Ben 76-12-20

BETA-SYSTEMET: En sammanfattning.

Benny Brodda, Stockholm.

BETA-systemet är ett programmeringsspråk uppbyggt helt och hållet kring substitutionsgrammatikens principer. Substitutionsregler har länge använts inom lingvistiken för att beskriva vissa fenomen (både syntaktiska och fonologiska). Chomsky är väl den som gjort metoden mest känd, men även andra har förespråkat substitutionsreglernas viktiga roll i lingvistiken oberoende av Chomsky, t.ex. Hocket och Harris. I logiken och matematiken har substitutionsmetoden varit länge känd; den "uppfanns" av norrmannen Thue omkr 1910, och logikerna E. Post och A.M. Turing undersökte dess teoretiska aspekter under 30- och 40-talen.

Det aktuella systemet har regler som innebär en mild "generalisering" av regler av Turing-typ utformade som "productions" av Post-typ; generaliseringen är gjord med syfte att åstadkomma regler som är "lagom" bekväma för morfologisk analys. (Som bekant är Turing-reglerna - trots sin enkelhet - kraftfulla nog att åstadkomma allt som överhuvud taget är möjligt att åstadkomma i den här branschen. Ev. generaliseringar adderar i sak ingenting annat än möjligen bekvämlighet.)

1: SUBSTITUTIONSGRAMMATIK:

En substitution innebär att man i en sträng W (mer om detta begrepp strax) ersätter en delsträng med X med en annan delsträng Y. Man erhåller så en ny sträng på vilken man sen kan utföra ytterligare subtitutioner etc. De substitutioner man får utföra bestäms av vilka substitutionsregler man har; dessa sammantagna utgör en substitutionsgrammatik.

En grammatik i den här meningen skall nu användas på följande sätt: Man tar en sträng av ett visst slag som input till grammatiken. Denna knådar så om strängen så länge det finns regler som passar, och när inga regler längre är tillämpliga definieras den nu kanske rätt kraftigt omknådade strängen som grammatikens output.

Vad som får vara input och vad som blir output är nu inte en gång för alla givet; det beror ju på vilken sorts grammatik man skrivit och för vilket syfte. Om t.ex. input är vanliga meningar och output syntaktiska analyser av demsamma kallar man grammatiken en analysgrammatik. Om input utgöres av en enda abstrakt meningssymbol S och output av en faktisk mening har vi en syntesgrammatik, vanligen kallad en generativ grammatik. Många andra slag finnes och det generella begreppet utgör transduktor (transducer). BETA-systemet är ett sätt att göra varje dator till en transduktor.

För att datorn skall förstå hur den skall tillämpa reglerna måste det vara ordning och reda på sättet att skriva reglerna. Reglerna

BETA-systemet: En sammanfattning
Benny Brodda
Proceedings of NODALIDA 1977, pages 20-26

bör ha något så när fixt format (utseende), och det måste vara väldefinierat vad som skall hända när man tillämpar regeln. Har man väl fixerat regel-formatet och dess tolkning ger det sig nästan själv hur man skall få datorn att uppföra sig på det förväntade sättet; de programmeringstekniska detaljerna lämnar vi helt därhån för ögonblicket - i någon mening är det ju ointressant h u r datorn bär sig åt att göra det man ber den och kanske mer intressant att veta v a d man kan be den göra. Här följer nu en kort sammanfattning av det senare.

2: TECKENREPRESENTATION OCH STRÄNGAR:

Med en sträng menas en sammanhängande sekvens av tecken (karaktärer), dvs bokstäver, siffror, typografiska tecken o dyl. Kort sagt allt det man kan skriva ned på en skrivmaskin (inkl. sådana tecken som normalt inte "syns", och därför brukar kallas "vita" tecken: vagnretur, radframmatning, mellanslag, tabbar o dyl). I det aktuella BETA-systemet antages den interna representationen vara ASCII-alfabetet, d v s den representation man erhåller om materialet stansas på en sk Teletype-maskin. I och för sig är BETA-systemet helt generellt och behöver inte alls förutsättas vara ASCII-orienterat, men för att förstå de exempel som ges är det bra att veta vad det innebär. För att kunna benämna tecknen användes då och då tecknets decimala kodnummer. Se separat bilaga. (Varav framgår att t.ex. mellanslag har kodnr 32, "!" har kod 33, "0" har kod 48, "a" har kod 65 etc.

