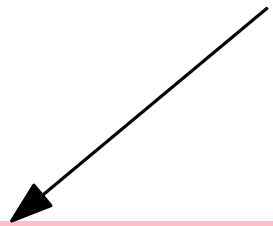
Fil Lexer.lex:

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

%%
%cup
%class Lexer

%%
```

Slutsymbolsnamnen från Parser.cup



```
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, Integer.parseInt(yytext())); }
[ \t\n] { }
```

Lexikal analys för binära träd i JFlex

Fil Lexer.lex:

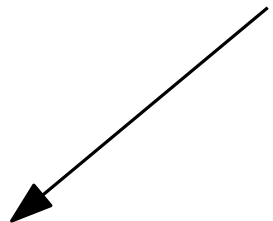
```
import java.lang.System;
import java_cup.runtime.Symbol;

%%
%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, Integer.parseInt(yytext())); }
[ \t\n] { }
```

Slutsymbolsnamnen från Parser.cup



Kör "`jflex Lexer.lex`", genererar `Lexer.java`

Lexikal analys för binära träd i JFlex

Fil Lexer.lex:

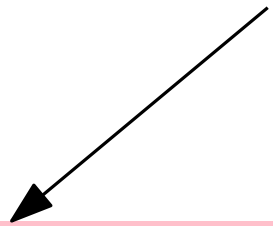
```
import java.lang.System;
import java_cup.runtime.Symbol;

%%
%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, Integer.parseInt(yytext())); }
[ \t\n] { }
```

Slutsymbolsnamnen från Parser.cup



Kör "`jflex Lexer.lex`", genererar `Lexer.java`

Fullständigt exempel med Main-klass upplagt på kurshemsidan

Idag

Rekursiv medåkning, fortsättning

Olika klasser av språk och grammatiker

Parsergeneratorer

Sammanfattning / Inför KS

Sammanfattning

Formella språk

- Språkklasser (ex. reguljära språk)
- Informella språkbeskrivningar (ex. “alla e-post-adresser”)
