Magisteruppsats i datalingvistik
Institutionen för Lingvistik
Göteborgs universitet.
Vårterminen 1999

ChartDeluxe

Ett verktyg för grafisk presentation av chartparsrar

Författare: Peter Svensson
Handledare: Robin Cooper

Abstract

The goal of my thesis is to develop a pedagogical tool which aims to help students at the institution of Linguistics at the University of Gothenburg understand how different parse strategies work. Wirén (1992) describes several different chart algorithms from which I have choosen to implement following strategies:

- Topdown
- Bottomup
- Bottomup à la Kilbury
- Topdown with bottomup filter
- Bottom up with topdown filter
- Bottomup á la Kilbury with topdown filter
- Bottomup á la Kilbury with bottomup filter
- Bottomup á la Kilbury with topdown & bottomup filter

The user can not only choose which of the strategies she/he wants to use in a current parse but also chose which search strategy the parser uses depth or breadthfirst. The user also has the opportunity to create, save and edit grammars and lexicon. The program ChartDeluxe v1.0 is exclusively implemented in the programming language Java.

In this paper I describe how the program is structured and other aspects which can be of interrest for possible developers. The paper also includes a user manual for the program ChartDeluxe v.1.0.

4.3.2 Detaljerad systembeskrivning 34
4.3.2.2 Klasser 34

4.4 Utvärdering 39
4.4.1 Eventuella ändringar i koden 39
4.4.2 Buggrapport 40

4.5 Utvecklingsdokumentation 40

5. Resultat 41

5.1 ChartDeluxe v1.0 41
5.1.2 Begränsningar 42
5.1.3 Möjliga utvidgningar 42

5.2 Slutsatser 43

5.3 Sammanfattning 43

6. Källförteckning 44

1. Inledning

Det finns en mängd olika sätt att analysera textstycken. När man som datalingvist skall sätta sig in hur olika strategier fungerar kan det ibland uppstå problem då de definitioner som ämnar beskriva metoderna tenderar att vara en aning kortfattade. Många anser att det bästa sättet att få förståelse för hur en parsemetod fungerar är att själv försöka konstruera en dylik. Problemet är att oerfarna utvecklare av en eller annan orsak ofta inte har möjlighet att genomföra det. Ett mindre krävande sätt att utforska hur en parsemetod fungerar är att man som användare nyttjar ett verktyg som utifrån given indata demonstrerar resultatet från analysen på ett någorlunda pedagogiskt sätt.

Målet med mitt arbete är att implementera ett verktyg som grafiskt demonstrerar olika chartparsestrategier. Tanken är att programmet skall användas som ett hjälpmedel i utbildningen på Datalingvistprogrammet på Göteborgs universitet. Verktyget skall tillhandahålla flera olika chartparsestrategier. Givet en av användaren vald strategi, en grammatik, ett lexikon och en indatasträng skall programmet presentera den graf som blir resultatet av en parsning.
Tanken är att användare på så sätt skall kunna undersöka skillnader och likheter mellan de olika parsestrategier programmet presenterar. Det är också meningen att användaren skall ha möjligheten att skapa, redigera och spara egna grammatiker och lexikon i programmet.

Eftersom programmet skall användas i pedagogiskt syfte känns det viktigt att gränssnittet är användarvänligt. Programmet ska implementeras i programspråket Java vilket verkar passa alldeles utmärkt då det finns ett stort stöd för gränssnittsutveckling i programspråket. Dessutom tror jag det faktum att Java är plattformsoberoende kan det öka möjligheten för användaren att kunna arbete med programmet där det passar dem i stället för att vara tvungna att nyttja det i skolan.
Många studenter har nu mer dator hemma men de tvingas ändå ofta slåss om tid vid skolans datorer eftersom de aktuella programmen inte kan köras på de operativsystem studenterna har installerade på sina egna datorer.

Jag har länge känt att jag ville lära mig programspråket Java och jag är väldigt intresserad av att designa och implementera gränssnitt och i Torbjörn Lagers förslag till examensarbete, implementera javaprogram som demonstrerar olika chartparsestrategier, såg jag inte bara möjligheten att kombinera arbete inom dessa områden utan även möjligheten att själv utveckla ett fristående program som är tänkt att faktiskt användas.

2. Bakgrund

I det här kapitlet kommer jag att förklara begreppet chartparsing. En kort presentation av de verktyg jag använt mig av för att utveckla programmet återfinns också här. Jag tar för givet att läsaren är bekant med begrepp som parsing, grammatik, lexikon osv. och kommer därför använda dem utan närmare förklaring.

2.1 Chartparsing

Det finns en uppsjö av parsealgoritmer för analys av text. De vore knappast någon lögn att påstå samtliga algoritmer på något sätt strävar efter att arbeta så effektivt som möjligt. Arbeta effektivt innebär i ofta att algoritmen skall generera en analys av indatatexten under så kort tid som möjligt. Den information analysen presenterar kan vara av olika natur och kvantitet. Ett gemensamt dilemma för ett flertal av algoritmerna är att utveckla korrekta analyser utan att i en och samma parsning behöva utforska en sökrymd upprepade gånger. För att tydligare belysa problemet presenterar jag ett exempel nedan.

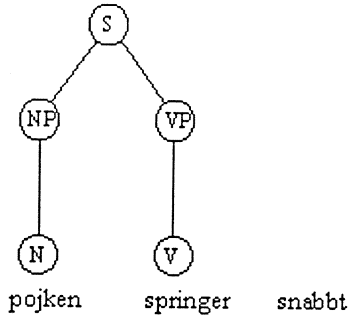
Anta att vi har :

en Grammatik **S** → **NP VP**
NP → **N**
VP → **V**
VP → **V AD**

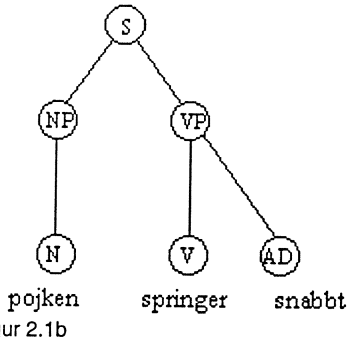
ett Lexikon **N** → **pojken**
V → **springer**
D → **fort**

och indata strängen: **pojken springer fort**

Vi antar att vi skall utföra en ordinär topdownparsning. För att parsern skall kunna hitta den fullständiga analysen av indatasträngen måste den identifiera en aktuell fras för varje regel som har ett element med dess kategori i sitt högerled. Skälet till det är att parsern inte har någon möjlighet att spara information om vilka fraser som tidigare identifieras i den aktuella analysen.



Figur 2.1 a



Figur 2.1b

Exempel 2.1.1

I exemplet ovan tvingas parsern söka efter V frasen två gånger .
Trots att den redan identifierat den en gång, måste den söka efter den igen för nästa regel där ett element av den kategorin förekommer. Dubbelarbetet leder till onödig tidsåtgång .Förvisso är varken grammatik, lexikon eller indatasträngen särskilt omfattande i det aktuella exemplet men i många fall arbetar man med mycket större grammatiker och text massor och där kan den tidsförlust dubbelarbetet ger upphov bli väsentligt större.

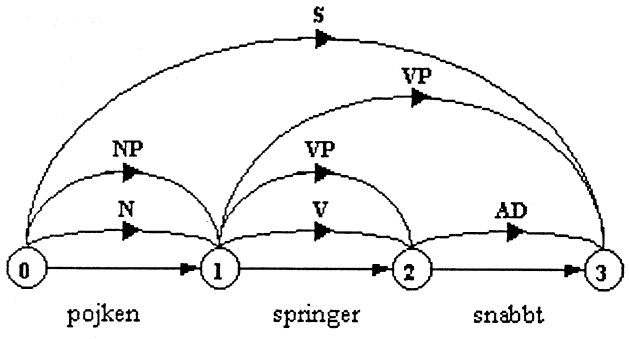
2.1.1 WFST

I exempel 2.1.1 har det antytts att det skulle vara möjligt att minska söktiden i en parsing om man lät parsern ha tillgång till en minnesdel där resultatet från tidigare sökningar i den aktuella parsingen lagras. WFST eller Well Formed Substring Table är en typ av minneskomponent som lagrar de fraser som parsern tidigare framgångsrikt identifierat. Notera att det endast är de resultat från lyckade sökningar som lagras i WFST:n misslyckade sökningar ignoreras.

(Gazdar & Mellish 1989) väljer att definiera en WFST som en riktade, icke cyklisk graf med unika start och slut noder som innehåller bågar som är märkta med syntaktiska kategorier och ord. Den första noden märks med 0 och den sista n där n är antalet ord i indata strängen.

Bågarna i WFST är tupplar med följande struktur: $<i, j, X_0>$.
<i> = Bågens starttillstånd. Ordningsnumret på den nod i grafen från vilken bågen utgår.
<j> = Bågens sluttillstånd. Ordningsnumret på den nod till vilket leder.
$<X_0>$ = Bågens syntaktiska kategori.

Resultatet av en WFST stödd parsning presenteras i form av en graf istället för parseträd som i exempel 2.1.1. De olika noderna är konkreta ordningstillstånd som har direkt anknytning till indatasträngen. Bågarna mellan noderna representerar de fraser som parsern framgångsrikt identifierat. Genom att införa en WFST ändras en del av själva förfarandet i den aktuella parsealgoritmen. WFST sparar alltså bågar som representationer av de fraser som framgångsrikt lyckats identifieras i parsern. När parsern senare söker efter en fras av speciell kategori som startar på en speciell position i indatasträngen letar den i WFST:n för att se om någon sådan båge finns lagrad. Om det gör det används den för att skapa en möjlig analys av det aktuella segmentet i parsningen. Det är viktigt att alla bågar av en viss kategori med ett viss starttillstånd läggs in i WFST:n samtidigt annars kan man endast få ut en analys av den aktuella parsningen.
Figuren nedan visar resultatet från en parsningen av topdown parser med WFST stöd som hade samma indata som i exempel 2.1.1. Grafen består av ett antal noder och bågar.



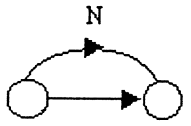
Figur 2.1.2

Bågarnas start och sluttillstånd utläses trivialt genom att notera mellan vilka noder de sträcker sig. Bågens syntaktiska kategori står strax ovanför mitten av bågen den tillhör. Nedan för noderna står de olika token ifrån indatasträngen. För att se vilka token en båge sträcker sig över är det bara att se vilka token som står mellan bågen start och sluttillstånd. I och med att alla framgångsrikt identifierade bågar lagras i WFST:n kan man utläsa de partiella tolkningar som parser hittat. Figur 2.1.2 innehåller båda de tolkningar som parseträden i exempel 2.1.1 visar.
Däremot saknas till viss del den hierarkiska struktur ett parseträd beskriver. Det går inte med säkerhet att se vilka bågar som är underordnade[1] en annan båge. Det beror på att bågarna saknar information om vilken regler som gett upphov till dem.

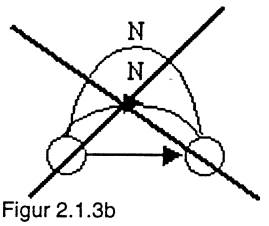
Vill man ändå undersöka den hierarkiska strukturen får man utföra det manuellt med hjälp av grammatiken. Låt oss gå tillbaka till figur 2.2 ovan utan att konsultera grammatiken hade det varit omöjligt för oss att veta vilka bågar som är direkt underordnade bågen <0,3,S>. Eftersom det bra finns en S regel kan vi räkna ut att det är bågarna <0,1,NP> och <1,3,VP> men utan tillgång till grammatiken kunde vi lika gärna antagit att var bågarna <0,1,N> ,<1,2,VP> och <2,3,AD>.
Men eftersom det ofta finns flera regler av en viss kategorier som skulle kunna gett upphov till en båge som skulle kunna sträcka sig över det aktuella området kan det ibland vara svårt för att inte säga omöjligt att manuellt hitta den rätta analysen.
En WFST tillåter inte att det finns någon som helst redundans i grafen. Alltså kan ett förhållande av den typ figur b visar förekomma vilket leder till att grafen inte kan presentera alla alternativ tolkningar

[1] En båge X är underordnad en båge Y om X ingår som en delanalys av Y.

av en text. Det är oturligt eftersom det i många fall är önskvärt att se samtliga analyser av en parsning.
Den eventuella fullsändiga analys parsern hittar är beroende av vilken sök strategi parsern använder.



Figur 2.1.3a



Figur 2.1.3b

En WFST kan förvisso minska söktiden genom att lagra de fraser som parsern lyckats identifiera och den kan presentera delar av de partiella lösningarna. Men den har
en del svaga sidor. Det finns inget som hindrar parsern ifrån att söka längs en felaktigt stig i sökrymden om och om igen i eftersom misslyckade sökningar inte lagras vilket är synd då söktiden förmodligen skulle kunna minskas ännu mer. Bristen på information i bågarna samt svårtydd hierarkiskstruktur samt avsaknaden av möjligheten att presentera alla möjliga tolkningar är andra svagheter som tillsammans gör WFST stödd parsning till en icke tillfredställande metod.

2.1.2 Chart

Som nämnts tidigare finns det både för och nackdelar med att låta en parser använda sig av en WFST men genom att införa modifikationer kan man undvika flera av de problem som uppkommer vid sådan parsning. De ändringar vi gör i WFST:n kommer resultera i en Chart. Chart är det centrala elementet i chartparsing.

Mats Wirén(1992) väljer att definiera en chart på följande sätt:
En chart är en riktad graf C = <V, E> sådan att V är ett finit ,icke tomt antal noder och $E \subseteq V \times V \times R \times Q$ är ett finit antal bågar. R och Q representerar den lingvistiska informationen av en båge. R är ett finit antal punkterade kontextfria regler och Q är ett möjligt antal särdragstrukturer i constraint-based grammar.

Den största skillnaderna mellan en chart och WFST är att bågarna i en chart innehåller mer information och att grafen inte är lika styrd i en char som i en WFST.
En båge i en chart har följande form <i, j, $X_0 \rightarrow Y_0.\phi$, D>.

<i> = Bågens starttillstånd. Ordningsnumret på den nod i grafen från vilken bågen utgår,
<j> = Bågens sluttillstånd. Ordningsnumret på den nod till vilken bågen leder.
<$X_0 \rightarrow Y_0.\phi$ > = En punkterad regel. En komponent som håller reda på vilka av bågens sökta fraser som är funna samt vilka som är kvar att hitta X_0 är bågens kategori. Elementen vänster om pilen är de som bågen kräver för att bli komplett.
<D> = Bågens särdragstruktur.

Elementen i och j är identiska med de som ingår i de bågar som finns i en WFST. Det tredje elementet är en punkterad regel, en komponent som innehåller information om vilken regel i grammatiken som har gett upphov till den aktuella bågen samt den information om vilka andra bågar den behöver finna för att bli komplett. Precis som den regel som givit upphov till bågen består den punkterade regeln av ett höger och ett vänsterled. Vänsterledet innehåller den bågens syntaktiska kategori och högerledet innehåller kategorierna på de andra bågar som behövs för att den aktuella bågen skall bli komplett.
Det som skiljer en punkterad regel från en regel i grammatiken är att den punkterad regeln innehåller en punkt i högerledet. Punkten ger information om vilka av de sökta fraserna som bågen lyckats identifiera. De fraser som står till vänster om punkten har bågen funnit de som står till höger söker den fortfarande efter.

a) S→.NP VP
b) S→ NP.VP

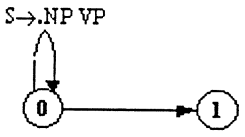
Exempel 2.1.2

I exempel 2.1.2 ser vi två punkterad regler i olika tillstånd . Post a har ännu inte funnit någon av de två sökta fraserna men b har funnit en NP fras.

Genom att införa den punkterade regeln skapas nya möjligheter för parsern. Det går nu inte bara att spara information om fraser som framgångsrikt identifierats utan även de ofullständiga sökningarna sparas. Det leder till att man kan minska söktiden ändå mer än i WFST stödd parsning. I grafen som blir resultatet av en chartparsning kan vi även utläsa den hierarkiska struktur vi tidigare saknat tack vare att vi ser precis vilken regel som gett upphov tillbågen samt exakt i vilket tillstånd bågen befinner sig.

Den sista komponenten i en båge D kan innehålla information om de särdragstrukturer som tillhör bågen. Informationen kan både gälla de särdrag bågen själv har och unifieringsregler. Särdrags komponenten kan se ut på olika sätt men är inte obligatorisk. I kapitel 3 Metod beskriver jag den representation jag valt.

Det är inte bara bågarna som modifieras utan även de riktlinjer grafen har. I motsats till WFST tillåter charten bågar som har samma start och slut nod samt bågar som ännu inte funnit någon av sin sökta fraserna. De sistnämnda kallas för tomma bågar och sådana måste tillåtas om bågarna skall innefatta punkterade regler.



Figur 2.1.4

Figur 2.1.4 visar hur en sådan tombåge ser ut.

Förutom en båge kan vara tom eller inte kan en båge samtidigt vara aktiv eller passiv. En båge som, funnit alla de fraser den behöver för att bli komplett är passiv annars är den aktiv. I WFST tilläts endast passiva bågar i charten däremot måste båda typerna tillåtas för att det skall fungera

Charten liksom WFST:n tillåter ingen redundans men eftersom den punkterade regeln ersätter den tidigare kategorin i WFST-bågarna kommer charten att kunna innehålla flera bågar som leder mellan samma noder så länge som de är skiljer sig i minst ett avseende. Det senare leder till att charten kan presentera alla möjliga tolkningar av en parsning.