Med en delsträng menas helt enkelt en lämplig sammanhängande sträng som ingår som del av den större; cd är alltså en delsträng av bcda, men ca är inte en delsträng. Om vi har en regel som tillåter utbyte av cd mot ers kan vi alltså få strängen berse.

3: INPUT:

Om man vill databehandla en hel text, kan det kanske vara opraktiskt (ja, omöjligt) att i maskinen handskas med hela texten på en gång. Om det är syntaktiska egenskaper på meningsnivå man vill syssla med, vill man som input ge en mening i taget. Om det är morfologisk analys på ordnivå är det ju ordet som är det meningsfulla objektet. Man måste alltså kunna tala om för datorn hur stora sjok som skall in åt gången. Varje tecken skall i BETA-systemet åsättas ett typvärde som användes för att styra input. Allt som typats som typ 1 (under rubriken DEFTYP i exemplen) betraktas som postavskiljare, eller om man så vill, allt mellan två 1:or blir det datorn får som input. För att också styra output (så att man få läsbart format) behandlas allt som är typat som en 2:a som en potentiell radavslutare. Om man vill jobba på meningsnivå bör man alltså typa punkt och frågetecken som 1:or, mellanslag och komma som 2:or och allt annat som högre. Om bokstäver typas som 4:a får man vid inläsningen sammanföring av ord avbrutna vid radslut med bindestreck. Typningen måste obligatoriskt vara med i en BETA-regeluppsättning (anges under rubriken DEFTYP). Konventionen är den, att de tecken som räknas upp till höger åsättes det typvärde som står längst till vänster; vid denna uppräkning kan man använda antingen tecknet självt eller dess ASCII-kod. Siffror, plus och minus samt vita tecken måste anges med sin kod.

Följande typning rekommenderas som standard (post=rad)

DEFTYP
1: 0
2: 32-34 36-47 58-64 (=Skiljetecken)
3: 48-57 (=Siffror)
4: 65-127 (=Bokstäver)

Anm.: "0" Användes som "stand in" för radslut. Om "0" typas som 2 men "," och "?" blir post=mening. Om mellanslag (32) dessutom typas som 1 blir post=ord.

4: REGELFORMAT:

BETA-systemet arbetar på följande sätt: När en post kommer in (vilket sker automatiskt: det finns ingen särskild läsinstruktion, ej heller någon tryck-d:o, när en post är färdigbehandlad åker den ut och nästa kommer in; när man trycker på "START" kommer första in) kan man tänka sig att en liten tomte, dot kallad, ställer sig längst till vänster i strängen. Sen kutar han fram och tillbaka, tjänstevillig som bara den, och substituerar och stär i. Vid varje ögonblick när han inte håller på och substituerar "är" han någonstans i strängen och det han där gör är att kolla om någon regel är tillämplig just d ä r. Om så i c k e är fallet tar han ett steg till höger och upprepar processen. När han ramlat utanför strängen till höger är "jobbet" klart och man erhåller resultatet som output. Det andra fallet, dvs han upptäcker e t t en regel är tillämplig är förstås det intressantaste:

En regel innehåller tre huvuddelar
A) vilken substitution som skall utföras
B) villkoren för att substitutionen skall få utföras
C) vad göra sen. ("Action")

Regelformatet är mer specifikt följande

X	Y	LC	RC	SC	SR	MV	MD
A			B			C	

Där symbolerna har följande betydelse:

X är den delsträng som ev. skall substitueras: dot bryr sig enbart om sådana delsträngar som är omedelbart till höger om honom.

Y är den delsträng som skall in i stället för X (dvs om substitutionen utföres).