- Formella språkbeskrivningar/algoritmer (ex. automater, reguljära uttryck)

Reguljära språk

- Reguljära uttryck och DFA
- Skillnad mellan reguljära språk och regex
- Begränsningar (språk som inte är reguljära)

Kontextfria språk

- Kontextfria grammatiker
- Stackautomater, PDA
- Härledningar/syntaxträd
- Tvetydighet

Lexikal analys

Rekursiv medåkning

Förkortningar

DFA *Deterministic Finite Automaton*, ändlig automat.

Den enklaste typen av automat, lika uttrycksfull som reguljära uttryck.

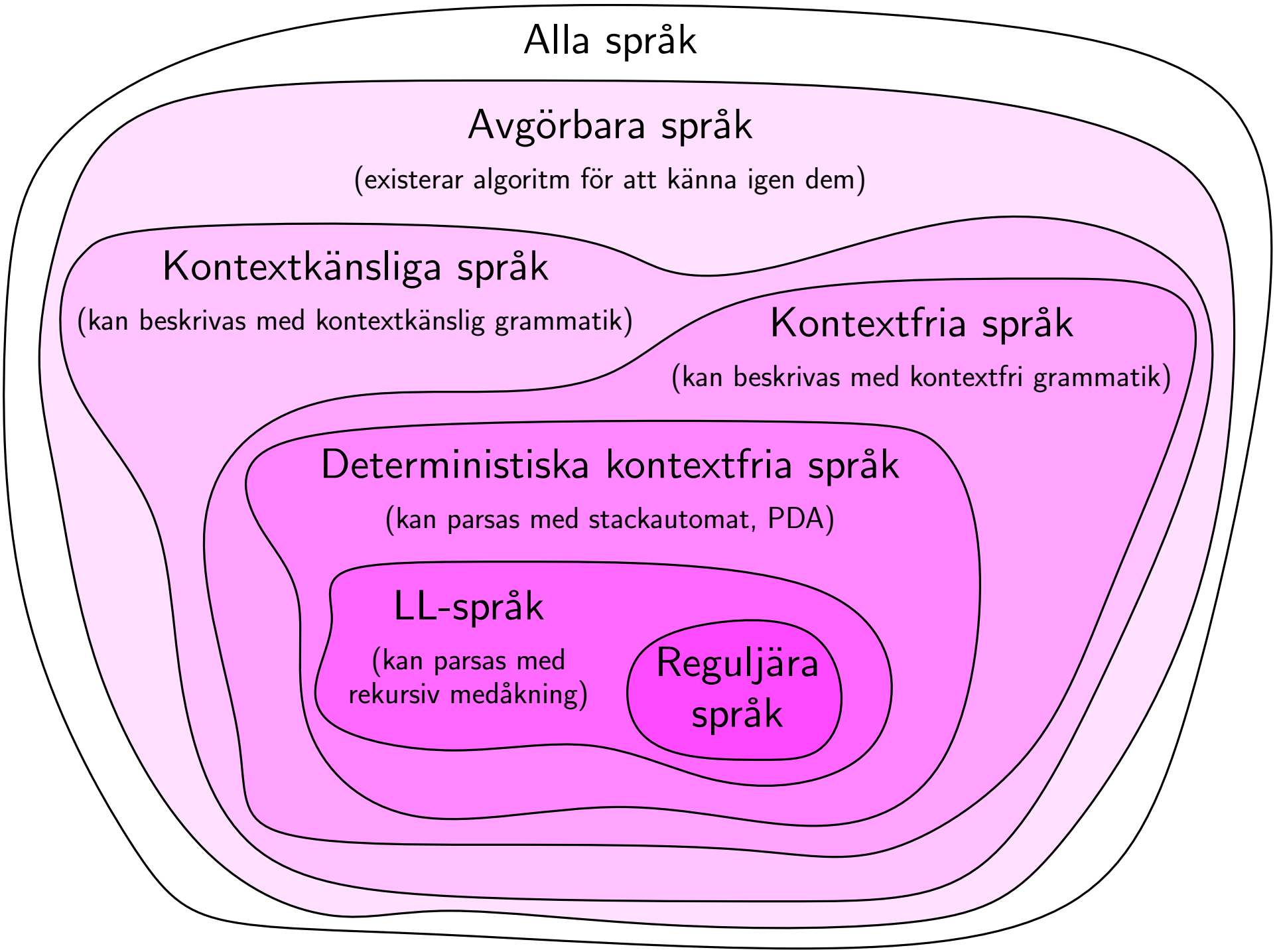
PDA *Push-Down Automaton*, stackautomat.

Som DFA fast med oändligt stort minne i form av en stack.

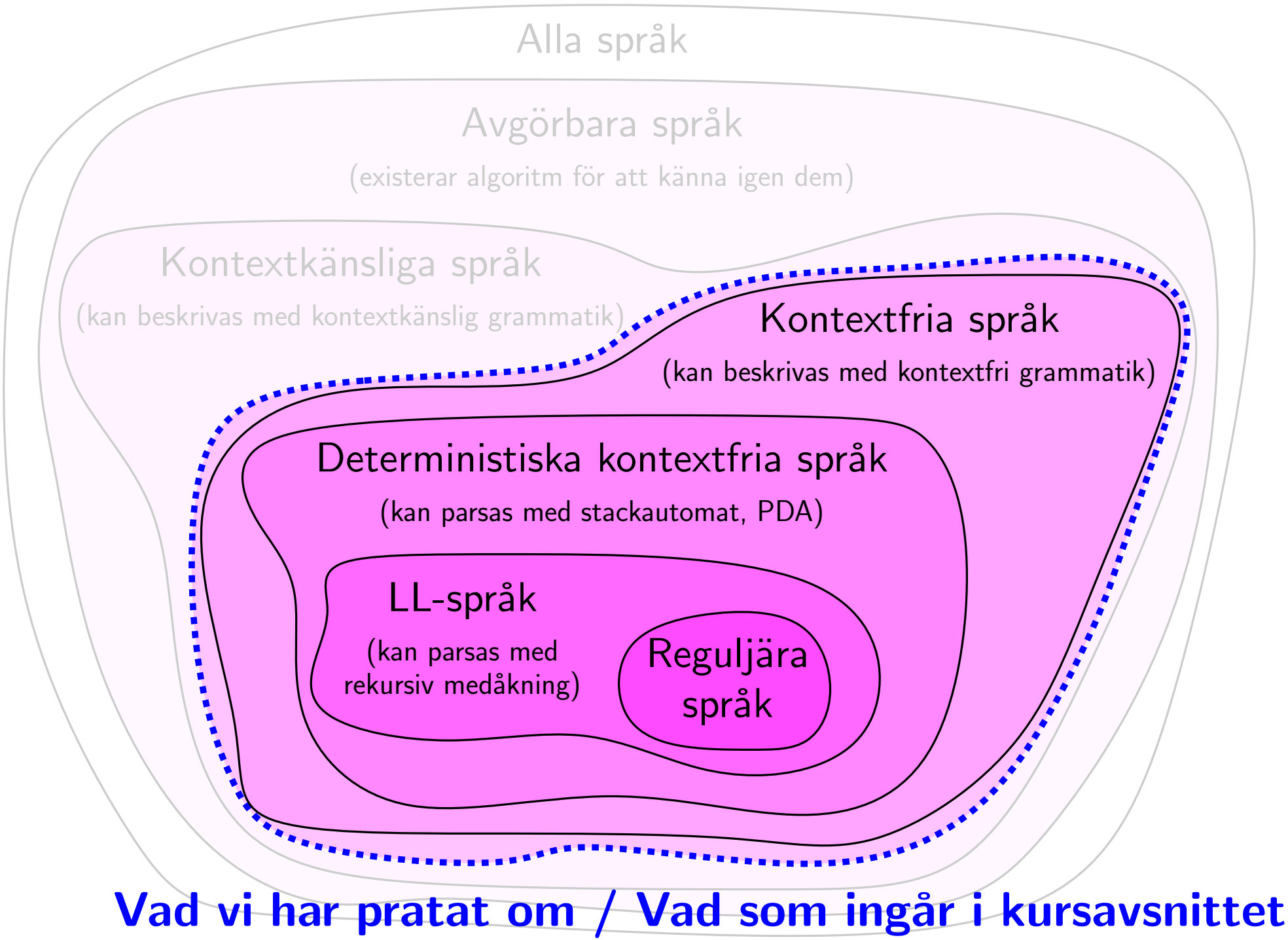
LL *Left-to-right Leftmost-derivation*.

- LL-parser: läser strängen vänster till höger och expanderar icke-slutsymboler från vänster till höger.
- LL-grammatik: grammatik som kan parsas av LL-parser

Perspektiv igen



Perspektiv igen



Vad vi har pratat om / Vad som ingår i kursavsnittet

Vad plugga på?

KS/Tenta kommer inte ta upp något som jag inte pratat om på någon av föreläsningarna.

KS samt tentorna från förra året är någorlunda representativa.

Jag har under ”Kursmaterial” på hemsidan lagt upp länkar till föreläsningsvideos från en del liknande kurser, samt lite annat material, som man kan titta på.