2.1.3 Fördelarna med chartparsing

Jag tänkte sammanfatta de fördelar med chartparsing som jag presenterat ovan samt introducera några jag tidigare inte nämnt:

- Sökningstiden kan minskas radikalt genom att parsern sparar alla de fraser man försökt identifiera tidigare i den aktuella analysen. Både de fraser som parsern framgångsrikt lyckats identifiera och de som inte lyckats identifieras sparas vilket leder till att parsern slipper söka efter dem om igen i samma parsning.
- Givet en grammatik och lexikon kan en parser presentera alla möjliga analyser av en textmassa fullständiga såväl som partiella.
- Grafen som blir resultatet av en parsning med hjälp av en chart visar den hierarkiska struktur man finner i parseträd. Det går att uttyda de samband som råder mellan grammatiken och den aktuella analysen då bågarna bl. a. innehåller information om vilken regel som givit upphov till den.
- Chartens bågar har om man så önskar möjligheten att uttrycka information om särdragstruktur och unifieringsregler.
- Eftersom en chartparser inte styrs av att parsning måste ske i någon speciell riktning fungerar det bra att använda den i applikationer som exempelvis textredigeringsprogram.

2.2 Java

Programspråket Java har i takt med Internets utbredning ökat dramatiskt i popularitet. Jag tror jag vågar påstå att de är det hetaste programspråket på marknaden just nu, även om det i många avseenden ännu inte är det effektivaste. Följande aspekter tror jag hjälpt Java att vinna denna popularitet:

- Språket är objektorienterat.
- Det går att köra program skrivna i Java i en webbläsaren över Internet.
- Java är tänkt att vara plattformsoberoende, det vill säga programmet skall fungera i valfritt operativsystem oberoende av i vilken miljö det är utvecklat.
- Programspråket har en liten kärna vilket gör det förhållande vis lätt att lära sig[2].
- Kompilatorn och många paket[3] är gratis.
- Det finns stöd för gränssnittutveckling.

[2] I jämförelse med exempelvis C++.

[3] Paket är samlingar av Java klasser. Se Campione & Walrath(1998)

2.2.1 Objektorientering

Själva begreppet objektorientering innebär att man centrerar utvecklingen av ett system runt olika fenomen i datasystemets omgivning istället för de funktioner och metoder man vill att systemet skall innehålla. En fördel med att använda sig av ett objektorienterat tillvägagångssätt är att man som utvecklare får en god överblick över de olika delarna i systemet samt att det underlättar uppdatering av delar av systemet.

Man ska akta sig för att blanda ihop begreppen objektorienterad analys och design med objektorienterade programspråk. Objektorienterad analys och design är en metod att arbeta fram den struktur och funktonalitet det aktuella systemet eller programmet skall ha. Objektorienterade programspråk är strukturerade kring objekt som är knutna till olika attribut och funktonalitet.

Analys och design[4] processen sker helt avskilt från implementeringen men tanken är att de dokument som blir resultatet av dessa processer skall ligga till grund för implementeringen. Har analys och design delen genomförts väl skall det vara lätt för programmerarna att dela upp arbetet emellan sig och översätta de olika komponenterna i design och analys dokumentet till komponenter i ett objektorienterat språk. Det är dock fullt möjligt att implementera system och program i objektorienterade språk utan att först ha genomfört en objektorienterad analys och design.
Ett program implementerat i ett objektorienterat programspråk blir inte automatiskt ett objektorienterat program det är fullt möjligt att exempelvis skriva imperativa program i Java.

Ett bra sätt att förklara vad ett objektorienterat programspråk är att beskriva några av de centrala komponenterna:

- Objekt
- Klass
- Meddelande
- Arv

2.2.2 Objekt

Definition av Objekt: *Ett objekt är en knippe mjukvara som består av variabler och relaterade metoder*
Campione & Walrath (1998)

Man skulle kunna jämföra objekt i ett programspråk med instanser av entiteter i det verkliga livet. Det som karaktäriserar dessa objekt är att samtliga innehåller diverse attribut och ofta flertal olika beteenden. Attribut är fenomen som beskriver objektet i fråga och kan skilja ut det från andra objekt av samma typ. Beteenden är de aktioner som objektet kan utföra. Ofta när man skriver program vill man på ett eller annat sätt simulera verkliga företeelser. Objekt är ett sätt att objektorienterat representera dessa företeelser. Man låter objekten inneha en mängd metoder istället för att bygga programmet runt en mängd funktioner som utför något händelse med ett speciellt objekt. Givetvis kan en metod i ett objektorienterat språk ta parametrar som indata.
Som exempel skulle man kunna ta en personbil, den har attribut som färg, aktuell hastighet, antal hjul, model osv. och beteenden som att svänga, stoppa, ändra växel osv.

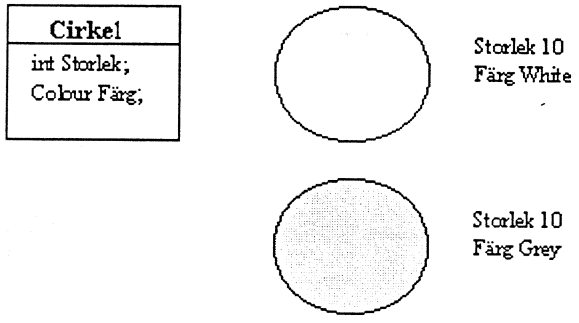
2.2.3 Klass

Definition av Klass: *En klass är en ritning eller prototyp som definierar de variabler och metoder som är gemensamma för alla objekt av en speciell typ*
Campione & Walrath (1998)

Om objekt i programspråk är reflektioner av instanser av entiteter i världen skulle man kunna beskriva en klass som en komponent som beskriver det som alla objekt av en speciell typ har gemensamt. När man utvecklar ett program är man ofta intresserad av att behandla flera komponenter av samma typ då kan det vara smidigt att definierar en mall som beskriver hur en komponent av den aktuella typen är uppbyggd. Det är det en klass gör. Den definierar vilka variabler och vilken funktonalitet objekt som är instanser av den aktuella klassen har.
Varje instans av klassen kan givetvis ha unika värden på men strukturen är identisk.

---

[4] För mer information se Madsen et al(1998)



Exempel 2.2.1

Exempel 2.2.1 visar två objekt av den enkla klassen Cirkel. Klassen har endast två attribut, en integer med cirkelns diameter och en variabel med cirkelns fyllnadsfärg. De två objekten har samma storlek men har olika värde på variabeln Färg. Den ena är vit den andra är grå.
De vore ju helt onödigt att skapa två separata klasser för de två cirklarna då de endast skiljer sig i ett enda avseende. Klassen Cirkel i det aktuella exemplet är tämligen enkel. En mer komplicerade klass hade motiverat valet av en samlingsklass ändå mer.

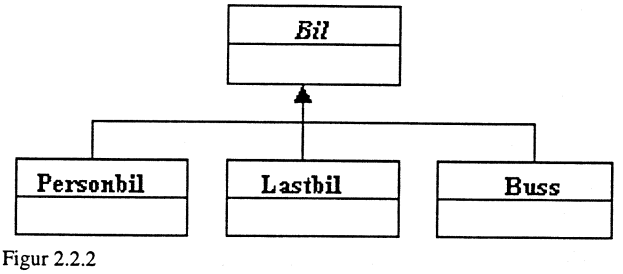
Men det finns andra skäl till att använda sig av klasser exempelvis arv och polymorfism[5].Klasser i Java är mycket kraftfulla och i princip allt definieras i termer av klasser; hela program, mer eller mindre komplexa strukturer, arkiv med metoder osv. Det gör att språket är väldigt dynamiskt.

2.2.4 Meddelande

Meddelande är det sätt objekt kommunicerar med varandra i Java. De objekt som kommunicerar med varandra kan vara instanser av en och samma klass eller av olika klasser. Meddelanden representeras av metoder i Java. Alla metoder är sammankopplade med en klass. Metoderna definieras då man definierar den aktuella klassen. För att anropa en metod från en klass måste för det mesta[6] en instans av de aktuella klassen skapas. En instans av en klass kan endast anropa de metoder som den har definierat i klassens kropp. Vid första anblick kan det tyckas som en begränsning men i själva verket tycker jag det är ett utmärkt sätt att kontrollera var, vad och när aktioner utförs i ens program. Tack vare att man definierar vilka metoder en klass får anropa begränsar det riskerna att en metod anropas på ett felaktigt sätt. När det gäller vidareutveckling av ett system eller program kan det vara till mycket står nytta för utvecklaren då han/hon på ett relativt översiktligt sätt få en uppfattning av vilken funktonalitet som tidigare implementerats i den aktuella klassen.

2.2.5 Arv

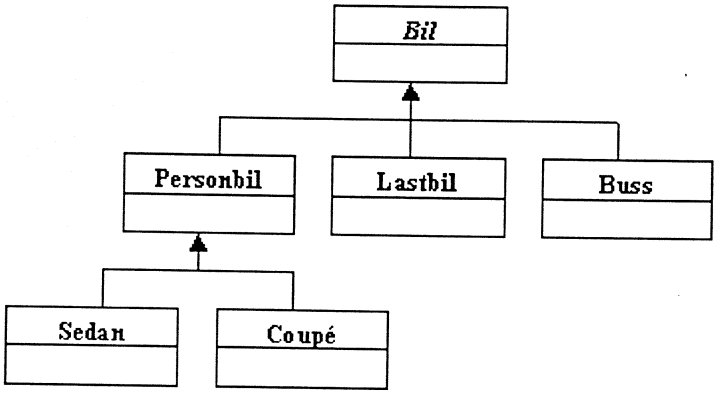
När man skall utveckla ett system är det inte ovanligt att man skall modellera med ett flertal olika komponenter som liknar varandra i olika grad. Har de olika komponenterna identisk struktur representerar man lämpligen detta med att skapa en gemensam klass från vilken de olika instanserna av klass utgår. I de fall komponenterna har stora likheter men inte identisk i struktur kan man inte lösa problemet på det sättet. Det kan också vara på det sättet att man som utvecklare vill ha de olika komponenterna i separata klasser samtidigt vill man gärna ta tillvara på de likheter de olika komponenterna har. I Java kan man lösa det genom något som kallas arvstruktur. En arvstruktur består av minst två klasser en superklass och en subklass. Subklassen ärver alla attribut och metoder som ingår i superklassen men det är också möjligt för subklassen att definierar metoder och attribut så det passar in på det fenomen man vill modelera. Subklassen kan även utöka superklassen med fler metoder och attribut annars hade det inte varit så stor idé med arv
Jag tänkte illustrera arv med exempel om bilar. Det finns många typer av bilar personbil, minibuss, lastbil osv. de skiljer sig åt på många sätt men de har alla speciella kriterier vilka gör att man kategoriserar dem som en bil. Samtliga har ratt, motor, växelåda, hjul, dörrar de kan köra ,svänga, stanna osv. men alla typer fordon är olika i flera avseende de kan ha olika vikt, storlek på motorn , antal säten, antal hjul,



Figur 2.2.2

[5] För mer information se Campione & Walrath (1998).
[6] Undantag för statiska metoder se Campione & Walrath(1998).

listan kan göras otroligt lång. Det finns alltså goda skäl för mig som utvecklare att representera dem som separata klasser men jag vill ju ta vara på de likheter som finns. Jag löser det genom att skapa en abstrakt klass Bil, att en klass är abstrakt innebär att man inte kan skapa en instans av den klassen. Därefter skapar man en klass för varje biltyp som skall ingå i systemet. I Figuren ovan finns klasser Personbil ,Lastbil och Buss, men man skulle kunna gjort uppdelningen mer ingående om man så önskar.



Figur 2.2.3

Det är också fullt möjligt för en subklass att vara superklass åt en annan klass. En klass kan däremot endast ärva från högst en annan klass i Java i motsats till exempelvis C++. Vill man att en klass skall få attribut eller metoder från fler än en klass får man använda sig av interface[7]. Arv är mycket användbart både när man skall utöka ett befintligt system eller skapa ett nytt, det är även mycket användbart då man kan tillämpa polymorfism[8]

2.2.6 Java Historik

Programspråket Java utvecklades på företaget Sun Microsystems Inc. i Kalifornien.
James Gosling anses vara skaparen av Java, och enligt honom kan man spåra utvecklingen av programspråket till en liten grupp på Sun Microsystems Inc som redan 1991 diskuterade möjligheterna med det ännu ej utvecklade world wide web. Gosling startade med att utöka C++ kompilatorn men så småningom kom han på att det inte skulle vara nog.
Vid år 1994 hade World Wide Web blivit stort och James Gosnings och hans kompanjoner insåg att de med hjälp av Java hade möjligheten att bygga häftig webbläsare. I början av hösten var webbläsaren Hotjavas föregångare WebRunner färdigutvecklad.
Java skapade helt nya möjligheter för webbutvecklare som tidigare hade varit tvungna att förlita sig på html och cgi, inte minst när det gäller komplicerade grafiska element. En stor tyngdpunkt i Java är det stöd som finns behandling av grafiska element. Eftersom programspråket i viss mån är plattformsoberoende kan man utveckla relativt komplicerade program som kan köras av användare med olika system, att programspråket inte riktigt är så plattformsoberoende som en del påstår är en annan historia.

2.4 Tidigare Arbete

2.4.1 Studies in incremental Natural Language Analysis

I sin doktorsavhandling "Studies in Incremental Natural Language Analysis" utforskar Mats Wirén problemet med "Incremental analysis of natural language text". Incremental Parsing innebär att parsa en text token för token snarare än en text massa i taget. Det ger enligt Mats Wirén fördelen att parsern kan fungera effektivt i interaktiva analyser såsom i textredigerare. Vidare presenterar Wirén en algoritm han konstruerat som analyserar arbiträra ändringar i texter. Algoritmen bygger på chartparser algoritmen och tillåter att man använder sig av olika kontrollstrategier och grammtikformalismer. Tanken är att parsern genom att hålla reda på samband mellan olika delanalyser av den aktuella texten skall det vara möjligt att upptäcka ändringar i textmassa. För att testa algoritmen har flera experimentiella implementationen gjorts Wirén själv anser att den bästa är LIPS 2.0 skriven av Ulf Dahlén i programspråket Xerox Common Lisp. Wirén presenterar en mängd olika parsestrategier som tidigare utvecklats, en delmängd av dem jag har valt att implementera. Parsestrategierna var bl. a. topdown, bottomup, bottomup à la Kilbury, topdown with bottomup filter, bottomup med topdownfilter, bottomup á la Kilbury med topdownfilter, bottomup á la Kilbury med bottomupfilter, bottomup á la Kilbury med topdown & bottomupfilter.

---

[7] För mer information se Campione & Walrath (1998).
[8] För mer information se Campione & Walrath (1998).

3. Metod

Den uppgift jag valt att lösa har en praktisk snarare en teoretisk riktning vilket givetvis präglat arbetsgången. Den allra största delen av arbetstiden har gått åt till implementationen av programmet jag skulle konstruera. Givetvis har arbetet också haft teoretisk inslag men endast i egenskap av att göra det praktiska arbetet möjligt. Det teoretiska arbetet har framförallt rört de algoritmer som skulle implementeras.

3.1 Strategierna

Eftersom det program jag skulle konstruera centrerades kring en mängd chartparsealgoritmer verkade det vara en god idé att inleda projektet med att fräscha upp mina kunskaper inom området och välja ut de strategier som skulle implementeras. I sin doktorsavhandling Mats Wirén(1992) olika chartparsestrategier, från vilka jag valde ut åtta strategier som mitt program skulle demonstrera. De åtta strategierna var följande:

- Topdown
- Bottomup
- Bottomup á la Kilbury
- Topdown med bottomupfilter
- Bottomup med topdownfilter
- Bottomup á la Kilbury med topdownfilter
- Bottomup á la Kilbury med bottomupfilter
- Bottomup á la Kilbury med bottomup & topdownfilter

Samtliga algoritmer kommer att beskrivas nedan, jag har valt att presentera dem med något modifierade versioner av de definitioner som Wirén (1992) använde. Det som framförallt skiljer våra beskrivningar åt är angreppssätt beträffande särdragstrukturen.

En båge representeras av en tuppel $<v_i,v_j,X_0\rightarrow.\alpha,D>$ där:

i = Bågens starttillstånd. Ordningsnumret på den nod i grafen från vilken bågen utgår.

j = Bågens sluttillstånd. Ordningsnumret på den nod till vilken bågen leder.

$X_0\rightarrow Y_0.\phi$ = En punkterad[9] regel.X_0 är bågens kategori. En komponent som håller reda på vilka av bågens sökta fraser som är funna samt vilka som är kvar att hitta. Observera att ingen information om bågens särdrag lagras i det här elementet.

D = Bågens särdragstruktur innehåller dels en mängd poster med särdragsvärden D:s dels en mängd särdragsposter D:u.

Ett särdragsvärde har formen $<k,Q_0,x>$.

<k> = Ett heltal som korresponderar till den båge från vilken särdraget härstammar. Siffran 0 betyder att den härstammar från sig själv 1 den första begärda bågen osv.

<Q> = Namnet på särdraget.

<x> = Särdragets värde.

En båge skulle exempelvis kunna ha ett särdragsvärde som ser ut på följande sätt <0,num,sing>.

En båge kan dels lagra de egna särdragsvärden dels de särdrag som förvärvats då sökta bågar identifierats. Men det är endast särdags där k = 0 i den nyligen identifierade bågen som förs vidare till den som nya bågen. Om inte den triggering bågen innehåller särdragsposter som säger något annat. Se exempel 3.1.1 nedan.

En särdragspost har formen $<m:Z_0=n:W_0.>$.

m = Ett heltal som representerar den första av de fraser aktionen gäller.

n = Ett heltal som representerar den andra av de fraser aktionen gäller.