LC = Left Context Condition, villkor på tecknet omedelbart till vänster om X (se nedan)

RC = Right Context Condition, villkor på 1:a tecknet t h om X

SC = State Condition: vid varje ögonblick anteges ett "tillstånd" råda och SC uttrycker ett villkor på det rådande tillståndet

SR = Resulting State: (det tillstånd som inträder om regeln tillämp-

pas)

MV = MOVE, order om vart dot skall ställa sig härnäst

MD = MODE, uppgift om ev. alternativa substitutioner skall utföras (för att ta hand om ambiguösa strängar).

5: VILLKOREN:

För att regeln i fråga över huvud taget skall tillämpas skall tre villkor vara uppfyllda, nämligen på vänsterkontext, på högerkontext och på rådande "tillstånd". Om något av dessa villkor icke är uppfyllt tillämpas regeln icke (och dot prövar en annan regel eller - om ingen sådan finns - knallar ett steg åt höger). Dessa tre villkor utvärderas helt enligt samma princip. Men innan jag beskriver den principen kanske vi skulle tala något om begreppet "rådande tillstånd".

Vid varje ögonblick antages systemet befinnas vara i ett slags tillstånd S, som dot måste kolla av för att se om han överhuvudtaget får tillämpa regeln. Man kan tänka sig det hela som att det hänger kulörta lyktor lite runt omkring, och i varje ögonblick lyser en av dessa (vid starten lyser den "neutrala" vita). I en viss regel kan det ingå instruktioner att ändra detta rådande tillstånd, dvs släcka den aktuella lyktan och tända en ny (detta är innebörden av SR, resulterande tillstånd). Detta är ett sätt att minnas vad som hänt tidigare. I systemet anges tillstånden med tal liggande i intervallet 1-127 (SJälva talvärdet betyder ingenting i sig, det är bara ett namn).

Parametern SC (som också är ett tal i samma intervall men kan också vara negerat) innebär nu ett v i l l k o r på det rådande tillståndet, och detta är viktigt att komma ihåg. Om det rådande tillståndet är '12' och villkoret i regeln säger '17' kan det mycket väl inträffa at '12' uppfyller villkoret '17'. Alltså: parametern SC är ett villkor på rådande tillstånd och inte ett namn på det tillstånd som skall råda. Det är lätt att tänka fel här, men just denna subtilitet gör systemet mycket flexibelt.

Hur vet man nu att rådande tillstånd S uppfyller villkoret SC? Ja, ta exemplet ovan: '12' uppfyller '17' om '12' finns uppräknad bland de tillstånd som är angivna till höger om rubriken 17: under DEFSET. DEFSET har samma konvention som DEFTYP men tillåter "cross-classification". Ex:

```
DEFSET
1:  1 2 3
2:  3
17: 3 12 15 17 32 35 , . ?
```

Tillstånd '1' uppfyller enbart villkor '1', likaså '2' (som alltså ej uppfyller '2'), '3' uppfyller både '1', '2' och '17' och som sagt vad, '12' uppfyller '17'. Villkoret '-17' vilket innebär villkoret '17' negerat, dvs rådande tillstånd får i n t e ingå i '17' uppfylles bara av '1' och '2'.

När det gäller vänster- och högerkontext tittar dot bara på t e c k n e n närmast till vänster och till höger om den sträng X (som

ev. skall substitueras). Vill man jobba med längre kontexter får man använda sig av tillstånd som klättrar upp och ned; BETA-systemet är närmast tänkt för fonologiskt/motologiska tillämpningar och där gäller i förbluffande hög grad att man endast behöver närmaste grannkontext). Parametrarna LC och RC är nu v i l l k o r på dessa närgrannar, och villkoren är av samma art som tillståndsvillkoren. Anger man villkoret '17' som LC betyder det att tecknet omedelbart till vänster om X skall finnas uppräknat till höger om 17: under DEFSET. I exemplet ovan är tecknen mellanslag (32), punkt, komma, frågetecken uppräknade vid '17', dvs typiska ordavskiljare; LC = 17 skulle alltså innebära att regeln endast får tillämpas om tecknet till vänster är en ordavskiljare, eller m a o enbart i ordbörjan.