Typiska KS-uppgifter

Grundläggande relationer mellan olika saker, vilka sätt att beskriva språk som är mer eller mindre kraftfulla, etc

Typiska KS-uppgifter

Grundläggande relationer mellan olika saker, vilka sätt att beskriva språk som är mer eller mindre kraftfulla, etc

Givet en DFA/PDA och en sträng förklara vad automaten gör med strängen som indata

Typiska KS-uppgifter

Grundläggande relationer mellan olika saker, vilka sätt att beskriva språk som är mer eller mindre kraftfulla, etc

Givet en DFA/PDA och en sträng förklara vad automaten gör med strängen som indata

Givet en grammatik och en sträng, konstruera ett syntaxträd / avgör om det finns flera syntaxträd / etc

Typiska KS-uppgifter

Grundläggande relationer mellan olika saker, vilka sätt att beskriva språk som är mer eller mindre kraftfulla, etc

Givet en DFA/PDA och en sträng förklara vad automaten gör med strängen som indata

Givet en grammatik och en sträng, konstruera ett syntaxträd / avgör om det finns flera syntaxträd / etc

Skriv ett reguljärt uttryck/kontextfri grammatik eller konstruera en DFA för ett givet språk

Fortsättningskurser

- DD1352 Algoritmer, datastrukturer och komplexitet
  Mer om de högre nivåerna i språkhierarkin

- DD2372 Automater och språk
  Mer på djupet om automater och formella språk.