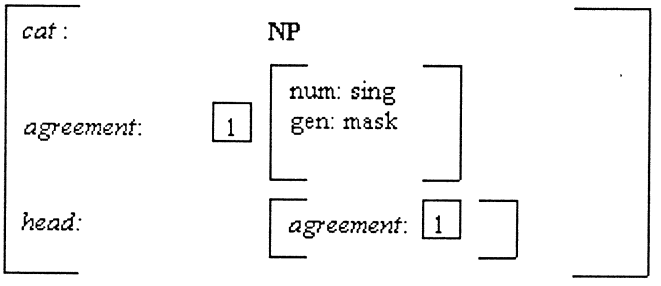
Z_0 = Det särdrag i den första frasen i särdragsposten som är av intresse vid den aktuella unifieringen.

---

[9] Se kapitel 2 Bakgrund.

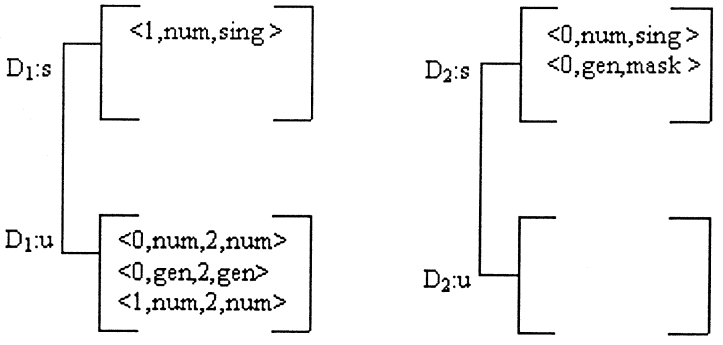
W_0 = Det särdrag i den andra frasen i särdragsposten som är av intresse vid den aktuella unifieringen.

Ett exempel på en särdagspost skulle kunna se ut på följande sätt <1:num=2:num>. Observera jag har valt att konstruera bågarna på ett sådant sätt att de inte kan behandla så kallade rekursiva särdrag . En rekursiv särdragsstruktur kan exempelvis se ut på följande sätt:

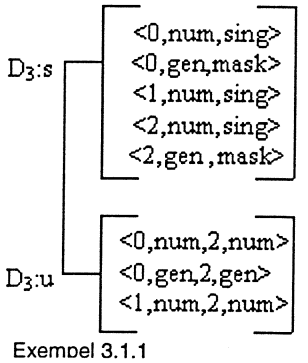


En särdragspost kan fylla två olika funktioner men endast en per post. Antingen fyller den funktionen av en unifieringsrestriktion eller funktionen tilldelning av särdragsvärde.
Det är endast i de fall där m = 0 som en särdragspost fungerar som en tilldelning.
Tilldelningen sker genom att ett nytt särdragsvärde $<0,Z_0,x_0>$ adderas till den aktuella bågen. Där x_0 är värdet av särdraget W_0 från den m:te båge som hjälp till att expandera den aktuella bågen. Se exempel 3.1.1 nedan.
Unifierings restriktionerna bestämmer de särdragsvillkor som måste vara uppfyllda för att en passiv båge skall kunna hjälpa till att expandera en triggering aktiv båge. Det en post säger är att värdet av särdraget Z_0 i den n:te båge som hjälpt till att expandera den aktuella bågen är lika med värdet av särdraget W_0 i den m:te båge för att unifiering skall kunna äga rum. Skulle någon av de aktuella bågarna sakna sitt respektive särdrag lyckas unifieringen också.

Anta bågen $<v_0,v_1$, Np→Det.N,$D_1>$ och bågen $<v_1,v_2$,N→X_0.,$D_2>$ anta att deras respektive särdragsvärden skulle ha följande struktur.



Bågen Np undersöker om bågen märkt N kan användas för att expandera den. Vi antar att alla kriterier utan just särdragsunifieringen har undersökts med framgång. Det finns endast en särdagspost att ta hänsyn till vid unifieringen av särdrag i det aktuella fallet. Posten<1,num,2,num>. Det är trivialt att se att vilkoret är uppfyllt och en ny båge $<v_0$,v2, Np→Det,N.,$D_3>$ kan skapas. Bågens särdragstruktur beskrivs i figuren till höger. Den nya bågen är en utökade version av den första Np bågen. Flera nya särdragsvärden har lagts till genom unifieringen särdragsvärdena <0,num,sing>, <0,gen,mask > ,<2,,num,sing>, <2,gen,mask >. De två första erhölls särdragsposterna <0,num,2,num> samt <0,gen,2,gen>.



Exempel 3.1.1

Det två andra posterna erhölls automatisk från då bågen $<v_1,v_2,N\rightarrow X_0.,D_2>$ hjälpte till att expandera bågen $<v_0,v_1, Np\rightarrow Det.N,D_1>$.

Andra termer och tecken som används i beskrivningarna är:
Metod = Information om hur parsestrategin initieras och om hur länge strategins huvudloop fortgår.
Sökning = Det sätt strategin hanterar orden i indatasträngen
Förutse = Det sätt strategin skapar nya aktiva bågar
Kombinera = Det sätt strategin försöker expandera existerande aktiva bågar med existerande inaktiva bågar.

$\subseteq$ = Subsummering, en båge subsummerar en annan om de tre första elementen har identiska värden och det fjärde elementet i den första bågen innehåller en delmängd av den andra bågens.

För varje parsestrategi finns även ett exempel där läsaren konkret kan se vilka bågar som skapas vid användande av de olika strategierna. Indata till samtliga exempel återfinns under avsnitt 5.1 ChartDeluxe v1.0.

3.1.1 Topdown (Wirén 1992 sidan 18)

Metod: Lägg till en båge $<v_1,v_1,X_0\rightarrow.\alpha>$ för varje regel $<X_0\rightarrow\alpha,D>$ sådan att X_0 = S där S är grammatikens start kategori. Upprepa följande steg för varje båge v_j sådan att j = 1,.....n+1 tills inga flerbågar kan läggas på charten.

Sökning: Om t_j = a, lägg då för varje lexikal ingång av formen $<X_0\rightarrow a, D>$ till en båge $<v_j,v_{j+1},X_0\rightarrow a., D>$

Mer informellt betyder det att man lägger till en inaktiv båge för varje tolkning av en aktuella token.

Förutse (Topdown): För varje båge formen $<v_i,v_j,X_0\rightarrow\alpha.X_m\beta,D>$ och varje regel av formen $<Y_0\rightarrow\gamma,E>$ sådan att $Y_{0=}$ X_m lägg till en båge $<v_j,v_j,Y_0\rightarrow.\gamma,E>$ såvida den inte subsummeras av en redan existerande båge.

Mer informellt betyder det att man lägger till en båge för varje regel vars första vänsterledkategori matchar den sökta kategorin på den aktiva utlösande bågen, såvida den inte subsummeras av en redan existerande båge.

Kombinera: För varje båge av formen $<v_i,v_j,X_0\rightarrow\alpha.X_m\beta,D>$ och varje båge av formen $<v_j,v_k,Y_0\rightarrow\gamma.,E>$ där $X_m = Y_0$, lägg till en båge $<v_i,v_k,X_0\rightarrow\alpha X_m.\ \beta,D:E(Y_0)>$ om unifieringen är framgångsrik och den aktuella bågen inte subsummeras av en redan existerande båge.

Principen med topdownparsning är att söka konstituenter vars typ matchar högerledet på grammatik regler. Man utgår i från de regler vars vänsterled består av grammatikens startsymbol. Topdown strategin är egentligen inte en rakt igen topdown eftersom själva sökningen är en bottomup operation.

Exempel 3.1.1
Antal bågar: 39

Som ni ser i grafen (Bilaga 2 2.1) så initieras charten först genom att alla regler som är nåbara från grammatikens startsymbol S skapas därefter söker man i indata för att hitta en båge som kan en expandera eller flera av dessa. Varje gång en aktiv båge poppas från agendan läggs de eventuella tomma aktiva bågar som skapas på charten om de är nåbara från den poppade bågens första begärda element dessutom ser man efter om någon tidigare skapad båge kan expandera poppade bågen. Går inte det söker man i indata för att finna någon som gör det.

3.1.2 Bottomup (Wirén 1992 sidan 18)

Metod: Upprepa följande steg för varje båge sådan att j =1,.....n+1 tills inga flerbågar kan läggas på charten.

Sökning: Som i Topdown

Förutse (Bottomup): För varje båge av formen $<v_j,v_k,X_0 \rightarrow \alpha.,D>$ och varje regel av formen $<Y_0 \rightarrow Y_1\beta,E>$ sådant att $Y_1 = X_0$ lägg till en båge $<v_j,v_j,Y_0 \rightarrow .Y_1\beta,E>$ på charten såvida den inte subsummeras av en redan existerande båge.

Mer informellt betyder det att man lägger till en båge för varje regel vars första högerledkategori matchar kategorin på den utlösande inaktiva bågen såvida den inte subsummeras av en redan existerande båge.

Kombinera: Som i TopDown

Principen med bottomupparsning är att reducera konstituenter vars typ matchar högerledsidan av en grammatikregel till en konstituent av typen för vänsterledet av regeln med start vid indatasträngens tokens. Notera att ingen initialisering av charten behövs i det här fallet.

Exempel 3.1.2
Antal bågar: 39

Som ni ser i grafen (Bilaga 2 2.2) börjar strategin att söka i indata för att hitta bågar i från vilka aktiva tomma bågar kan skapas. Varje gången en passiv båge poppas ifrån charten skapas en tom aktiv båge för varje regel i grammatiken vars första högerledselement matchar kategorin på den poppade bågens första begärda element. Skulle bågen istället vara aktiv ser den efter om bågen kan expanderas med hjälp av tidigare skapade inaktiva bågar går inte det sök i indata.

3.1.3 Bottomup à la Kilbury (Wirén 1992 sidan 28)

Metod: Som i Bottomup.

Sökning: Som i Topdown

Förutse 2(Kilbury): För varje båge av formen $<v_j,v_k,X_0 \rightarrow \alpha.,D>$ och varje regel av formen $<Y_0 \rightarrow Y_1\beta,E>$ sådan att $Y_1 = X_0$ lägg till en båge på charten $<v_j,v_k,Y_0 \rightarrow Y_1.\beta,E:D(X_0)>$.

Mer informellt betyder det lägger till en båge för varje regel vars första högerledskategori matchar på den utlösande inaktiva bågen på ett sådant sätt att den nya bågen är omedelbart kombinerad med den utlösande bågen.

Kombinera: Som i TopDown

Kilburys strategi kan summeras som följande: istället för att lägga till tomma aktiva bågar skapas bågar vilka omedelbart kombineras med den utlösande inaktiva bågen. En ny båge kan antingen aktiv eller inaktiv beroende på innehållet i den inaktiva bågen samt på den vad den nya bågens söker efter. Fördelen med den här strategin är att den sparar bågar eftersom den direkt expanderar de nya bågarna. Dessutom behövs ingen redundanskontroll eftersom den utlösande bågen garanterat är unik kommer. En möjlig nackdel med strategin är at den reducerar flexibilitet i sökning. eftersom loopande aktiva bågar inte generas.

Exempel 3.1.2
Antal bågar: 31

Tack vare att den här strategin direkt expanderar de eventuella nya bågar som skapas vid poppande av en passiv båge lyckas man reducera antalet bågar gentemot strategi 3.1.2 med 7 st. i det aktuella fallet(se bilaga 2 2.3).

3.1.4 Topdown med bottomupfilter (Wirén 1992 sidan 27)

Metod: Upprepa följande steg för varje **vertex** v_j sådan att j =1,.....n+1 tills inga flerbågar kan läggas på charten.

Sökning: Som i Topdown

Förutse :För varje båge formen $<v_i,v_j,X_0 \rightarrow \alpha.X_m\beta,D>$ och varje regel av formen $<Y_0 \rightarrow Y_1\gamma,E>$ sådan att$Y_0 = X_m$ lägg till en båge $<v_j,v_j,Y_0 \rightarrow .Y_1\gamma,E>$ om $R^*(Y1,T)$ gäller för någon kategori T av token t_j såvida den inte subsummeras av en redan existerande båge.

Mer informellt betyder det att vi lägger till bågar för varje regel vars först vänsterledskategori matchar den första sökta kategorin i den utlösande aktiva bågen såvida dess första begärda konstituent går att nå från någon kategori av näst token i indatastränge såvida den inte subsummeras av en redan existerande båge.

Kombinera: Som i TopDown

I sin grundläggande form, kräver den här strategin en speciell sök strategi och/eller sök ordning, eftersom token t_j måste ha blivit funnen varje ett förutseende skall utföras av bågen v_j. Men det går att använda ordningsoberoende sökning om man för så kallade sleeping edges[10].

Exempel 3.1.4
Antal bågar: 26

Genom att bottomup filtret endast tillåter att de aktiva bågar som har någon möjlighet att nå nästa token i indata läggs till chart kan antalet bågar i charten i det aktuella fallet minska med 13 st. i det aktuella fallet (se bilaga 2 2.4).

3.1.5 Bottomup med topdownfilter(Wirén 1992 sidan 27)

Metod: Som i Bottomup

Sökning: Som i Topdown

Förutse (Bottomup med topdownfilter): För varje båge av formen $<v_j,v_k,X_0 \rightarrow \alpha.,D>$ och varje regel av formen $<Y_0 \rightarrow Y_1\beta,E>$ sådant att $Y_1 = X_0$ lägg till en båge på charten $<v_j,v_j,Y_0 \rightarrow .Y_1\beta,E>$ om det finns en båge av formen$<v_i,v_j,Z_0 \rightarrow \gamma.Z_m\delta,F>$ sådan att $R^*(Z_m,Y_0)$ och såvida den inte subsummeras av en redan existerande båge.

Mer informellt betyder det att man förutser en båge för varje regel var första högerledkategori matchar kategorin på den utlösande inaktiva bågen, om kategorin på den nya bågen går att nå från den första sökta kategorin i någon aktiv båge och såvida den inte subsummeras av en redan existerande båge.

Kombinera: Som i TopDown

Den här strategi behöver en topdowninitiering precis som i vanlig topdownparsning för att bågar som startar vid första noden inte skall sorteras bort.

---

[10] Se Wirén 1992 sidan 27 för mer information

Exempel 3.1.5
Antal bågar: 35

Genom att lägga till ett filter som kontrollerar att de eventuella nya bågar som skapas vid poppande av en inaktivbåge läggs endast på charten om de kan nås från någon tidigare aktiv båge hindras icke expanderbara bågar från att läggas på charten vilket kan reducera antalet bågar jämfört med "vanlig" bottomup parsning men eftersom strategin kräver topdown initiering till kommer också endel bågar som inte bottomup annars skapar (se bilaga 2 2.5).

3.1.6 Bottomup á la Kilbury med bottomupfilter (Wirén 1992 sidan 29)

Metod: Som i Bottomup

Sökning: Som i Topdown

Förutse (Kilbury med bottomupfilter)**:** För varje båge av formen $<v_j,v_k,X_0 \rightarrow \alpha.,D>$ och varje regel av formen $<Y_0 \rightarrow Y_1Y_2\beta,E>$ sådant att $Y_1 = X_0$ lägg till en båge på charten $<v_j,v_k,Y_0 \rightarrow .Y_1.Y_2\beta,E>$ om $Y_2\beta$ är tom eller $R^*(Y_2,T)$ gäller för någon kategori av token t_j. såvida den inte subsummeras av en redan existerande båge.

Mer informellt betyder det att man lägget till en båge för varje regel vars första högerledskategori matchar kategorin av den utlösande inaktiva bågen på ett sådant sätt att den nya bågen omedelbart kombineras med den utlösande bågen. Men om den nya bågen är aktiv läggs den endast till om dess första begärda konstituent kan nås från någon kategori i nästa token.

Kombinera: Som i TopDown

Exempel 3.1.6
Antal bågar: 22

Minst antal bågar i det aktuella exemplet fick den här strategin, Kombination av Kilbury och ett filter som ser till att endast de bågar som går att nå från tidigare skapade bågar läggs till charten då en aktiv båge poppas från agendan (se bilaga 2 2.3).

3.1.7 Bottomup á la Kilbury med topdownfilter (Wirén 1992 sidan 29)

Metod: Som i Bottomup

Sökning: Som i Topdown

Förutse (Kilbury med topdownfilter): För varje båge av formen $<v_j,v_k,X_0 \rightarrow \alpha.,D>$ och varje regel av formen $<Y_0 \rightarrow Y_1\beta,E>$ sådant att $Y_1 = X_0$ lägg till en båge på charten $<v_j,v_k,Y_0 \rightarrow Y_1.\beta,E:D(X_0)>$ om den finns en båge av formen $<v_j,v_j,Z_0 \rightarrow \gamma.Z_m\delta>$ sådant att $R^*(Z_m,Y_0)$.

Mer informellt betyder det att man lägger till en båge för varje regel vars första högerledskategori matchar kategorin av den utlösande inaktiva bågen på ett sådant sätt att den nya bågen omedelbart kombineras med den utlösande bågen såvida kategorin av den nya bågen går att nå från någon inkommande aktiv båge.

Kombinera: Som i TopDown

Den här strategin kräver en topdowninitiering liksom i vanlig topdown för att inte för att inte bågar som startar vid grafens första nod skall filtreras bort.

Exempel 3.1.6
Antal bågar: 28

Kombinationen av Kilbury och ett topdownfilter gör att ett antalet bågar kan reduceras med 11 st. jämfört med vanlig bottomup men det faktum att strategin kräver en topdowninitiering drar upp antalet bågar något (se bilaga 2 2.6).

