

## Parsningsalgoritmer

### OH-serie: "recursive descent"/Avslutning

<http://stp.lingfil.uu.se/~matsd/uv/uv09/pa/>



UPPSALA  
UNIVERSITET

Mats Dahllöf  
Institutionen för lingvistik och filologi  
Maj 2009

1

### "Recursive Descent": konfigurationer

Vi måste i varje läge hålla reda på:

- Vad som är kvar av input (som konsumeras vänster till höger).
- Vilka kategorier vi har predicerat, och alltså för tillfället är "skyldiga" att verifiera.  
Stacken används därför här för "obesvarade frågor" snarare än för "verifierade påståenden" som i shift-reduce.

Denna information är den essentiella i Recursive-Descent-systemeriets konfigurationer.

3

### "Recursive Descent": scan (2)

- Man kan predicera ned till terminalsymbolsivå (ordnivå), innan man scannar input, men det blir normalt ineffektivt.
- Det är bättre att scanna input mot icke-terminaler (kategorier). (De är normalt mycket färre.)
- Sista steget, terminal (ord) mot icke-terminal (ordkategori) blir därmed bottom-up.

5

### Recursive Descent som transitionssystem

- Vi kan se Recursive Descent som ett transitionssystem, precis som de olika typerna av shift-reduce.
- Mycket liknar shift-reduce:
  - Systemet befinner sig i varje läge i en konfiguration.
  - Systemet är icke-deterministiskt och tillåter för varje konfiguration ett antal (noll, en eller fler) transitioner till en annan konfiguration.
  - Scan är "motsats" till shift.  
Expansion är "motsats" till reduktion.

7

### "Recursive Descent" allmänt

- Trädkonstruktion: Top-down (predicerande), djupet-först, vänster-till-höger (läsning av input).
- GRAMMATIK: Icke-vänsterrekursiv kontextfri grammatik (CFG).  
(Vi återkommer till egenskapen vänsterrekursivitet.)
- Alltså (om vi tänker i termer av trädbyggande): Börja med att predicera nod för eftersökt kategori, i första vändan startsymbolen (S).

2

### "Recursive Descent": scan

Det finns två huvudtyper av transitioner i en Recursive-Descent-parser, expansion och scan.

#### Scan

Försök verifiera den första predicerade terminalen eller (smartare) icke-terminalen med hjälp av matchning mot input och ta om detta lyckas ett steg framåt i läsningen av input.

Både stacken och bufferten blir ett snäpp kortare om detta lyckas.

4

### "Recursive Descent": expansion

- Försök expandera den första predicerade icke-terminalen på stacken med en regel.
- I trädperspektiv: expandera vänstraste lägsta icke-terminala nod.
- Omskrivningsregler tillämpas vänsterled-till-högerled.  
T.ex. "S → NP VP" tillämpas så att "S" expanderas till "NP VP".
- Vi rör inte inputbufferten vid expansion.

6

### Recursive Descent som transitionssystem

- Vi kan "köra" ett sådant system genom att gå från konfiguration till konfiguration.
- Systemet behöver en initialkonfiguration, så att vi vet var vi skall börja.
- Systemet bestämmer möjliga transitioner.
- Systemet behöver kunna säga när den accepterat input.
- Vi (eller vårt backtrackingsystem) väljer vilka transitioner vi följer.

8

## RD som transitionssystem (1)

Förutsättningar:

- Vi kan tänka oss att en icke-vänsterrekursiv CFG är en del av transitionssystemet.
- En CFG brukar vi skriva som  $(T, N, P, S)$ , där
  - $T$ : terminalsymboler,
  - $N$ : icke-terminala symboler,
  - $P$ : regelbas (mängd regler),
  - $S$ : startsymbol,  $S \in N$ .

9

## RD som transitionssystem (3)

Systemets konfigurationer är av formen  $(stack, buffert)$ , där

- *stack* är en stack med icke-terminaler, som visar vad systemet *har predicerat* och alltså är ”skyldigt” att verifiera.  
Om vi predicerat en NP och en VP (i angiven ordning) och pushat dem (baklänges) på stacken så blir dess utseende  $(NP, VP, \dots)$ , med stackens huvud till vänster.
- *buffert* är den del av input vi för tillfället har kvar.

11

## RD som transitionssystem (5), scan

Grammatik:  $(T, N, P, S)$ .