LC, RC eller SC = 0 innebär "intet villkor" (noll defaultvärdet)

6: ACTION:

Om dot nu konstaterat att regeln f å r tillämpas, vad gör då dot? Ja, först och främst utföres substitutionen, men vad mer? Först och främst skall rådande tillstånd ändras till SR resulterande tillstånd, men endast under förutsättning att SR är = 0, i annat fall b i b eh å l l e s rådande tillstånd.

Nästa parameter MV (MOVE) säger var dot skall gå härnäst, och man har i stort sett 6 standardpositioner att gå till, och dessa är inte fixa utan relaterade till den nyss utförda substitutionen. För att förklara dessa framställer vi det hela schematiskt: Fig 1, är hur det ser ut just innan substitutionen utförts. Varje ruta symboliserar ett tecken, en avlång låda en sträng. Den sträng som skall bytas ut är "mittlådan". De tre lådorna tillsammans utgör den aktuella strängen.

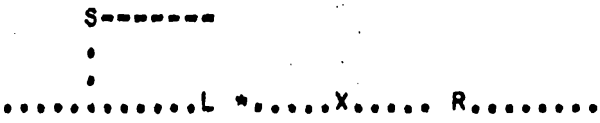

Fig 1.

(Dot, dvs "*", finns inte med i själva strängen utan kutar s å s ovanpå)

Vi antager nu att dot kollat att R ingår i RC, L ingår i LC och S ingår i SC. Dot plockar då bort X och plugger in Y i stället. Just vid själva substitutionsögonblicket kan man tänka sig att hela strängen man arbetar med har tre delar:
Delsträngen till vänster om Y, Y självt och delsträngen till höger om Y, detta symboliserat med de tre lådorna nedan i fig 2. Varje sådan delsträng har (kan ha) ett första tecken och ett sista tecken, vi har alltså 6 väldefinierade punkter i strängen. Vi numrerar dessa från vänster till höger 1 - 6, och får då de standardpositioner till vilka man kan dirigera dot. Skriver man 4 som MV parameter ställer sig dot alldeles t v om det sista tecknet i Y (Om Y är tom, d v s X deleterats blir förstås positionerna 3, 4 och 5 desamma). Utöver dessa sex standardpositioner kan man dirigera dot till några andra positioner utanför den aktuella strängen.

Dessa är:

0: Hela strängen deleteras
7: Strängen betraktas som färdigbehandlad och ges som output (ut 1)
8: Strängen ges som output på separat fil (ut 2)
9: Strängen dirigeras till radskrivaren (Lpr)

Genom möjligheterna 8 och 9 kan man använda BETA som ett mycket avancerat excerperingsprogram. 0-Ning kan tillämpas om flera alternativ bearbetas samtidigt och något av dem visade sig vara dödfött; jfr MD-parametern nedan)

```
0   1        2 3    4 5        6 7    8   9
•   •........L YYYYYY R........• •    •   •
```

Positionerna 0, 1, 6 och 7 finns faktiskt som en del av strängen som BETA arbetar med och innehåller alla tecknet " " ("brädgård"). Strängen i utrymmet mellan 1 och 6 är den egna strängen man jobbar med. Startposition är 1, "halkar" man ut till höger är man färdig, "halkar" man ut till vänster dör strängen.

Default för MV är 5.

7: ALTERNATIV

Om man håller på med syntaktisk analys är det bekant att man ibland kan erhålla strängar (konstruktioner) som är analyserbara på mer än ett sätt. Problemet är att kunna ange samtliga möjliga analyser. Detta är ganska lätt att åstadkomma i BETA, ÅTMINSTONE OM MAN KAN SKRIVA REGLERNA SÅ ATT MAN SER ATT EN VALSITUATION ÄR FÖR HANDEN. Detta anges i BETA-reglerna med den sk MODE-(MD) parametern. Denna parameter kan ha 3 värden 1,2 och 3 (samt också "ekvivalenta" värden 5, 6 och 7; Om dessa senare användes "frågar" dot om regeln i fråga får användas. Denna facilitet förutsätter att man kör BETA interaktivt och sitter vid terminalen och besvarar dessa frågor).