3.1.8 Bottomup á la Kilbury med bottomup & topdownfilter (Wirén 1992 sidan 30)

Metod: Som i Bottomup

Sökning: Som i Topdown

Förutse (Bottomup): För varje båge av formen $<v_j,v_k,X_0 \rightarrow \alpha.,D>$ och varje regel av formen $<Y_0 \rightarrow Y_1Y_2\beta,E>$ sådant att $Y_1 = X_0$ lägg till en båge på charten $<v_j,v_k,Y_0 \rightarrow Y_1.Y_2\beta,E:D(X_0)>$ om det finns en båge av formen $<v_j,v_j,Z_0 \rightarrow \gamma.Z_m\delta,F>$ sådant att $R^*(Z_m,Y_0)$ och om $Y_2\beta$ är tom eller $R^*(Y_2,T)$ gäller för någon kategori T av token t_j såvida den inte subsummeras av en redan existerande båge.

Mer informellt betyder det att man lägger till en båge för varje regel vars första högerledskategori matchar kategorin av den utlösande inaktiva bågen på ett sådant sätt att den nya bågen omedelbart kombineras med den utlösande bågen såvida kategorin av den nya bågen går att nå från någon inkommande aktiv båge. Men om den nya bågen är aktiv läggs den endast till om dess första begärda konstituent kan nås från någon kategori i nästa token.

Kombinera: Som i Topdown

Den här strategin kräver en topdowninitiering liksom i vanlig topdown för att inte för att inte bågar som startar vid grafens första nod skall filtreras bort.

Exempel 3.1.6
Antal bågar: 26

Genom att kombinera både topdown och bottomupfilter med kilbury blir antalet bågar i det aktuella fallet till 26 st. Det är alltså två mindre än strategi 3.1.7 men på grund av topdowninitieringen, fyra bågar mer en strategin som kombinerar bottomdownfilter med Kilbury (se bilaga 2 2.7).

Totalt alltså 8 st. strategier och bestämde mig för att samtliga skulle kunna använda både djupet och bredden först sökning. Som synes är många strategier mycket lika varandra men olikheter gör ändock såpass storskillnad att det är värt att inkludera samtliga i programmet.

3.2 Programspråk

I exjobbsspecifikationen stod det uttryckligen att programspråket Java skulle användas vid implementering av programmet vilket gjorde att jag slapp ta ställning till val av verktyg. Jag saknade helt erfarenhet av objektorienterade programspråk och därför föll det sig naturligt att jag på ett tidigt stadium bekantade mig med det. Koncepten att dela upp program och system i olika komponenter genomsyrar det objektorienterade tänkandet. Min nästa uppgift var att skapa en modell som innehöll de komponenter av vilka programmet skulle bestå.

3.3 Modell

Det nästa steget i programutvecklingen var att försöka identifiera de olika komponenter som programmet skulle bestå av. Jag gjorde tidigt valet att gränssnittet skulle vara separerat från programmets andra beståndsdelar och valde att vänta med utvecklingen av det elementet. Istället koncentrerade jag arbetet på de delar som var direkt relevanta för själva parsningen. Jag utgick från de delar som ingår i beskrivningen av en chartparser.
Modellen inkluderade sext element. Nedan beskriver jag kortfattade den struktur och den funktonalitet jag ville att de olika elementen skulle innehålla.

- **Regler**

En regel skulle ha formen $<X_0 \rightarrow Y_0\ \beta, D>$
En regel måste bestå av ett vänsterled ,minst ett element i högerledet samt en struktur som beskriver en regels eventuella unifieringsrestriktioner-.
$<X_0>$ = Regelns vänsterled.
$<Y_0>$ = Regelns högerleds första element β är andra eventuella element i högerledet.
<D> = Den struktur som innehåller särdragsunifieringsrestriktioner.

Metoder som rör manipulering av regler kommer också att representeras i den aktuella klassen.

- **Grammatikmodul**

En grammatikmodul är en struktur som innehåller en ändlig mängd regler samt diverse metoder för administration av dessa.

- **Ord**

Ett ord skulle ha följande struktur $<X_0,S,D>$.
$<X_0>$ = Ordets syntaktiska kategori.
<S> = Ordets syntax.
<D> = De eventuella särdrag och värden som är associerat med det aktuella ordet. Den klass som representerar ord kommer även innehålla metoder som rör manipulering av ord.

- **Lexikonmodul**

En lexikonmodul är en struktur som innehåller en ändlig mängd lexikala ingångar samt diverse metoder för administration av dessa.

- **Båge**

En båge har formen $<i, j, X_0 \rightarrow Y_0.\phi, D>$
Se sid. 13 för mer information.

Den klass som representerar bågen skall också innehålla diverse metoder som manipulering av bågar samt diverse metoder som är rent parserelaterade.

- **Chart**

Charten skulle representeras av en mängd bågar.

3.3 Implementering

Själva implementering av programmet började med att jag skapade avskalade representationer av komponentmodellens olika element för att senare successivt utöka dem. Redan tidigt hade jag en uppfattning om hur jag ville att gränssnittet skulle se ut men det var inte förrän de andra komponenterna var klara jag började utvecklade det. När det gäller utvecklingen av gränssnittet har jag använt mig av mitt eget omdöme och erfarenheter samt i många fall konventionella strukturer för att konstruera ett användarvänligt gränssnitt.

3.4 Testning

De tester som gjorts under utvecklingen har jag uteslutande utfört själv. De har framförallt genomförts i syfte att kontrollera att de algoritmer och klasser som implementerats betett sig som det var avsett. Eftersom gränssnittet var den sista delen av programmet som utvecklades skapade jag ett testprogram med ett enkelt gränssnitt som gjorde att jag på ett smidigt sätt kunde genomföra tester på programmet under utvecklingens gång. När så slutligen hela programmet var färdigställt testkörde jag det med varierande indata och val parsestrategier.

4 Implementationsdokumentation

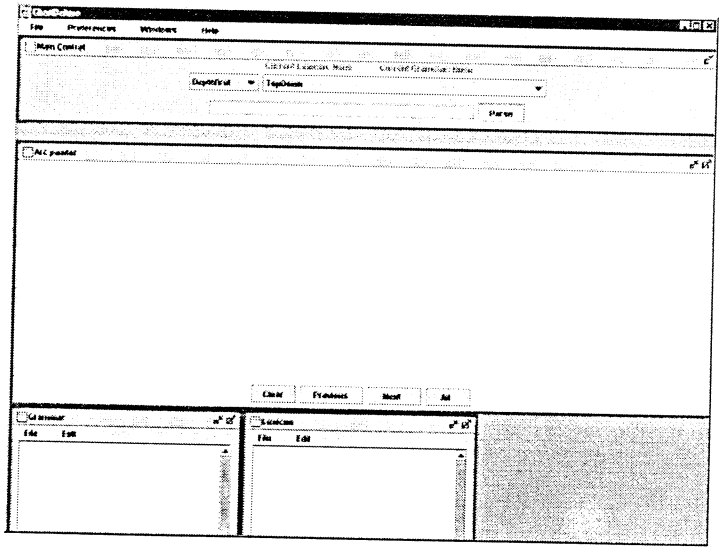
4.1 Inledning

ChartDeluxe är ett program som är tänkt att användas i ett pedagogiskt syfte. Programmet demonstrerar genom att rita ut en graf hur olika chartparsestrategier fungerar. Genom att programmet demonstrerar hur de olika metoderna arbetar grafiskt samt det faktum att det är möjligt att stega sig fram i parsningen anser jag att det är lätt för användaren att jämföra hur de olika metoderna skiljer sig från varandra. Användaren kan också öppna, spara, skapa och redigera lexikon och grammatiker på så sätt kan hon/han undersöka hur de olika strategierna arbetar utifrån olika indata.

Det här kapitlet beskrivs omständigheter med programmet ChartDeluxe v1.0 som kan vara till intresse för användare samt personer med avsikt att vidareutveckla programmet. Först presenteras information om hur man skall gå till väga vid installation och nyttjande av programmet ChartDeluxe därefter följer information om hur programmet är uppbyggt och implementerat.

När programmet startar öppnas ett maximerat fönster upp. I fönstret finns diverse interna fönster vilka ansvarar för olika funktioner. Från det översta fönstret kan användaren både bestämma vilken strategi han /hon samt bestämma om djupetförst eller breddenförst skall användas samt skriva in den sträng som skall parsas. I det mittersta fönstret ritas den graf som blivit resultat av parsern upp.

Med knapparna i nedre delen av det fönstret kan användaren styra hur grafen skall visas.
De två fönstrena längst ner ansvarar för grammatiker respektive lexikon. Alla manipulering av grammatiker och lexikon styr användaren från fönstrens egna menyer. När användaren vill parsa en sats skall hon/han gå till väga på följande sätt:

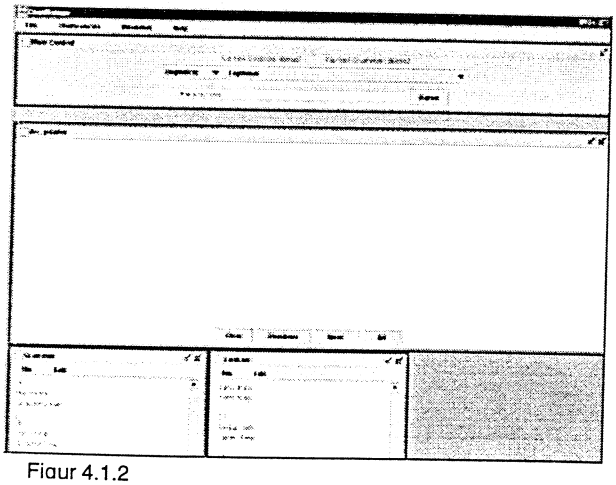


Figur 4.1.1

1. Börja med att öppna en grammatik och ett lexikon genom att gå in på arkivmenyn i de båda fönstrena. Grammatikens regler kommer nu att bli synliga i det högra av de mindre fönstren och lexikonets ord i det vänstra.
Användaren kan lägga, till och radera regler och lexikala ingångar. För att göra det går användaren in menyn redigera i respektive fönster

2. Skriv in den textsträng som skall parsas i textfältet i det interna fönster som är beläget längst upp i huvudfönstret.

Figur 4.1.2 visar hur gränssnittet ser ut efter det att användaren har öppnat en grammatik och ett lexikon samt skrivit in den textsträng som skall analyseras.

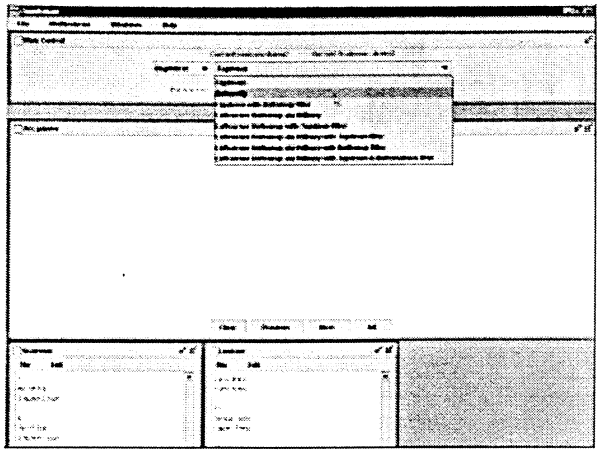


Figur 4.1.2

3. Välj vilken parsestrategi som skall användas från komboboxen i det översta fönstret.
Följande strategier kan användaren välja på:

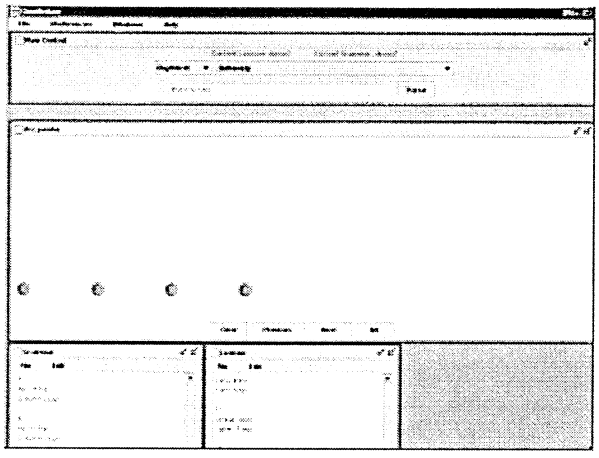
- Topdown
- Bottomup
- Bottomup à la Kilbury
- Topdown med bottomupfilter
- Bottomup med topdownfilter
- Bottomup á la Kilbury med topdownfilter
- Bottomup á la Kilbury med bottomupfilter
- Bottomup á la Kilbury med bottomup & topdownfilter

Figur 4.2.3 visar hur det ser ut när en användare väljer parsestrategi.



Figur 4.1.3

4.Välj om parsern skall använda bredden eller djupetförst sökning i den vänstra komboboxen i det översta fönstret.

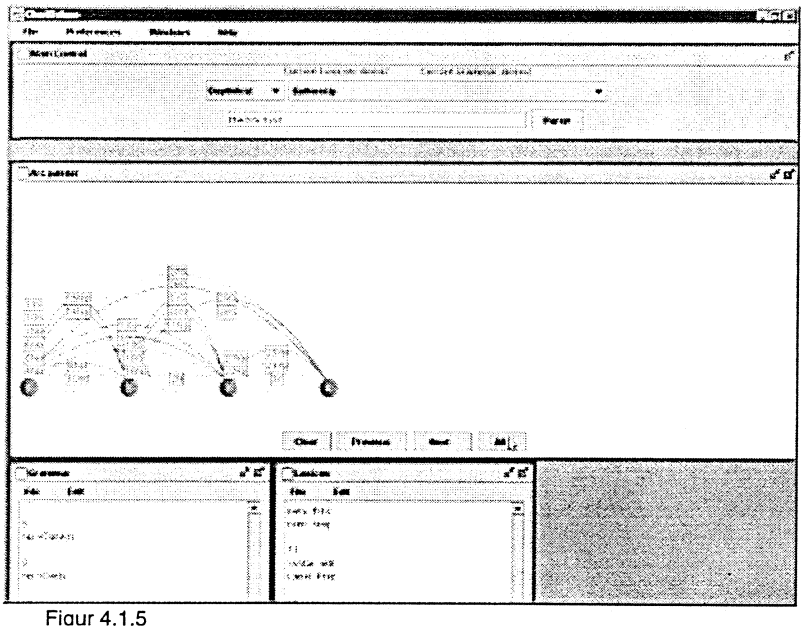


Figur 4.1.4

5. Tryck på knappen parse längst till höger i det översta fönstret för att starta parsningen. Parsningen kan ta ett par sekunder. De noder som ingår i den resulterade grafen ritas nu tillsammans med indatasträngens alla token upp i skärmens mittersta fönster (se figur 4.1.4).

6. Genom knapparna nästa, föregående, töm samt alla längst ner det mittersta fönstret kan användaren bestämma hur grafen skall visas upp. Användaren kan stega sig fram genom grafen, då läggs en båge i taget till grafen eller så kan användaren välja att visa alla grafens bågar från början.

I Figur 4.1.5 har användaren valt att se grafens alla bågar. Samtliga bågar har en skylt placerad rakt över sig. Skylten innehåller information om bågens syntaktiska kategori samt dess ordningsnummer i grafen.



Figur 4.1.5

7 För att få mer utförlig information om en båge klicka på den så öppnas ett separat fönster där informationen presenteras.

4.2 Användardokumentation

Här hittar du en användarmanual till programmet ChartDeluxe v1.0. Jag förutsätter att läsaren har viss datavana och är familjär med att arbeta med fönstergränssnitt och kommer därför inte förklara vad komponenter som menyer och knappar är.

4.2.1 Installera programvaran

I skrivande stund kräver programmet att programvaran jdk1.2 eller senare är installerat i systemet för att kunna det skall kunna exekveras. Kompilatorn finns att ladda ner gratis från Sun Microsystems sida. http://java.sun.com på Internet. Om du har jdk1.2 installerat behöver du bara packa upp filen ChartDeluxe.zip i det valfritt bibliotek.