**Scan med lexikonuppslagning:** Icke-terminal kvittas mot inputsymbol.

- *Konfiguration:*  $((C_0, C_1, \dots, C_n), (w_m, \dots, w_{l-l}))$ .  
(Bufferten skrivs med nästa ord i input, som förväntat.)
- *Villkor:*  $w_m$  tillhör kategorin  $C_0$ . D.v.s.  $C_0 \rightarrow w_m \in P$ .
- *Ny konfiguration:*  $((C_1, \dots, C_n), (w_{m+1}, \dots, w_{l-l}))$ .  
( $C_1, \dots, C_n$  och  $w_{m+1}, \dots, w_{l-l}$  kan vara tomma sekvenser.)

13

## Transitions- kontra härledningssystem

Två huvudformer för de parsningalgoritmer som presenterats:

- Transitionssystem: konfigurationer representerar ett läge på väg mot en bestämd analys av hela strängen.  
Alternativa möjligheter ligger utanför konfigurationen.  
Alternativa vägar kan dock undersökas med backtracking på överordnad kontrollnivå.
- Härledningssystem: Enskilda sats/”items” representerar självständigt giltiga analyser av delsträngar. Charten kan samtidigt innehålla olika alternativa analyser.

15

## RD som transitionssystem (2)

Vi har regler av följande två slag

- Omskrivningsregler:  $L \rightarrow R_0 \dots R_n$   
 $\{L, R_0, \dots, R_n\} \subseteq N$ .
- Lexikoningångar:  $L \rightarrow t$   
 $L \in N$  och  $t \in T$ .

Grammatiken skall vara icke-vänsterrekursiv. Ingen kategori får hamna som första (omedelbara eller medelbara) konstituent i ett uttryck av samma kategori.

Förbjuder, t.ex.:  $NP \rightarrow NP \text{ conj } NP$ .

10

## RD som transitionssystem (4)

Givet input  $I = (w_0, \dots, w_{l-l})$ . (en ordsekvens):

- **Initialkonfigurationen** är  $((S), I)$ .  
Vi har predicerat startsymbolen  $S$  och har hela input kvar.
- **Acceptanskonfigurationen** är  $(( ), ( ))$ . Vi har verifierat allt det vi predicerat (alltså  $S$ , givet initialkonfigurationen) och har inget kvar att läsa i input.  
(Ett träd spänner över hela input. Här presenterar vi algoritmen i recognizer-form, och struntar för tillfället i hur vi bäst samlar ihop syntaxträdinformationen.)

12

## RD som transitionssystem (5), expansion

Grammatik:  $(T, N, P, S)$ .

**Expansion utifrån**  $C_0 \rightarrow R_0 \dots R_m$   $m \geq 0$ .

- *Konfiguration:*  $((C_0, C_1, \dots, C_n), B)$ .  
( $C_1, \dots, C_n$  kan vara tomma sekvensen.)
- *Villkor:* Regeln  $C_0 \rightarrow R_0 \dots R_m \in P$ .
- *Ny konfiguration:*  $((R_0 \dots R_m, C_1, \dots, C_n), B)$

$B$  (bufferten) oförändrad, d.v.s. ingen läsning i input.

14

## För- och nackdelar (på ett abstrakt plan)

- Transitionssystem: Minimal mängd information hanteras.  
Kan göras deterministiska med linjär komplexitet. Risk att de går in i återvändsgränder och, vid backtracking, att de utför dubbelarbete (slänger bort och gör om analyser).
- Härledningssystem: Samlar systematiskt på sig alla dragna slutsatser. Kostnader för lagning och sökning.  
Dubbelarbete elimineras.

16

## Parsningsalgoritmer på kursen

- Vi har främst tittat på renodlade algoritmer för analys utifrån enkla grammatiker (CFG, en typ av dependensgrammatik).
- Ambiguitet har fångats genom backtracking eller lagring i chart. Ingen disambiguering.
- Robustheten följer grammatiken. Typiskt låg grad av robusthet givet dessa typer av grammatik.
- Transitionssystem/orakel ger en ingång till robust disambiguerande parsning. (Utvecklas på senare kurs.)

## Bottom-up kontra top-down

- Bottom-up: Delsträngars analys utförs oberoende av deras kontext.
- Top-down: Delsträngars analys drivs av vad som i kontexten är relevant (typiskt efterfrågat genom prediktion).
- I robusta disambiguerande system torde BU- och TD-mekanismer kombineras.