

Parsningsalgoritmer

OH-serie 2: shift-reduce-parsning

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



UPPSALA
UNIVERSITET

Mats Dahllöf
Institutionen för lingvistik och filologi
April 2009

1

Shift, två varianter

- Konsumera ett ord i input och lägg det på stacken.
Begreppsligen enkelt. Lexikonuppslagning, att tillämpa "Ord \rightarrow Kategori", blir då en reduce-operation.
- Shift med lexikonuppslagning: Konsumera ett ord i input, slå upp det, och lägg dess kategori på stacken.
I detta fall kan "konflikter" uppstå mellan olika lexikoningångar.
Shift kan också vara otillgänglig p.g.a. att aktuellt ord inte finns i lexikon.

3

SR som transitionssystem (1)

Transitionssystemet bygger på:

- En CFG, som vi brukar skriva som (T, N, P, S) , där
 T : terminalsymboler,
 N : icke-terminala symboler,
 P : regelbas (mängd regler),
 S : startsymbol, $S \in N$.
Inskränkning: En symbol får inte i ett eller flera steg kunna skrivas om till sig själv. Inga regler med tomma strängen som högerled.
- En input, en sekvens av ord: (w_0, \dots, w_{l-1}) . (l : längd.)

5

SR som transitionssystem (3)

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

- *stack* är en stack med icke-terminaler, som visar vad maskinen *faktiskt har verifierat* så långt och enligt grammatiken.
Om vi verifierat en NP, en VP och en PP (i angiven ordning) och pushat dem på stacken så blir dess utseende (\dots, NP, VP, PP) , med stackens huvud till höger.
- *buffert* är den del av input vi för tillfället har kvar.
- (Senare lägger vi till en lista för att minnas uträknat syntaxträd.)

7

Shift-reduce-parsning

- En familj av bottom-up-algoritmer.
- Verifierade konstituenten (lästa i input, kontrollerade mot grammatik) bokförs i en stack.
- Omskrivningsregler tillämpas högerled-till-vänsterled.
T.ex. "S \rightarrow NP VP" skulle *tillämpas* NP VP \Rightarrow S.
T.ex. stacken (S, conj, NP, VP) *reduceras* till (S, conj, S).
Stacken skrivs här med huvudet (senast verifierade) till höger.
- Shift: Att ta ett inputord och lägga det eller dess kategori på stacken.

2

Reduce

- Alla dotterkonstituenten måste ha verifierats för att en regel skall kunna tillämpas för reduktion.
- Konflikter (val som ger upphov till icke-determinism) kan uppstå mellan olika tillämpliga regler och mellan att reducera eller shifta.
Genom att reglera dessa val kan man få deterministiska shift-reduce-parsrar. Sådana spelar stor roll inom datavetenskapen.
- Vi skall först se shift-reduce som ett icke-deterministiskt "transitionssystem".

4

SR som transitionssystem (2)

Vi har regler av följande två slag (ytterligare viss inskränkning):

- Omskrivningsregler: $L \rightarrow R_0 \dots R_n$,
 $\{L, R_0, \dots, R_n\} \subseteq N$,
(Som sagt: En symbol får inte i ett eller flera steg kunna skrivas om till sig själv. Inga regler med tomma strängen som högerled.)
- Lexikoningångar: $L \rightarrow t$
 $L \in N$ och $t \in T$.

6

SR som transitionssystem (4)

Givet input $I = (w_0, \dots, w_{l-1})$. (en ordsekvens):

- **Initialkonfigurationen** är $((), I)$.
Vi har ännu inte verifierat någonting och har hela input kvar.
- **Acceptanskonfigurationen** är $((S), ())$. Vi har verifierat ett S -uttryck (vi förutsätter att S är startsymbol) 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.)

8

SR som transitionssystem (5), reduce

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

Reduktion utifrån $L \rightarrow R_0 \dots R_m$ $m \geq 0$.

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

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

9

SR som transitionssystem (7)

Givet en grammatik och en input definierar detta system ett träd med konfigurationer på noderna:

- Initialkonfigurationen är på trädets rot.
- Transitionerna ger ett antal barn till varje nod.
- Det finns en ändlig väg från initialkonfigurationen till varje annan nod (med en konfiguration). Systemet terminerar alltså. Hur vet vi det?
- Backtracking: att traversera detta träd.