4.2.2 Starta ChartDeluxe

Win

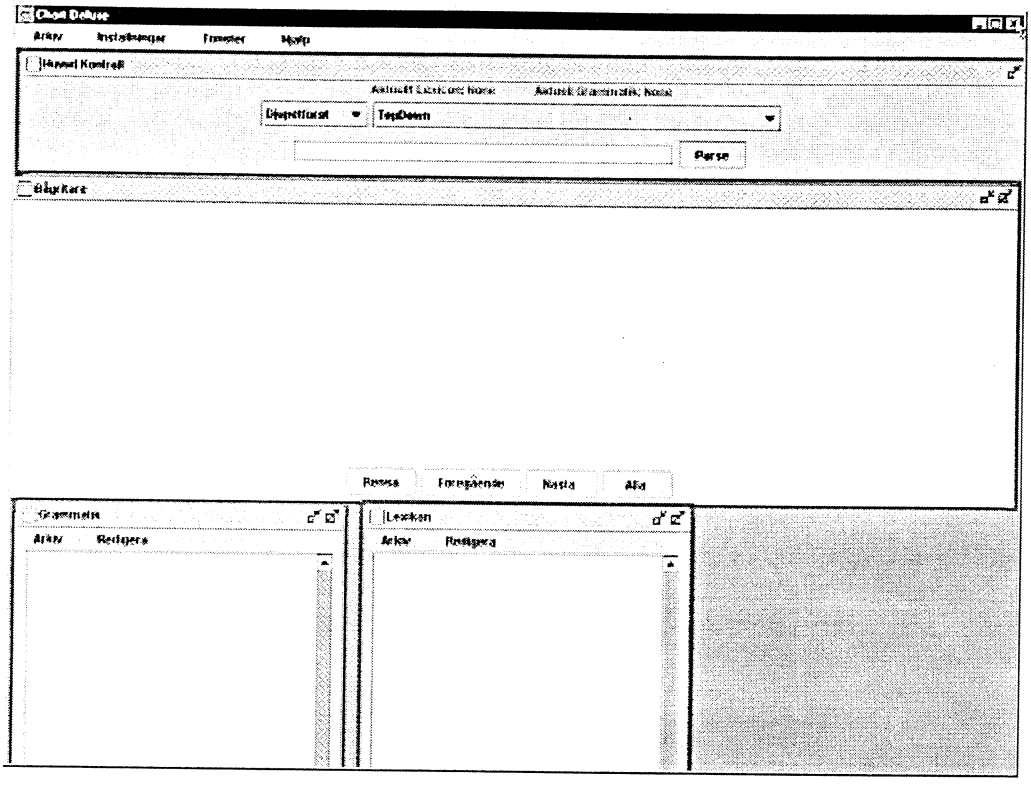
1. Öppna ett Ms-dos fönster.
2. Ställ dig i biblioteket ChartDeluxe
3. Skriv deluxe
4. Programmet startar

Unix

1. Öppna ett terminalfönster
2. Ställ dig i biblioteket ChartDeluxe
3. Skriv deluxe
4. Programmet startar

4.2.3 Programmets struktur

Eftersom de olika funktionerna i programmet är väl åtskilda har jag valt att bygga upp gränssnittet efter en platt men bred struktur. Användaren kan på så sätt lätt skifta mellan programmets olika funktioner samtidigt som hon/han behåller en överblick över programmet. När ChartDeluxe startar öppnas ett fönster som täcker hela bildskärmen.



Figur 4.2.1

I huvudfönstret finns en hel rad interna fönster från vilka programmets funktonaliteten styrs. Samtliga interna fönster kan ikonifieras genom ett musklick på en av de ikoner som sitter uppe i högra hörnet. För att stänga ett fönster går man in i menyn "Fönster" och avmarkerar den post som hänvisar till fönstret i fråga. Det går att flytta omkring och ändra storlek på fönster. Som jag nämnde ovan har jag valt att presentera programmets funktioner utifrån de olika fönster och menyer programmet har. Jag har delat upp det på följande sätt:

- **Huvudmenyerna** - De menyerna som är belägna i huvudfönstrets överkant.
- **Huvudpanel** - Det fönster från vilken val av parsestrategi och inskrivning av indatasträngen sker.
- **Grammatik** - Det fönster från vilket användaren styr de funktioner som har med grammatikerna att göra.
- **Lexikon** - Det fönster från vilket användaren styr de funktioner som har med lexikonen att göra.
- **Bågritaren** - Det fönster där den graf som blir resultatet av en parsning grafiskt presenteras.
- **Båginformation** - Fönster som ger utförlig information om en av användaren utvald båge.

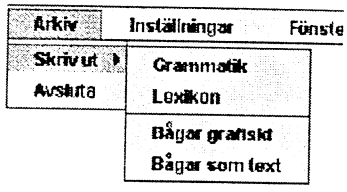
4.2.3.1 Huvudmenyerna

Huvudmenyerna finns på den menylist som är belägen i huvudfönstrets överkant. Från menyerna kan användaren styra generella inställningar som rör programmets gränssnitt. Det går även att få tillgång till ett hjälpavsnitt om programmets samtliga funktioner. Jag kommer att presentera menyerna var för sig .

Arkiv

- **Skriv ut**

  - **Grammatik** skriver ut den grammatik som för tillfället visas i Grammatikfönstret. Är ingen grammatik öppen skrivs inget heller ut.
  - **Lexikon** skriver ut det lexikon som för tillfället är visas i Lexikonfönstret. Är inget lexikon öppet skrivs inget heller ut.
  - **Bågar grafiskt** skriver ut de bågar som är uppritade i bågritarfönstret.



Figur 4.2.2

Notera att det är endast de bågar som användaren valt att visa i bågritarfönstret som skrivs ut.

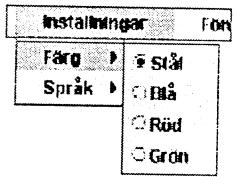
-Bågar som text skriver ut information om samtliga bågar som var resultat av den senaste utförda parsningen. Informationen skrivs ut i text form och är mer utförlig en den som förkommer i bågritaren.

-Avsluta programmet ChartDeluxe avslutas. Alla ändringar tänkbara ändringar användare utfört på grammatiker eller lexikon och som inte sparats förkastas.

Inställningar

I den här menyn kan användaren styra generella inställningar som gäller programmets användargränssnitt. Han/Hon kan vilket språk programmet skall använda på sina menyer och knappar samt i vilka färger gränssnittets komponenter skall visas.

- **Färg**- Ändrar färginställningarna på programmet användaren kan välja mellan fyra olika föreinställda färgscheman: Stål, Blå, Röd och Grön.
- **Språk**- ändrar språk på de texter programmets menyer och knappar använder sig av. I skrivande stund användaren endast välja på engelska och svenska.

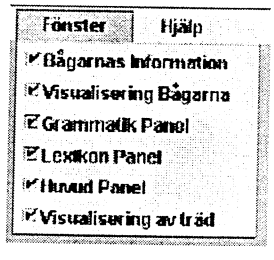


Figur 4.2..3 a

Fönster

I den här menyn kan användaren välja vilka av programmets fönster som skall vara synliga i huvudfönstret. Genom att markerar eller avmarkera den post i menyn som motsvarar det fönster man är intresserad av kan användaren bestämma om ett fönster skall vara synligt eller ej.

- **Båginformation:** Döljer eller visar alla fönster av denna sort.
- **Visualisering av bågarna:** Döljer eller visar bågritarfönstret.
- **Grammatikpanel:** Döljer eller visar grammatikfönstret.
- **Lexikonpanel:** Döljer eller visar lexikonfönstret.
- **Huvudpanel:** Döljer eller visar huvudpanelen
- **Visualisering av träd:** Ej implementerat.

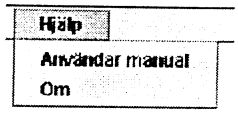


Figur 4.2.3 b

Varje fönster har en föreinställd storlek och position vilket leder till att det inte är säkert att ett fönster efter det användaren valt att det skall vara synligt har den storlek och har den position som den hade innan den doldes. Notera att det för samtliga poster i menyn förutom Båginformation gäller att endast ett fönster av varje sort kan vara synligt åt gången.

Hjälp

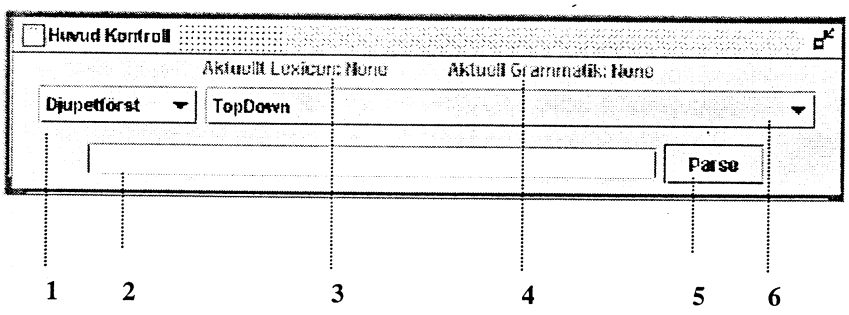
- **Användarmanual**- öppnar ett fönster där en användarmanual över ChartDeluxe visas. Manualen är i htmlformat men p.g.a. att javas stöd för html ännu är på utvecklingsstadiet kan det förekomma endel konstigheter i dokumentet.
- **Om ChartDeluxe** öppnar ett fönster där användare kan få en kortfattad information om program version, samt diverse kuriosa som rör utvecklingen av programmet.



Figur 4.2.3 c

4.2.3.2 Huvudpanel

Huvudpanelen är det fönster från vilket användaren styr vilken indata och vilken parsestrategi hon/han vill använda vid nästa parsning. Observera att man måste ha en grammatik och ett lexikon öppet för att analysen skall gå igenom. Se 4.2.3.3 & 4.2.3.4 om hur man skall gå tillväga för att öppna och stänga en grammatik/lexikon.



Figur 4.2.4

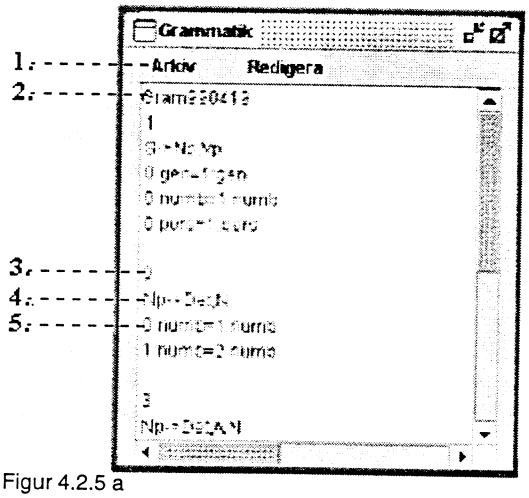
1. **Sökriktningsväljare**- Användaren välja om hon/han skall använda djupet eller bredden först sökning vid nästa parsning.
2. **Indatafält-** I det fältet skriver användaren in den textsträng som hon/han vill att parsern skall analysera. Varje ord skrivs in avskilt av ett mellanslag. Punkter, komman och andra skiljetecken får inte förekomma såvida de inte förkommer som separata poster i lexikonet. Parsern tolkar alla token som ord. Om ett eller flera av orden i indata strängen inte förekommer i det aktuella lexikonet sker inget annat än att analysen inte blir fullständig.
3. **Aktuellt Lexikon -** Visar namnet på det lexikon som är öppet för tillfället. Om inget är öppet står det "None".
4. **Aktuell Grammatik-** Visar namnet på den grammatik som är öppet för tillfället. Om ingen är öppen står det "None".
5. **Start -** Genom att klicka på den här knappen startar användaren parsningen. Resultatet av analysen åskådliggörs i Bågritaren, det finns även möjlighet att skriva ut till en skrivare bågarna som text eller som grafik, se huvudmenyerna.
6. **Parsestrategiväljare-** Här väljer användaren vilken parsestrategi han/hon vill använda för den aktuella parsningen. Man kan givetvis endast använda en strategi i taget men för varje strategi kan användaren välja om sökningen skall ske breddenförst eller djupetförst. De parsestrategier som programmet kan erbjuda är följande:

- TopDown
- Bottomup
- Topdown med bottomupfilter
- Leftcorner bottom up med topdownfilter
- Bottomup á la Kilbury
- Bottomup á la Kilbury med topdownfilter
- Bottomup á la Kilbury med Bottomupfilter
- Bottomup á la Kilbury med bottomup & topdownfilter

Strategierna beskrivs närmare i kapitel 2 Bakgrund avdelning 2.1.4 tidigare i uppsatsen

4.2.3.3 Grammatik

I det här fönstret kan användaren styra all funktonalitet som har med grammatiken att göra. Det går endast att ha en grammatik i taget öppen. Den öppna grammatiken blir automatiskt den som används vid en eventuell parsning. Det är alltså härifrån användaren bestämmer vilken grammatik som skall användas vid parsning inte från huvudpanelen. Det måste alltså finnas en grammatik öppen för att parsningen skall gå igenom.



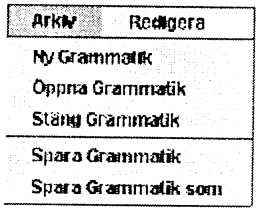
Figur 4.2.5 a

För att manipulera grammatikens regler eller administration av grammatik hänvisas användaren till någon av de menyer fönstret tillhandahåller. De beskrivs mer i detalj nedan. Mitt i fönstret visas information om den öppna grammatiken. Överst står grammatikens namn under det alla ord det innehåller. Orden står i den ordning de skapades och de är numrerade för att underlätta vid borttagning och för att ge god en överblick. Grammatikfönster har även två egna menyer Arkiv och Redigera, det är från dessa man kan manipulera med grammatiken de beskrivs mer i detalj nedan. Observera att man inte kan redigera direkt i grammatiken genom att klicka fönstret och börja skriva sådana ändringar ignoreras av programmet.

1. **Menylist:** Här finns menyerna Arkiv och Redigera ifrån vilka man kan manipulera grammatiken.
2. **Grammatikens namn:** Namnet på grammatiken. Namnet bestämmer användaren då han/hon sparar grammatiken med kommandot spara som i Arkiv menyn (se nedan). När en ny grammatik skapas får den alltid namnet "New Grammar" tills man döper om den.
3. **Regelns ordningstal:** Reglerna står utskrivna i den ordning de skapades, ordningstalet talar om var i grammatiken regeln står. Ordningstalet används bl. a. när man skall radera en regel från grammatiken.
4. **Kategori:** vänsterledet och högerledet i regeln. Regeln har formen Kategori -> Kategori (,Kategori)* Regel består alltså av ett vänsterled följt av en pil och därefter högerledet, ett godtyckligt antal kategorier avskilda med att kommatecken. Det måste finnas minst en kategori i högerledet och exakt en i. vänsterledet
5. **Unifieringsregler:** Här står de särdragsöverensstämmelser som gäller för regeln.

Arkiv

Den här menyn används för att skapa, öppna, stänga och spara grammatiker. Observera att Huvudpanel automatiskt uppdateras vid samtliga av de funktioner som kan utföras här.



Figur 4.2.5 b

- **Ny Grammatik:** Skapar en ny tom grammatik.
- **Öppna Grammatik**: En fildialog öppnas där man kan välja den grammatik man vill öppna. Alla grammatikfiler har suffixet ".gram ". Försöker användaren öppna en fil som inte är en grammatikfil ges ett felmeddelande.
- **Stäng Grammatik:** Stänger den öppna grammatiken
- **Spara Grammatik:** Sparar den aktuella grammatiken
- **Spara Grammatik som:** Öppnar en fildialog där användaren kan skriva i det namn hon/han vill döpa grammatiken till och var du vill spara det.

Redigera

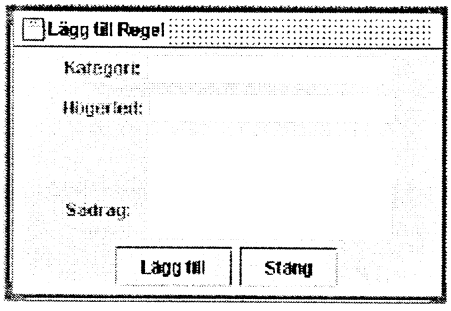
I den här meny kan användaren lägga till och radera regler ur grammatiken. Notera att grammatiken måste sparas innan parsern kan ta hänsyn till ändringarna.

Figur 4.2.5 c

- **Lägg till regel:** Öppnar ett fönster där man kan fylla i alla uppgifter om de nya posten man vill lägga till.
- **Ta bort regel:** Öppnar ett fönster där man fyller i numret på den regel man vill ta bort

Lägg till Regel

Formuläret består av tre fält för användaren att fylla i varav de två första är obligatoriska. I det översta fältet skall man fylla i regelns vänsterled. I det mittersta fältet skall man fylla i regelns högerled. Regel måste minst ha ett objekt i högerledet. Högerledet har formen *kategori (,kategori)**.



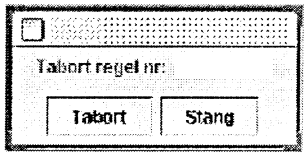
Figur 4.2.5 d

a) 1:numb=2:numb
b) 0:numb=2:numb

Exempel 4.2.1

I det nedersta fältet skriver man in regelns särdragsposter alltså de restriktion som gäller särdragsunifiering i de bågar som regeln eventuellt kommer ge upphov till.
En särdragspost har formen `i:X0=j:Y0`. Där i & j är heltal som representerar de olika fraserna i regelns höger och vänsterled särdragsunifieringen gäller. Siffran 0 avser vänsterledet och 1 den första i högerledet osv. X_0 & Y_0 talar om vilket särdag i de olika fraserna som restriktionen gäller i den aktuella särdragsposten. Kolon tecknen fyller ingen annan funktion en att fungerar som separatorer lika så likmedtecknet. En särdragspost kan antingen uttrycka ett vilkor eller en tilldelning. Värdet av postens första element bestämmer vilket av dem det blir. Är det 0 tilldelning annars villkor.
Med hjälp Exempel 4.2.1 till vänster skall jag försöka förtydliga fenomenet. Post a i exempel 4.2.1 uttrycker att värdet av särdraget numb i vänsterledet första element skall vara identiskt med värdet av särdraget numb i vänsterledet andra element för att de skall kunna unifieras. Post b i samma exempel uttrycker istället tilldelning. Den påpekar att värdet på särdraget numb i samtliga bågar som regeln kommer ge upphov till skall få sitt värde från det andra sökta elementets särdrag numb.
Om någon av de bågar som posterna kommer att gälla saknar det aktuella särdraget ignorerar parsern restriktionen. För att lägga till en regel tryck på knappen "Lägg till". Man kan lägga till flera regler innan man stänger fönstret. För att stänga fönstret tryck på knappen "Stäng".