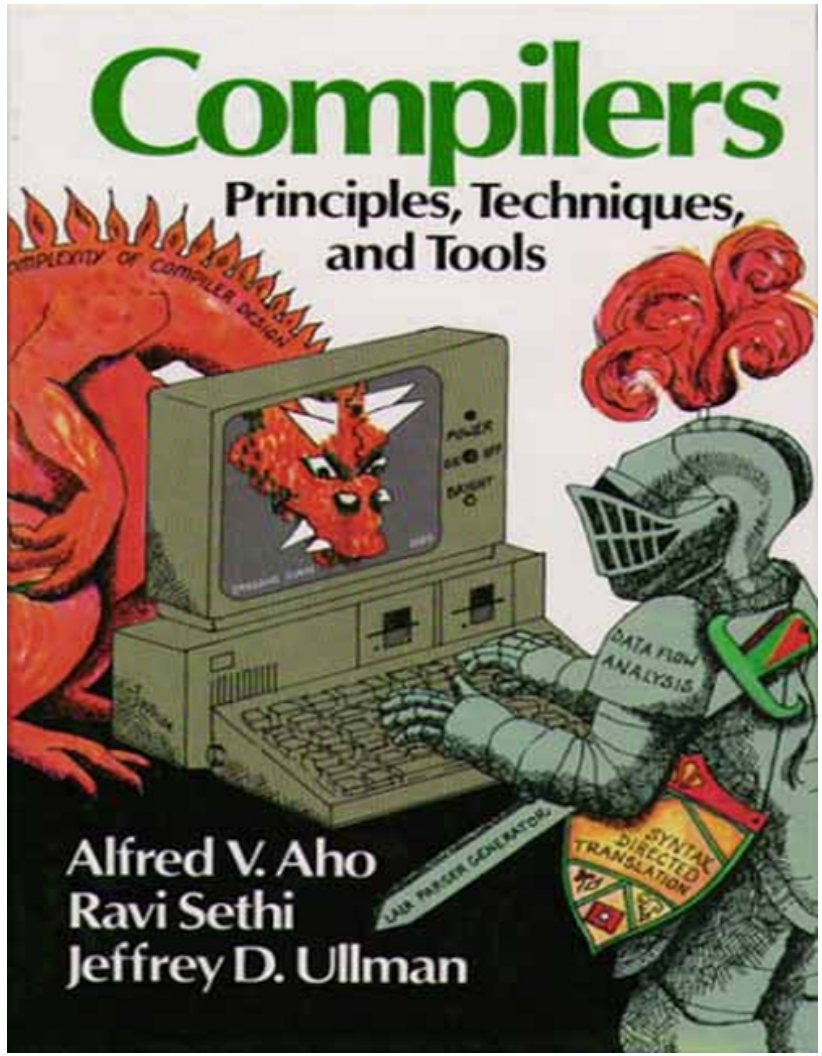
- DD2488 Kompilatorkonstruktion
  Skriv en hel kompilator!

- DD2418 Språkteknologi
  Naturlig språkbehandling, inriktad mot text.

- ID2202 Kompilatorer och exekveringsmiljöer
  Tekniker för implementation av programspråk.
  (Ges i Kista)

För den som vill läsa mer

Aho, Sethi, Ullman:
Compilers: Principles, Techniqes and Tools



Lycka till på kontrollskrivningen!