Reglerna i BETA tolkas normalt disjunktivt. Det betyder att den första regel uppifrån som är tillämpbar tillämpas och om MD är = 1, vilket är det normala (Jfr dock avsnittet om regelordning), så är det därmed inget mer tal om den saken. Om MD är satt till 2, lägges det nyss erhållna resultatet en stund åt sidan, och dot söker efter ytterligare en regel, och om en sådan återfinnes utföres även denna substitution, är även denna regel märkt som en 2:a slaskas även det nya resultatet undan och nästa tillämpbara regel tillämpas etc.

De uppstående alternativen lägges sedan i en kö, och denna kö avbetas först innan någon ny post inhämtas utifrån. I denna kö lagras det "halvfabrikat" som hittintills uppstått inklusive det aktuella tillståndet. När detta halvfabrikat återhämtas från kön fortsätter man alltså behandlingen precis där man avbröt den (för att fortsätta med andra alternativ).

MD = 3 är ett specialfall av 2. Substitutionen utföres och resultatet av densamma lagras i kön precis som i fallet MD = 2. Någon ny regel uppsökes inte och dot fortsätter att jobba med "originalet" som ingenting hade hänt,: man kan alltså både ta och släppa en regel.

Default för MD är 1.

8: GENERELLA STYRPAR:

Överst på varje regellista skall upp till 3 generella parametrar skrivas. Desse kallar vi REGDEL, HL och PROGVAR. REGDEL (Regeldelimitern) är ASCII-koden för det tecken som användes för att definiera vad som menas med att vänsterledet och högerledet i en substitution är "slut". Default är 32 (mellanslag), men om man måste skriva en regel omfattande mellanslaget måste man välja ett annat tecken (som dock aldrig får förekomma i en regel).

HL förklaras i avsnittet om regelordning nedan. PROGVAR kan ha värden 1 eller 3 och att man har lite olika programvarianter. 1 är defaultvärdet och innebär precis det system som vi förutsett här, PROGVAR = 3 innebär att man har inga statusparametrar i reglerna. Regelformatet blir då:

X Y LC RC MV MD

9: REGELORDNING:

Reglerna utvärderas i princip i den ordning de lästs in. Den första regeln uppifrån som är tillämpbar tillämpas. En sådan regelordning kallas för disjunktiv. Att reglerna utvärderas disjunktivt är dock en sanning med modifikation, det är i stort sätt sant, men inte alldeles. Hur utvärderingen i detalj utföres bestämmes av parameter nr 2, HL, i de generella styrparametrarna. Denna parameter lägger upp reglerna i s k hash-klasser enligt vänsterleden i substitutionsreglerna. Om HL t.ex. är =2 (defaultvärdet) betraktas alla regler med minst två identiska tecken i vänsterledet ("X") i substitutionsreglerna som hörande till samma hashklass, och dessa utvärderas före de som innehåller endast ett tecken i vänsterledet. Om HL sättes = 3 utvärderas de regler som innehåller minst tre tecken före de som innehåller två, vilka i sin tur utvärderas före de som innehåller endast ett tecken. Inom varje hash-klass utvärderas reglerna strikt disjunktivt (detta är viktigt att komma ihåg, så att dot inte "nyper" fel regel tidigare). Huvudprincipen för regelskrivandet skall alltså vara att regler med starkare villkor (d v s längre eller havande "härdare" kontext-villkor) måste skrivas tidigare i listan. Något krav på att reglerna skall skrivas i bokstavsordning eller dyl föreligger inte. Tvärtom, det rekommenderas att regler som "logiskt" hör ihop sammanföres. Sådana logiskt hopförda regler kommer då att fungera som subrutiner i vanliga programmeringsspråk. Konventionen med hash-klassernas utvärderingsordning underlättar att man kan skriva reglerna på detta sätt.