Ta bort regel

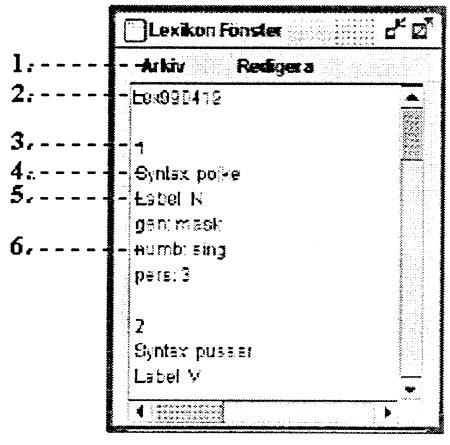


Figur 4.25 e

För att radera en regel skall användaren skriva in ordnings numret på den regel hon/han vill ta bort. Den första regeln i grammatiken har alltid ordningsnummer 1.För att ta bort ordet tryck på knappen "Ta bort". Du kan ta bort flera regler innan du stänger fönstret. När en regel tas bort skiftas alla regler ett steg uppåt för att fylla tomrummet som blir i grammatiken. Alltså ändras ordningsnumren efter varje avlägsnande.

4.2.3.4 Lexikon

Lexikonfönstret påminner mycket om grammatikfönstret all funktonalitet som rör lexikon kan användare styra härifrån.
Liksom grammatikfönstret kan lexikonfönstret hantera ett lexikon i taget. Den öppna lexikonet blir automatiskt det som används vid en eventuell parsning.
Det är alltså härifrån användaren bestämmer vilket lexikon som skall användas vid parsning och inte från huvudpanelen. För att parsningen skall gå igenom måste ett lexikon vara öppet. För att manipulera de ord som lexikonet innehåller eller administration av lexikon hänvisas användaren till någon av de menyer som är belägna i fönstrets överkant. Mitt i fönstret visas information om det öppna lexikonet. Överst står lexikonets namn under det alla ord det innehåller. Orden står i den ordning de skapades de är numrerade för att underlätta vid borttagning och för att ge en överblick. Observera att man inte kan redigera direkt i grammatiken genom att klicka fönstret och börja skriva. Sådana ändringar ignoreras av programmet.

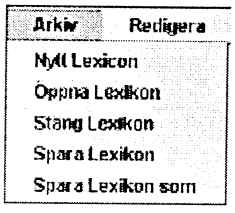


Figur 4.2.6 a

1. **Menylist:** Menyerna Arkiv och Redigera innehåller de funktionerna som användaren kan nyttja för att styra lexikonet.
2. **Lexikon namn:** Namnet på lexikonet. Namnet bestämmer användaren då han/hon sparar lexikonet, med kommandot "Spara Lexikon som" i Arkiv menyn se nedan. När ett nytt lexikon skapas får den alltid namnet "New Grammar" tills användaren döper om den.
3. **Ordningsnummer:** Varje ord i lexikonet har ett ordningstal det är förknippat med var i lexikonet ordet är beläget. Ordningstalet används bl.a. vid bortagning av en post.
4. **Syntax:** Ordets syntax..
5. **Kategori:** Den syntaktiska kategori ordet tillhör..
6. **Särdagspost:** Information om de särdrag ett ord har lagras i form av särdragsposter. Särdragspost har formen X_0:V_0 .Där X_0 är namnet på särdraget och V_0 är dess värde. Ett ord kan ha ett godtyckligt antal särdragsposter.

Arkiv

När användaren vill skapa, öppna ,stänga eller spara ett lexikon väljer hon/han något av alternativen i den här menyn. Observera att Huvudpanel automatiskt uppdateras vid samtliga av de funktioner som kan utföras här.

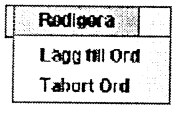


Figur 4.2.6 b

- **Nytt Lexikon:** Skapar ett nytt tomt lexikon. Huvudpanelen uppdateras automatiskt och sätter det som aktuellt lexikon
- **Öppna Lexikon:** En fildialog öppnas där användaren kan välja det lexikon hon/han vill använda. Alla lexikonfiler har suffixet ".lex ". Försöker man öppna en fil som inte är ett lexikon ges ett felmeddelande. När ett lexikon har öppnats uppdateras Huvudpanelen automatiskt och sätter det som ny öppnade lexikonet som aktuellt lexikon.
- **Stäng Lexikon:** Stänger det öppna lexikonet.
- **Spara Lexikon:** Sparar det aktuella lexikonet.
- **Spara Lexikon som:** Öppnar en fildialog där användaren kan skriva i det namn hon/han vill döpa lexikonet till och var du vill spara det.

Redigera

I den här meny kan användaren lägga till och radera ord ur lexikonet. Notera att grammatiken måste sparas innan parsern kan ta hänsyn till ändringarna.

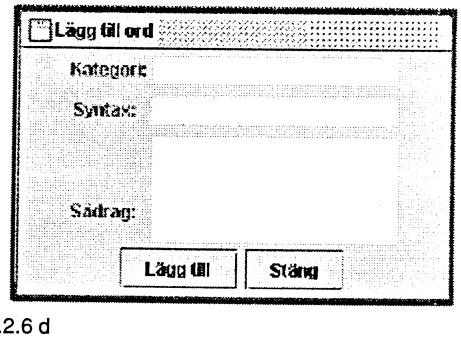


Figur 4.2.6 c

- **Lägg till ord:** Öppnar ett fönster där man kan fylla i all uppgifter om de nya posten.
- **Ta bort Ord:** Öppnar ett fönster där man fyller i numret på det ord man vill ta bort.

Lägg till ord

Formuläret består av tre textfält : Kategori, Syntax och Särdrag varav två översta är obligatoriska för användaren att fylla i. I Fältet Kategori fyller användaren i en valfri sträng som skall representera ordets syntaktiska kategori i fältet syntax. I det nedersta fältet fyller användaren i de särdrag som hon/han vill tilldela det aktuella ordet. Särdagen skrivs in i form av ett godtyckligt antal särdagsposter. En särdragspost har strukturen X_0:V_0 Där X_0 är namnet på särdraget och V_0 är dess värde exempelvis `gen:mask` .

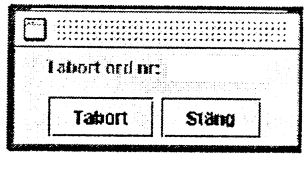


Figur 4.2.6 d

För att lägga till ett ord till lexikonet trycker klickar man på knappen "Lägg till". Det är möjligt att lägga till flera ord innan man stänger fönstret lexikonet. För att stänga fönstret tryck på knappen "Stäng".

Ta bort ord

Skriv in ordnings numret på det ord du vill ta bort, 1 är det första.. För att ta bort ordet tryck på knappen "Ta bort". Du kan ta bort flera ord innan du stänger fönstret. När ett ord tas bort skiftas alla ord ett steg för att fylla tomrummet.



Figur 4.2.6 e

4.2.3.5 Bågritaren

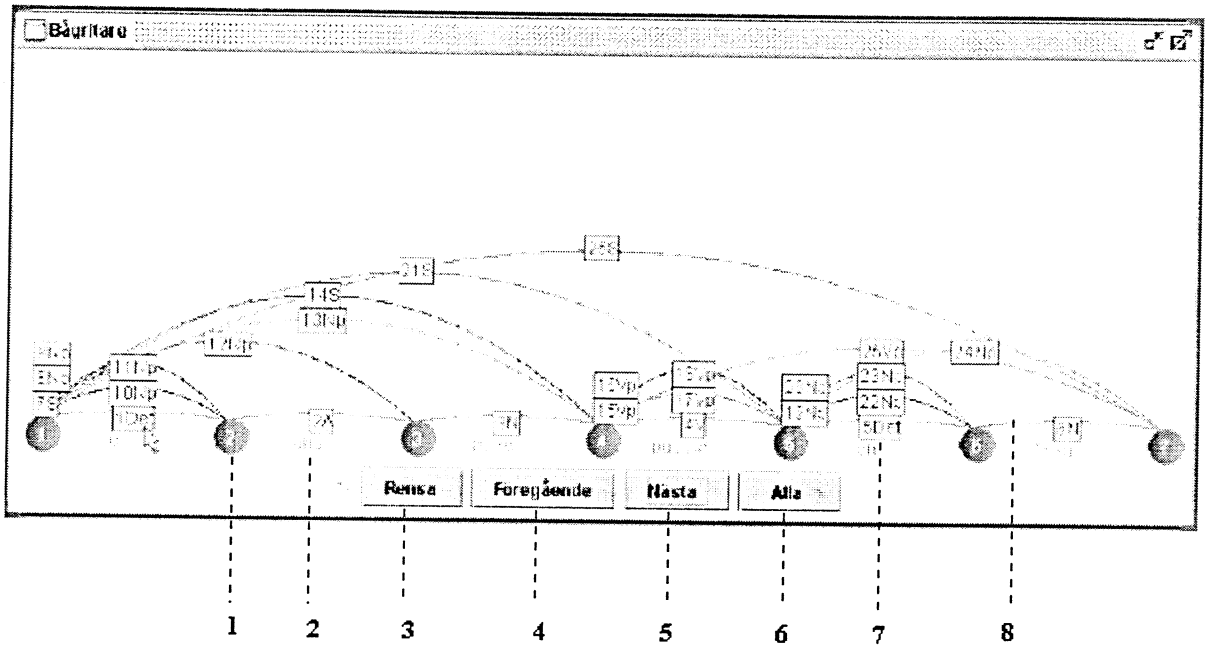
Den grafiska representationen av bågarna som blev resultatet från den senaste parsningen kan beskådas i det här fönstret. När en indatasträng blivit analyserad av parsern ritas först alla grafens noder ut i fönstret. Därefter måste användaren själv interagera med programmet för att grafens bågar skall bli synliga.

Användaren kan välja om hon/han vill att grafens samtliga bågar skall visas på en gång eller om hon/han manuellt vill stega sig fram genom grafen. Om användaren väljer det senare alternativet läggs en båge i taget till i fönstret. Bågarna läggs till i den ordning de skapades i parsningen.

Användaren kan även avlägsna bågar från bågritarfönstret om det skulle vara önskvärt. Bågarna avlägsnas i omvänd kronologisk ordning. Antingen en i taget eller alla på en gång. Den information som presenteras om bågarna innehåller bågens ordningsnummer, syntaktisk kategori samt start och sluttillstånd. Skälet till att inte mer information visas är att risken att det skulle bli väldigt rörigt är överhängande

Vill man ha mer information om en båge klickar man med musen på den aktuella bågen. Då öppnas ett nytt separat fönster där mer utförlig information presenteras (se båginformation).

All interaktionen med användaren sker genom de knappar som är belägna längst ner i fönstret.



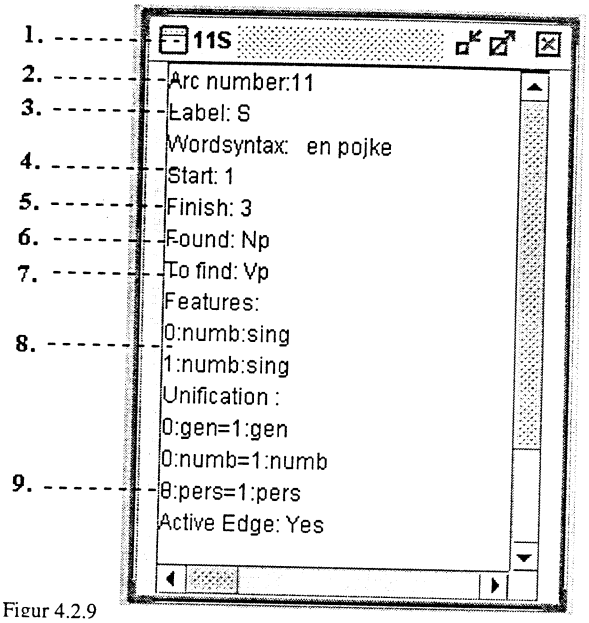
Figur 4.2.7

1. **Nod:** De grafiska representationerna noder. Siffran avser ordningsnumret i grafen.
2. **Båge:** Den grafiska representationen av en båge. Aktiva bågarna gestaltas med mörkare färg än de inaktiva. Genom att klicka på en båge kan man få mer utförlig information om den i ett separat fönster som öppnas. Se båginformation
3. **Informationsruta**: Varje grafisk båge har en ruta som visar bågens ordningsnummer samt vilken kategori den har. Genom att klicka på en sådan ruta kan man få mer utförlig information om den båge som den representerar i ett separat fönster. Se båginformation.
4. **Rensa**: Avlägsnar alla bågar från ritytan.
5. **Föregående**: Avlägsnar den senast ritade bågen från ritytan.
6. **Nästa**: Ritar ut den bågen som ligger ordningsmässigt i tur.
7. **Alla**: Ritar ut alla bågar i den aktuella parsningen.

4.2.3.6 Båginformation

Den information användaren får om de olika bågarna i bågritaren kan tyckas knapphändig men det går att få mer utförligare uppgifter om bågens tillstånd. Användaren kan klicka på den båge hon/han är intresserad av i bågritaren då öppnas ett fönster jag valt att kalla båginformation. I det radas diverse information om bågen upp i textform. Det finns information om bågens särdragsstruktur, unifieringsrestriktioner, start och slutnod ,ordningsnummer, kategori samt vilka sökta kategorier som bågen hittat.
Det är fullt möjligt att ha flera båginformationsfönster öppna samtidigt vilket kan vara smidigt då man vill jämföra två eller fler bågar.

1. **Titel på fönster :** Först står bågens ordningsnummer därefter bågens kategori- Titeln är till för att underlätta för användaren då man direkt kan se om vilken bågen information visas då flera fönster av aktuell typ kan förekomma.
2. **Ordningsnummer**: Avser vilken när i parsningen den aktuella bågen skapades,.
3. **Kategori**: Bågens syntaktisk kategori.
4. **Startnod**. Ordningsnumret på den nod i grafen från vilken bågen startar.
5. **Slutnod**. Ordningsnumret på den nod i grafen som är bågens destination.
6. **Funna:** De sökta bågar som den aktuella bågen lyckats identifiera representerade av deras syntaktiska kategori.
7. **Att finna:** De bågar den aktuella bågen behöver för att bli komplett men ännu inte funnit representerade av deras kategorier.
8. **Särdrag:** De särdrag bågen äger.(Se sidan 13)
9. **Unifieringsregler:** De unifieringsrestriktioner som gäller för den aktuella bågen.
10. **Aktiv:** Visar om bågen är aktiv eller inte.



Figur 4.2.9

4.3.2 Detaljerad systembeskrivning

Jag kommer här att ge mer ingående information om de klasser de olika paketen innehåller. Jag har valt ut några klasser och metoder som jag tror kan vara av intresse och kommer ge en ändå mer ingående förklaring av dem. Referensmanual till klasserna finner ni i bilaga 1.

4.3.2.2 Klasser

Jag kommer här att belysa de icketriviala omständigheter och algoritmer som ingår i den delmängd av klasser jag valt att fokusera mig på. Då många av klasserna är triviala eller har struktur och/eller funktonalitet som jag bedömer var ointressant för en eventuell utvecklare kommer jag endast beskriva en del klasser. Då jag anser att rena implementerings detaljer i form av utdrag av källkod inte skulle fylla någon större funktion här har jag valt att utlämna sådant. De algoritmer jag valt att belysa kommer att beskrivas enligt följande:

Namn: Namnet på Javametoden
Indata: Metodens indata parametrar representeras av dess datatyp samt en variabel struktur. Void betyder att metoden inte tar några indata parametrar.
Utdata: Typen på det objekt metoden returnerar. Void betyder att metoden inte returnerar något värde.
Metod: Det arbetsätt utifrån vilket den aktuella metoden arbetar.
Motivation: Innehåller en förklaring till när metoden används och varför jag valt ovanstående algoritm. Diverse kommentarer kan också förekomma.

ChartParser

ChartParser är den klass som innehåller många av de metoder som är väsentliga när det gäller programmets parsning. Alla de olika parsestrategier som jag valt att implementera återfinns i den här klassen tillsammans med diverse söknings och hjälp metoder. Klassen ChartParser har en väldigt enkel struktur den innehåller endast två attribut.

```
ChartStack agenda;
int howManyarcs;
```

Variabeln agenda är stacken som parsemetoderna jobbar emot. Beroende på om användaren väljer djupet eller bredden först typomvandlas agenda till en Fifo eller Lifostack. De bågar som blir resultatet av parsningen hamnar här. Variabeln howManyarcs håller reda på hur många bågar som finns i agenda den används för att hålla tilldela rätt ordnings nummer till bågarna.

Namn: **scan**
Indata: ComplexArc <i, j, $X_0 \rightarrow .Y_0\,\beta, D_0$>
Utdata: Vector som de nya tommat bågar den inaktiva indata bågen givit upphov till.
Metod: För varje inaktiv båge <m,n,$Z_0 \rightarrow W_n.,D_n$> skapa en ny båge< i, j, $X_0 \rightarrow Y_0.,D_0:D_n$ > om unifiering lyckas.
Motivation: Metoden som krävs för att aktiva bågar skall kunna expanderas. Metoden anropas i från de olika parsemetoderna.

Namn: **letsParse**
Indata: int i, boolean depthfirst, ChartGrammar gram
Utdata: void.
Metod: Integern är en switch som bestämmer vilken parsemetod som skall användas;
0. Topdown
1. Bottomup
2. Bottomup à la Kilbury
3. Topdown med bottomup filter
4. Bottomup med topdownfilter
5. Bottomup á la Kilbury med topdownfilter
6. Bottomup á la Kilbury med bottomupfilter
7. Bottomup á la Kilbury med bottomup & topdownfilter

Parametrarna gram och depthfirst skickas vidare till parsemetod anropen.

Motivation: Metoden anropas är parserns kontakt med gränssnittet. Metoden anropas i klassen ParselFrame.

Namn: **topdown**
Indata: ChartGrammar gram, boolean depthfirst
Utdata: void.
Metod: Se kapitel 3
Motivation. Implementationen av Topdown algoritmen.