11

Transitionssystem

- Klassen är ett "systemskelett", t.ex. `ShiftReduce`. Den implementerar gränssnittet `TransitionSystem`.
Konstruktion:
`ShiftReduce(String file, String[] input)`
- En grammatik av klassen `CFG`
Konstruktion: `CFG(String file)` (från filnamn).
- En input (`String[]`). Internt som `LinkedList<Token>`.
Token : förekomst som sträng och identifierande index.

13

Transitioner

Instanser av klassen `Transit` konstrueras så här, med t som identifierare av transitionstypen:

- `Transit(String t, CFGrule r)`
t.ex. `new Transit("reduce", rule)`
- `Transit(String t, String l)`
t.ex. `new Transit("shiftCat", category)`

15

SR som transitionssystem (6), shift

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

Shift av (icke-terminal) L (alltså med lexikonuppslagning):

- *Konfiguration*: $((C_0, \dots, C_n), (w_m, \dots, w_{l-1}))$.
(Bufferten skrivs med huvudet, d.v.s. nästa ord i input, först, så att det blir förväntad ordning.)
- *Villkor*: Det finns en lexikongång $L \rightarrow t$ i P och $t = w_m$.
- *Ny konfiguration*: $((C_0, \dots, C_n, L), (w_{m+1}, \dots, w_{l-1}))$.
(C_0, \dots, C_n och w_{m+1}, \dots, w_{l-1} kan vara tomma sekvensen.)

10

En exempelimplementation i Java

- Separation transitionssystem (t.ex. shift-reduce) och sökmekanism (backtracking).
- Skall husera ett transitionssystem för dependensparsning framöver.
- Själva transitionssystemet bryts ut som en klass.
- Detta kräver viss överflöd information.
- Implementationen skulle kunna rensas om man bara vill ha en kompakt och effektiv implementation av en enda algoritm.

12

Konfigurationer

Fält i klassen `Config`:

- Symbolstacken: `LinkedList<Token> stack`
(Token förekomster, sträng med indextal.)
- Inputbufferten: `LinkedList<Token> buffer`
- `LinkedList<TreeInfo> tree` för att samla ihop syntaxträdsinformation. (`TreeInfo` enkel enhet med trädinformation.)

14

Transitionssystem

Gränssnittet `TransitionSystem`:

- Definition av initialkonfiguration:
`Config initialConfig()`
- Definition av acceptansvillkor.
`boolean accepting(Config c)`
- Definition av vilka transitioner som är tillgängliga:
`LinkedList<Transit> findTransits(Config c)`
- Definition av hur den konfiguration ser ut som man når genom en transition:
`Config computeTransit(Config c, Transit t)`

16

Backtrackingmekanism

- Konstruktör: `Backtracker(TransitionSystem s)`
(grammatik och input definierat i transitionssystemet)
- `LinkedList<BTState> parse()`
ger en lista med de tillstånd backtrackingmekanismen passerar (givet ett transitionssystem).

17

Backtracking av transitionssystem I

- Tillståndsstack av `BTState` (konfiguration + kvarvarande transitioner). Stack: senast överst. Ger top-down, djupet först traversering av sökträdet/systemet.
- Skapa en `BTState` av initialkonfiguration och övergångarna från den (utifrån transitionssystemet); lägg den på den f.ö. tomma tillståndsstacken, om den har utgående transitioner.

19

Backtracking av transitionssystem

Den här backtrackingmekanismen terminerar om de konfigurationer transitionssystemen tillåter en att nå (via transitioner) från initialkonfigurationen är av ändligt antal.

Vi kan extrahera den information vi vill ur konfigurationerna, t.ex. parse-träd ur acceptanskonfigurationerna.

21

Backtrackingmekanismens tillstånd

Konstruktör:

```
BTState(Config c, LinkedList<Transit> to,  
        Config p, Transit ti)
```

c: aktuell konfiguration

to: möjliga utgående ännu ej följda transitioner

p: föräldrakonfiguration

ti: transitionen från p till c

p och ti bara för spårningspresentationen.

18

Backtracking av transitionssystem II

Själva sökningen/backtrackingen:

- Iteration (while-slinga), tills stacken är tom:
 - gå vidare med översta `BTState`'s första oprövade transition.
 - släng bort denna `BTState` om det var den sista sådana transitionen.
 - utför transitionen, skapa ny `Config/BTState`, räkna ut tillgängliga transitioner, lägg `BTState`'t överst på stacken om det finns någon transition därifrån.

20