Namn: **selectivetopdown**
Indata: ChartGrammar gram, boolean depthfirst
Utdata: void.
Metod: Se kapitel 3
Motivation. Implementationen av Topdown med bottomupfilter algoritmen.

Namn: **ICBupalaKill**
Indata: ChartGrammar gram, boolean depthfirst, boolean kilbury
Utdata: void.
Metod: Om kilbury == false arbetar metoden efter Bottomup annars LeftCorner á la Kilbury Se kapitel 3
Motivation. Implementationen av Bottomup och LeftCorner á la Kilbury algoritmerna..

Namn: **ICBupalaKillTop**
Indata: ChartGrammar gram, boolean depthfirst, boolean kilbury
Utdata: void.
Metod: Om kilbury == false arbetar metoden efter Bottomup med topdownfilter annars LeftCorner á la Kilbury med Topdown filter Se kapitel 3
Motivation. Implementationen av Bottomup med topdownfilter och Bottomup á la Kilbury med topdownfilter algoritmerna..

Namn: **ICBupalaKillBot**
Indata: ChartGrammar gram, boolean depthfirst, boolean kilbury
Utdata: void.
Metod: Om kilbury == false arbetar metoden efter Bottomup med bottomup filter annars LeftCorner á la Kilbury med bottomup filter Se kapitel 3
Motivation. Implementationen av Bottomup med bottomup filter och Bottomup á la Kilbury med bottomupfilter algoritmerna..

Namn: **ICBupalaKillBotTop**
Indata: ChartGrammar gram, boolean depthfirst, boolean kilbury
Utdata: void.
Metod: Om kilbury == false arbetar metoden efter Bottomup med topdownfilter annars LeftCorner á la Kilbury med Topdown& filter Se kapitel 3
Motivation. Implementationen av Bottomup med topdown & bottomup filter och Bottomup á la Kilbury med bottomup & topdownfilter algoritmerna..

ChartGrammar

Jag har valt att representera grammatiker liksom lexikon med separata javaklasser. ChartGrammar respektive ChartLexikon innehåller regler och ord som även de representeras av separata klasser ChartRule och ComplexWord. Vid lagring av grammatiker och lexikon på disk använder jag mig utav något som i Java heter objektserialisering[11] som gör det möjligt att spara en instans av en klass som en separat fil. Fördelen med det är att man inte behöver göra några manuella omvandlingar av det aktuella objektet vid läsning eller skrivning.

[11] Se Flanagan(1997) sida 172

Klassen ChartGrammar är en relativt enkel struktur. Den består av två poster en sträng som innehåller namnet på den aktuella instansen av klassen och en vektor som innehåller grammatikens regler. Namnet används vid lagring av grammatiken, den fil som skapas får det aktuella namnet plus suffixet ".gram". Reglerna i vektor är instanser av klassen ChartRule.

ChartGrammar innehåller även en mängd metoder. En del metoder rör rent administrativa funktioner beträffande de olika reglerna lagrade i vektor. De metoderna är såpass triviala att jag valt att inte beskriva de närmare utan hänvisar till referensmanualen ovan. Det finns även metoder som behandlar områden som gäller själva parsningen det är ett urval av dem jag beskriver nedan.

Namn: **generateLeftCEArc**
Indata: ComplexArc <i, j, $X_0 \rightarrow Y_0.$,D>, boolean kilbury
Utdata: Vector som de nya tommat bågar den inaktiva indata bågen givit upphov till.
Metod: För varje regel av formen <$Z_0 \rightarrow Z_1\ \beta$, E> i grammatiken X om $X_0=Z_1$ skapa en ny båge om kilbury =true <i, j, $Z_0 \rightarrow Z_1.\ \beta$,E> annars <i, i, $Z_0 \rightarrow .Z_1\ \beta$,E> och lägg den i en vektor.
Motivation: Metoden används i bottomup metoderna.

Namn: **generateSArc**
Indata: int i
Utdata: Vector som de nya tomma bågarna.
Metod: För varje regel av formen <$Y_0 \rightarrow Y_1\ \beta$, D> i grammatiken X med startsymbol X_0, om $Y_0 = X_0$ skapa en ny båge <0,0, $Y_0 \rightarrow .Y_1\beta$,D > och lägg den i en vektor.
Motivation: I topdown parsestrategier initieras alltid agenda genom att man lägger på tomma aktiva bågar för varje regel som kan nås från grammatikens startsymbol. Den här metoden skapar bågar från de regler som har grammatikens startsymbol som vänsterled. Metoden anropas i metoden initilazeTopDown i klassen ChartParser.

Namn: **generateEArc**
Indata: <i, j, $X_0 \rightarrow X_1.$,D>
Utdata: Vector med de nya tomma bågarna.
Metod: För varje regel av formen <$Y_0 \rightarrow Y_1$, E> i grammatiken X gäller att om $Y_0 = X_1$ skapa en ny båge <j, j, $Y_0 \rightarrow .Y_1$, E >
Motivation: Metoden anropas vid initialisering av topdown strategierna

ChartLexicon

Lexikonet är liksom grammatiken enkel i strukturen. Klassen ChartLexikon innehåller innehåller två poster en sträng som representerar namnet på lexikonet samt en vektor där samtliga ord i lexikonet ligger lagrade i form av instanser av klassen ComplexWord. Jag har valt att endast beskriva en metod ur klassen ChartLexikon.

Namn: **initComplexcheck**
Indata: String α
Utdata: Vector vilken innehåller alla ord i lexikonet som har samma syntax som indata strängen.
Metod: För varje lexikoningång av formen <Y_0 ,D> i lexikonet Γ gäller att om $Y_0 = \alpha$ lägg en kopia av lexikon ingången i en vektor.
Motivation: Metoden anropas av metoden initComplex i klassen ChartParser. Metoden används för att kontrollera om en token finns med i lexikonet.

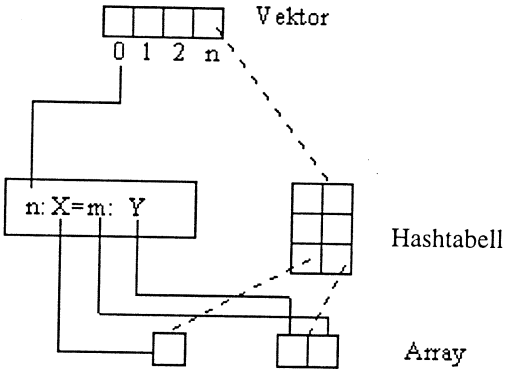
ChartRule

De regler som lagras i ett objekt av typen ChartGrammar representeras av ett instanser av klassen ChartRule. ChartRule är en datastruktur med tre poster:

```
String lhs;
Vector rhs;
Vector feat;
```

De två första poster är relativt simpla. Den första , `lhs` är den sträng som representerar regelns vänsterled, vektorn `rhs` innehåller de strängar som representerar de kategorier som återfinns i

regelns högerled. Den tredje posten `feat` är en vektor som innehåller poster som handhar de särdragsrestriktioner regeln besitter. En post i vektorn har följande struktur: En särdragspost har formen `i:`X_0`=j:`Y_0. Där i & j är heltal som representerar de olika fraserna i regelns höger och vänsterled särdragsunifieringen gäller. Siffran 0 avser vänsterledet och 1 den första i högerledet osv. X_0 & Y_0 talar om vilket särdag i de olika fraserna som restriktionen gäller i den aktuella särdragsposten. Kolontecknen fyller ingen annan funktion en att fungerar som separatorer lika så likmedtecknet. En särdragspost kan antingen uttrycka ett vilkor eller en tilldelning (se sid. 27)
När man i parsningen skall utföra särdragsmatchningen måste man få tillgång till särdagsposten olika beståndsdelar. Jag försökte ha det i åtanke då jag implementerade dem. Bilden nedan demonstrerar hur vektorn feat är uppbyggd.



Figur 4.3.4

ComplexArc

Bågar har en central roll i chartparsing. För att smidigt kunna hantera dem har jag skapat två klasser Arc och ComplexArc. Arc är en abstrakt klass jag som innehåller alla de metoder och attribut som bågar som inte hanterar särdragstrukturer behöver. Jag skapade den för jag tyckte det var en god idé att ha en relativt enkel klass att utgå i från då man skall tillföra särdragsmatchning. På det sättet binder en utvecklare sig inte vid den representation av särdrag jag skapat.
ComplexArc är en klass som utökar Arc. Utökning innefattar poster och metoder för handhavande av särdrag. ComplexArc har följande attribut:

`String label:`
Bågens kategori
`StringBuffer` `wordsyntax:`
Den sammanlagda syntaxen av de bågar som innefattas i den aktuella bågen.
`Vector found:`
En vektor som innehåller kategorierna på de sökta bågar som den har hittat
`Vector tofind:`
En vektor som innehåller kategorierna på de bågar som den aktuella bågen söker.
`boolean active:`
En flagga som visar om bågen är aktiv eller inte
`int start:`
Bågens start position
`int finish:`
Bågens slutposition
`int arcnumber:`
Bågens ordningsnummer-
`Vector features:`
De särdrag som bågen innefattar finns i vektorn. Varje post i vektorn innehåller en särdragsrepresentation av samma struktur som i ComplexWord.
`Vector unify:`
Bågens unifieringsregler ryms i den i en vektor struktur är identiskt med den i ChartRule.

Namn: **arcExists**
Indata: Vector arcs
Utdata: boolean. Returnerar true om det finns minst en båge i vektorn arcs som subsummerar($\subseteq$) den båge som anropat metoden.
Metod: Om β är den båge som anropat metoden gäller för varje båge α gäller att om $\alpha \subseteq \beta$ returnera true annars fortsätt leta igenom indata bågarna. Hittar man ingen båge som subsummerar den anropande bågen returnera false.
Motivation: Den här metoden används i samtliga parsestrategier i ChartParser samt. Den används då man skapat nya bågar för att se om de bågarna redan finns på agendan eller inte.

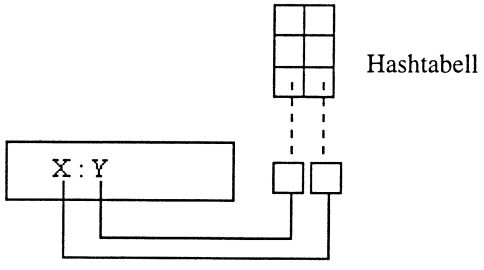
Namn: **isNext**
Indata: ComplexArc <i, j, $X_0 \rightarrow X_1.$,D>
Utdata: boolean. Returnerar true om den inkommande inaktiva bågen kan expandera anropande aktiva bågen
Metod: Den aktiva anropande bågen <k, n, $Y_0 \rightarrow .Y_1$,D> kan expanderas med hjälp av den inkommande bågen om $Y_0 = X_0$, k = j samt och om särdragsunifiering stämmer överens.
Motivation: Metoden anropas av metoden scan i klassen ChartParser för att se om någon tidigare skapad båge kan hjälpa till att expandera den anropande bågen.

ComplexWord

Klassen ComplexWord är en utökning av den abstrakta klassen Word. Den stora skillnaden mellan de olika klasserna är att ComplexWord behandlar särdrag. Det gör inte den enklare strukturen Word. Genom att skapa en superklass Word underlättar jag för eventuella utvecklare som skall skapa en helt annan program men som vill använda sig av en representation av ord som inte innehåller särdrag.

ComplexWord innehåller tre attribut:

`String label:` Ordets kategori.
`String syntax:` Ordets syntax.
`Hashtable features:` Textrepresentationen av ett ords särdrag har formen X:Y där X är en godtycklig typ av särdrag ex. `genus` och Y är det värdet på X ex. `fem`. En post i en haschtabell består av en nyckel som har ett värde associerat med sig. Den strukturen passar utomordentligt bra för mitt ändamål.



Figur 4.3.5

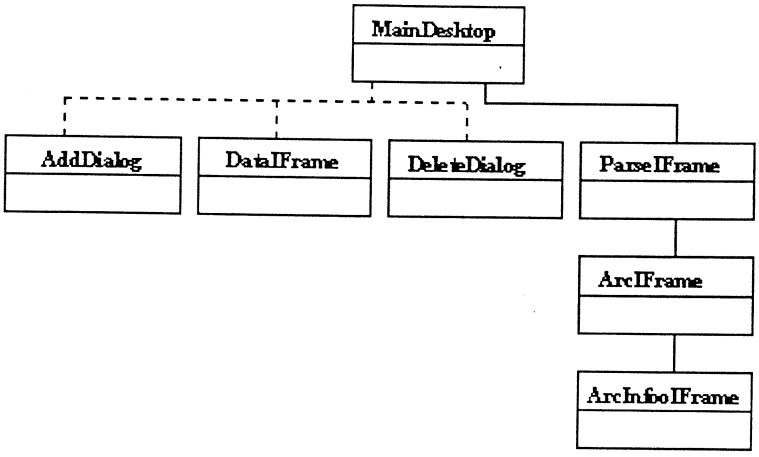
Klassen innehåller få metoder och samtliga är såpass intuitiva att jag valt att inte beskriva någon närmare.

MainDesktop

Som jag tidigare nämnt har jag i största möjliga mån försökt hålla gränssnittskomponenterna isär från resten av programmet däremot är inte gränssnitt komponenterna lika väl avskilda från varandra. Därför måste i många fall flera olika komponenterna ha tillgång till samma information samtidigt. För att lösa det problemet valde jag att använda mig av så kallade innerklasser[12]. Klasser som deklareras i en annan klass. Den klass som innehåller innerklasserna får på det sättet tillgång till all

[12] För mer information se Campione & Walrath(1998)

information som dessa och kan dirigera dem som de vill. Eftersom klassen MainDesktop är en utökning av klassen java.swing.Desktop
Figuren nedan visar hur tillgängligheten av information är hierarkiskt uppbyggd i MainDesktop. En moder i hierarkin kan hämta och tilldela information från sina barn och barnbarn men det fungerar inte tvärt om. Den nod som är högst i hierarkin har störst tillgång till information.
De streckade linjerna i figuren signalerar att dotter noder är innerklasser till modernoden de heldragna visar att klasserna är fristående.



Figur 4.3.6

MainDesktop innehåller diverse attribut, metoder och innerklasser men de allra flesta används endast internt av klassen själv.

4.4 Utvärdering

Eftersom jag har lärt mig programspråket Java under utvecklingens gång är de implementerade lösningar jag använt i programmet inte alltid de mest optimala i beträffande struktur och/eller effektivitet. Jag kommer här att beskriva en del av de fenomen jag med den kunskap jag har idag hade valt att implementera på ett annat sätt.

4.4.1. Eventuella ändringar i koden

Givetvis finns det en mängd detaljer i koden som man i efterhand insett kunnat utförts på ett bättre men jag har istället för att ta upp alla dem valt att belysa några mer omfattande fenomen.

Trådar

Det första jag skulle gjort hade varit att använda mig av trådar[13] en funktion i Java som gör det möjligt att köra flera processer samtidigt. Själva strukturen av gränssnittet skulle då bli ändå mer modulär på det sättet och jag hade sluppit använda innerklasser för att lösa problemet med delad information.

Statiska metoder

Som programmet är skrivet måste man först skapa en instans av klassen ChartParser innan man komma åt de parsemetoder som klassen innehåller. Detta kan tyckas klumpigt. En snyggare lösning vore att låta metoderna vara statiska. På det sätten kan man få använda dem utan att en instans av klassen först skapas. De ändringar hade förståss lett till att omfattande ändringar i programmet struktur skulle vara tvunget att göras.

[13] För mer information se Campione & Walrath(1998).

4.4.2 Buggrapport

Förutom de buggar som presenteras nedan har programmet en tendens att emellanåt låsa sig då flera program samtidigt körs på en och samma dator, åtminstone under Microsofts Windows 95

Spara som

Det spelar ingen roll vilket bibliotek användaren väljer när hon/han skall spara en lexikon eller grammatik fil de hamnar alltid i biblioteket chartdeluxe_v.1.0.

Namnet ändras inte

Om användaren valt att spara en grammatik eller lexikon med ett annat namn än det som filen hade tidigare uppdateras filen men inte huvudpanelen. Huvudpanel visar fortfarande det gamla filnamnet men använder den nya

4.5 Utvecklingsdokumentation

Implementering har fortgått i varierande takt som vanligt har allt tagit mycket längre tid än man planerat. Men en sak är säker om jag hade haft större kunskaper i Java hade många av de tidskrävande problem jag stött på kunnat undvikas. När det gäller programmering i Java kan problem uppstå då man inte är bekant med alla paket som finns tillgängliga och man försöker konstruera en klass eller metod vars funktion redan finns deklarerad i ett paket. Det kan också vara svårt att veta var man skall leta efter en klass som uppfyller de funktioner man är ute efter.

5. Resultat

I det här kapitlet kommer jag att kortfattat beskriva programmet ChartDeluxe v.1.0 som blev resultatet av mitt arbete. Jag kommer belysa ett par begränsningar samt möjliga utvidgningar till programmet. Ett avsnitt med slutsatser samt en sammanfattning återfinns också här.

5.1 ChartDeluxe v1.0

Målet med arbetet, att konstruera ett program som demonstrerar olika chartparsestrategier, har resulterat i ett verktyg jag valt kalla ChartDeluxe v1.0. Programmet är implementerat uteslutande i Java och körs som en fristående applikation. Närmare presentation av programmet återfinns i kapitel 4. Under den här rubriken kommer även demonstreras resultat av körningar för varje parsestrategi jag har implementerat. För att det skulle gå att uttyda bågarna på skärmdumparna från programmet var jag tvungen att behålla dess originalstorlek vilket gjorde att jag valde att presentera dem i form av bilagor.

Som indata hade jag

grammatiken: S->Np,Vp

Np->N

Vp->V,Np,Pp

Pp->P,Np

Vp->V,Np

Np->N,Pp

Np->Det,A,Np

lexikonet: Syntax: pojken
Label: N

Syntax: flickan
Label: N

Syntax: i
Label: P

Syntax: kiosken
Label: N

Syntax: pussar
Label: V

indatasträngen: pojken pussar flickan i kiosken

Topdown djupetförst (Se bilaga 2.1a).
Topdown breddenförst (Se bilaga 2.1b).

Bottomup djupetförst (Se bilaga 2.2a).
Bottomup breddenförst (Se bilaga 2.2b).

Bottomup à la Kilbury djupetförst (Se bilaga 2.3a).
Bottomup à la Kilbury breddenförst (Se bilaga 2.3b).

Topdown med bottomupfilter djupetförst (Se bilaga 2.4a).
Topdown med bottomupfilter breddenförst (Se bilaga 2.4b).

Bottomup med topdownfilter djupetförst (Se bilaga 2.5a).
Bottomup med topdownfilter breddenförst (Se bilaga 2.5b).

Bottomup á la Kilbury med topdownfilter djupetförst (Se bilaga 2.6a).
Bottomup á la Kilbury med topdownfilter breddenförst (Se bilaga 2.6b).

Bottomup á la Kilbury med bottomupfilter djupetförst (Se bilaga 2.7a).
Bottomup á la Kilbury med bottomupfilter breddenförst (Se bilaga 2.7b).

Bottomup á la Kilbury med bottomup & topdownfilter djupetförst (Se bilaga 2.8a).
Bottomup á la Kilbury med bottomup & topdownfilter breddenförst (Se bilaga 2.8b).

5.2 Begränsningar

Några av programmets begränsningar presenteras här.

Inga kortkommandon

Inmatning av regler, ord eller indata vid parsning, sker med tangentbordet all annan av interaktion med programmet sker med musen. Programmet svarar alltså inte på några kortkommandon.

En lösning i taget

ChartDeluxe har endast kapacitet att presentera lösningen från den senaste utförda parsningen. Programmet tillåter alltså inte att kompletta resultat från flera parningar samtidigt.

Spara parsningar

Programmet tillhandahåller inget sätt för användaren att spara resultatet från en parsning. Vill man dokumentera resultatet från parsningen får användaren antingen skriva ut det till en skrivare eller kopiera innehållet och klistra in dem i sitt favorit textredigerare.

Antal ord i indata

Det fält där indata skrivs in vid parsning kan endast visa 54 tecken i taget inklusive blanksteg. Och det blir svårt att få plats med bågarna i bågritaren om indata innehåller en sats med mer än sju ord. Det går men det blir lätt oöversiktligt. Problemet är strikt gränssnittsrelaterat.

5.3 Möjliga Utvidgningar

Här presenteras förslag till möjliga utvidgningar till programmet ChartDeluxe v 1.0.

Thistle

Ett tag var det meningen att ChartDeluxe skulle kopplas ihop med trädritningsprogrammet Thistle som på ett exemplariskt sätt kan presentera parseträd. Problemet var att jag hade kommit såpass långt i min utveckling av programmet att det skulle kräva omfattande arbete för att anpassa det till Thistle därför valde jag att avvakta med det.

Direktredigering

Det vore trevligt om användaren skulle kunna redigera lexikon och grammatiker direkt i det fönster de visas. Förfarnings sättet skulle vara detsamma som vid annan textredigering.

Statistik

Man skulle kunna tänka sig att programmet givet indata och ett valfritt antal parsestrategier skulle kunna presentera statistisk information över analyserna. Statistiken skulle kunna innehålla information om vilken strategi som innehåller flest respektive minst antal bågar, vilken typ av bågar de olika analyserna innehåller, hur lång tid de olika strategierna tog på sig osv.

Öppna filer skriven i ett annat programspråk

De användare programmet är avsett för kommer förmodligen komma i kontakt med flera olika programspråk under sin utbildning . Om hon/han skulle kunna använda grammatiker och/eller lexikon hon/han tidigare skrivit skulle de spara värdefull tid. Det vore även önskvärt att programmet skulle kunna erbjuda att användaren att spara lexikon och grammatik i valfritt programspråk eller som en textfil.

Behandla rekursiva särdrag

Programmet klarar endast av att behandla enkla särdragstrukturer men man skulle kunna tänka sig att man utvidgade regelstrukturen på ett sådant sätt att rekursiva särdragsrepresentationer skulle kunna behandlas.

5.3 Slutsatser

Det faktum att Java är objektorienterat gjorde att jag trots min ringa erfarenhet av området lyckades en modulär programstruktur. Det gjorde att det var lätt att få en översikt av programmet under utvecklingens gång. Vid en eventuell utveckling av programvaran kan användare i viss mån välja att byta ut hela komponenter av programmet utan att behöva utföra modifikationer på resten av programmet. Det enda stora problem jag ser med att utveckla program i Java är att programspråket är relativt långsamt ett problem jag tror kommer elimineras i takt med att snabbare Javakompilatorer utvecklas.

I slutet av arbetet med programmet ChartDeluxe slutförde jag kursen Objektorienterad Systemutvecklingen där jag fick tillfälle att arbeta praktiskt med områdena objektorienterade analys och design .Så här efteråt kan jag med säkerhet påstå att utvecklingen av ChartDeluxe v1.0 hade utförts mycket smidigare och koden hade blivit mycket mer strukturerade om jag hade haft dessa kunskaper innan jag startade med examensarbetet.

Slutligen måste jag påpeka att utvecklingen av programmet har varit mycket lärorik. Eftersom som jag själv utvecklat programmet har jag arbetat med många skilda områden inom programmering och har haft möjlighet i stor grad utforma programmet efter eget huvud något som har varit både spännande och roligt.

5.4 Sammanfattning

Min uppgift var att utveckla ett program som skall presentera hur olika chartparsestrategier fungerar. Tanken är att programmet skall användas i utbildningen på Datalingvistprogrammet på Göteborgs universitet. Programmet ChartDeluxe v1.0 erbjuder användaren att välja mellan flera olika parsestrategier och presenterar den graf som blir resultat av parsningen grafiskt.
Wirén(1992) presenterar en mängd strategier från vilka jag valde att implementera en delmängd.

- Topdown
- Bottomup
- Bottomup à la Kilbury
- Topdown med bottomupfilter
- Bottomup med topdownfilter
- Bottomup á la Kilbury med topdownfilter
- Bottomup á la Kilbury med bottomupfilter
- Bottomup á la Kilbury med bottomup & topdownfilter

Programmet ger även användaren möjlighet att skapa, redigera och administrera egna lexikon och grammatiker. Interaktionen mellan program och användaren sker framförallt genom datorns mus. Eftersom programmet är implementerat i Java är det plattformsoberoende går det att köra på en dator med valfritt operativsystem installerat.

6. Källförtäckning

Campione Mary och Walrath Kathy .(1998). *The Java Tutorial Second Edition object -Oriented Programming for the internet.* Massachusetts: Addison Wesley Longman inc.

Flanagan David (1997) *Java in a nutshell second edition.* Sebastopol, O´Reilly & Associates inc.

Gazdar Gerald & Mellish Chris.(1989).*Natural Language Processing in PROLOG An Introduction to Computional Lingustics.* Massachusetts: Addison Wesley Longman inc.

Wirén Mats (1992) *Studies incremental natural-language analysis.* Linköping: Linköping University

Mathiassen Lars, Munk-Madsen Andreas, Nielsen Peter Axel och Stage Jan(1998) *Objektorienterad analys och design.* Lund: Studentlitteratur

Bilaga 1

Referensmanual till klasser

I avdelning finner användaren en kortfattad referensmanual över alla klasser och paket som ingår i programmet ChartDeluxe v1.0 observera att privata klasser konstanter och variabler inte finns med men den fullständiga API:n går att nå man från http:\\www.cling.gu.se\~cl5psven\xjobb\api.html.

package cling.cl5psven.chart

cling.cl5psven.chart.Arc

```
public abstract class Arc
//Allmänna instans metoder
   public int getArcnumb();
   public int getFinish();
   public Vector getFound();
   public int getStart();
   public StringBuffer getSyntax();
   public Vector getToFind();
   public boolean isEmpty();
   public String toString();
//Skyddade instans metoder
   protected void addToFind(Object o);
   protected void addToFindAt(Object o, int i);
   protected boolean arcExists(Vector arcs);
   protected String getLabel();
   protected void foundIt();
   protected void insertFoundAt(Object o, int i);
   protected boolean isNext(Arc arc);
   protected boolean isActive();
   protected void setActive(boolean isActive);
   protected void setArcnumb(int inArcnumb);
   protected void setFinish(int inFinish);
   protected void setFound(Vector found);
   protected void setLabel(String inString);
   protected void setStart(int inStart);
   protected void setSyntax(String inSyntax);
   protected void setSyntax(StringBuffer inSyntax);
   protected void setToFind(Vector find);
```

cling.cl5psven.chart.ChartParser

```
public class ChartParser {
//PublicConstructors
   public ChartParser();
//Public Instance Methods
   public String toString();
//Protected  Instance Methods
   protected Vector giveArcs();
   protected int howMany();
   protected void InitializeComplex(Vector words,
   cling.cl5psven.chart.ChartLexicon lexicon);
   protected letsParse(int strategy, boolean depthfirst,
   cling.cl5psven.chart.ChartGrammar gram); }
```

cling.cl5psven.chart.ChartGrammar

```
   public class ChartGrammar {
//Publika konstruerare
   public ChartGrammar(cling.cl5psven.chart.ChartGrammar grammar);
public ChartGrammar(String name);
public ChartGrammar(String name, cling.cl5psven.chart.ChartRule aRule);
```

```
public ChartGrammar(String name, Vector theRules);
//Public Instance Methods
   public void addRule(ChartRule aRule);
   public void deleteLastRule();
   public boolean deleteRuleNo(int i);
   public java.lang.String getName();
   public java.lang.String toString();
//Protected Instance Methods
   protected java.lang.StringfindStartSymb();
   protected java.util.Vector allRulesLabeled(java.lang.String cat);
   protected java.util.Vector generateEArc(ComplexArc arc);
   protected java.util.Vector generateLeftCArc(ComplexArc arc);
   protected java.util.Vector generateLeftCArc(ComplexArc Inarc, boolean
kilbury);
   protected java.util.Vector generateSArcs(int howmany);
   protected boolean leadsTo(java.lang.String lookahead, Arc inArc);
   protected boolean leadsTo(java.util.Vector categories, Arc anArc);
   protected void setName(java.lang.String inName);
}
```

cling.cl5psven.chart.ChartLexicon

```
public class ChartLexicon{
//Publika konstruerare
   public ChartLexicon(String LexiconName);
   public ChartLexicon(String lexiconName, Word aword);
   public ChartLexicon(String lexiconName, int lexiconSize, Vector
   inWords);
//Public Instance Methods
   public void addWord(Word aword);
   public void deleteAllWords();
   public boolean deleteWordNo(int number);
   public Word firstWord();
   public Vector initComplexCheck(String aword);
   public void setName(java.lang.String inName);
   public String toString();
}
```

cling.cl5psven.chart.ChartRule

```
public class ChartRule{
//Publika konstruerare
   public ChartRule();
   public ChartRule(ChartRule rule);
   public ChartRule(String lefthandside, Vector righthandside);
   public ChartRule(java.lang.String lefthandside, java.util.Vector right,
   java.util.Vector strings);
//Public Instance Methods
   public java.util.Vector getFeat();
   public java.lang.String getLabel();
   public java.lang.String getLeftCorner();
   public java.util.Vector getRhs();
   public java.lang.String toString();
}
```

cling.cl5psven.chart.ComplexArc

```
public class ComplexArc
//Publika konstruerare
   public ComplexArc();
   public ComplexArc(ComplexArc inArc);
   public ComplexArc(ChartRule rule);
   public ComplexArc(int start, int finish, int anumber, ComplexWord
   aword);
```

```
    public ComplexArc(int start, int finish, String label, Vector found,
    Vector tofind, boolean active, Vector features);
 //Public Instance Methods
    public boolean arcExists(java.util.Vector arcs);
    public java.util.Vector getFeatures();
    public java.util.Vector getUnify();
    public void insertFeatureAt(java.util.Hashtable t, int i);
    public boolean isNext(ComplexArc inArc);
    public void setFeatures(java.util.Vector v);
    public boolean subsumes(ComplexArc inArc);
    public java.lang.String toString();
    public boolean unify(ComplexArc inArc);
 //Protected Instance Methods
    protected void foundIt(ComplexArc inArc);
    protected void setUnify(java.util.Vector inUnify);
    protected void setUnifyAt(java.util.Hashtable t, int k);
    protected void updateKillbury(ComplexArc inArc);
 protected void updateFeat(ComplexArc inArc);
 }
```

cling.cl5psven.chart.ComplexWord

```
public class ComplexWord extends Words implements java.io.Serializable{
//Publika konstruerare
   public ComplexWord(ComplexWord aComplexWord);
   public ComplexWord(String label, String syntax);
   public ComplexWord(String label, String syntax, Hashtable features);
//Public Instance Methods
   public java.util.Hashtable getFeatures();
   public java.lang.String toString();

}
```

cling.cl5psven.chart.Lookahead

```
public class LookaheadTable{
//Allmänna konstruerare
   protected LookaheadTable();
//Skyddade metoder
   protected void addPost(Arc anArc);
   protected java.util.Vector showCategories(int position);
}
```

cling.cl5psven.chart.Word

```
abstract class Word implements java.io.Serializable{
// Allmänna metoder
   public java.lang.String getLabel();
   public java.lang.String getSyntax();
   public java.lang.String toString();
}
```

```
package cling.cl5psven.chart.ui

cling.cl5psven.chart.ui.ArcIFrame
public class ArcIFrame
   protected ArcIFrame(int width, int height);
   protected void changeLanguage(int language) ;
   protected void empty();
   protected cling.cl5psven.chart.ArcInfoIFrame getFrames();
   protected void isAllClear() ;
   protected void printIt();
   protected void setArcs(Vector inArcs, Vector words);
   protected void setThemeNo(int number);

cling.cl5psven.chart.ui.ArcInfoIFrame
public class ArcInfoIFrame
//Publika konstruerare
   public ArcInfoIFrame(String titel);

cling.cl5psven.chart.ui.DesktopLabels
public class DesktopLabels {
// Skyddade fält
   protected java.lang.String[][] mainColourItems;
   protected java.lang.String[][] mainFileMItems;
   protected java.lang.String[][] mainHelpItems;
   protected java.lang.String[][] mainMenues;
   protected java.lang.String[][] mainPrefMItems;
   protected java.lang.String[][] mainWindowMItems;
//Allmänna Konstruerare
   public DesktopLabels();
}

cling.cl5psven.chart.ui.DialogLabels
public class DesktopLabels {
// Skyddade fält
   protected java.lang.String[][] addCont;
   protected java.lang.String[][][] addTexts;
   protected java.lang.String[][] addTitles;
   protected java.lang.String[][] dataMenues ;
   protected java.lang.String[][][] dataMenuI;
   protected java.lang.String[][] dataTitles;
   protected java.lang.String[][][] Deltexts;
//Allmänna Konstruerare
   public DialogLabels();
}

cling.cl5psven.chart.ui.MainDesktop
public class MainDesktop extends javax.swing.JFrame
implements java.awt.print.Printable{
//Allmänna konstruerare
   public MainDesktop(String);
//Allmänna Metoder
   public void paintComponent(java.awt.Graphics g)
   public int  print(java.awt.Graphics g, java.awt.print.PageFormat pf,
                                                    int pi):
}
```

```
cling.cl5psven.chart.ui.ParseIFrame
public class ParseIFrame extends javax.swing.JinternalFrame{
//Allmänna konstruerare
  public ParseIFrame(int height, int width, int language)
//Skyddade metoder
  protected void changeLanguage(int language);
  protected void drawArcs(java.awt.Graphics g)
  protected void emptyGrammar();
  protected void emptyLexicon();
  protected void myTokenizer(javax.swing.JTextField tf);
  protected void setGrammar(ChartGrammar inGram);
  protected void setLexicon(ChartLexicon inLex);
}
```

package cling.cl5psven.util

```
cling.cl5psven.util.About
public class About extends JFrame{
//Allmänna konstruerare
  public About(String information, String title);
}

cling.cl5psven.util.FifoStack
public class FifoStack extends ChartStack{
//Publika konstruerare
  public FifoStack();
//Public Instance Methods
  public boolean isEmpty();
  public synchronized Object pop();
  public Object push(Object item);
  public int size();
//Skyddade instans metoder
  protected void finalize();
}

cling.cl5psven.util.HtmlFrame
public class HtmlFrame extends JFrame{
//Allmänna konstruerare
public HtmlFrame(String adress);
}

cling.cl5psven.util.LifoStack
public class LifoStack extends ChartStack{
//Publika konstruerare
  public LifoStack();
//Public Instance Methods
  public boolean isEmpty();
  public synchronized Object pop();
  public Object push(Object item);
  public int size();
  protected void finalize();
}

cling.cl5psven.util.MyTokenizer
public class MyTokenizer{
//Allmänna klass metoder
public static Vector simplerTniser(String temp, String del);
  public static Hashtable wordFeatTnizer(String temp);
}
```

```
cling.cl5psven.util.Stack
abstract class Stack {
//Public Instance Methods
  public boolean isEmpty();
  public synchronized Object pop();
  public Object push(Object item);
  public int size();
  protected void finalize();
}
```

```
cling.cl5psven.util.StringPrinter
public class StringPrinter{
//Allmänna klass metoder
public static void linePrinter(Graphics g, String text);
}
```

package cling.cl5psven.util.metalthemes

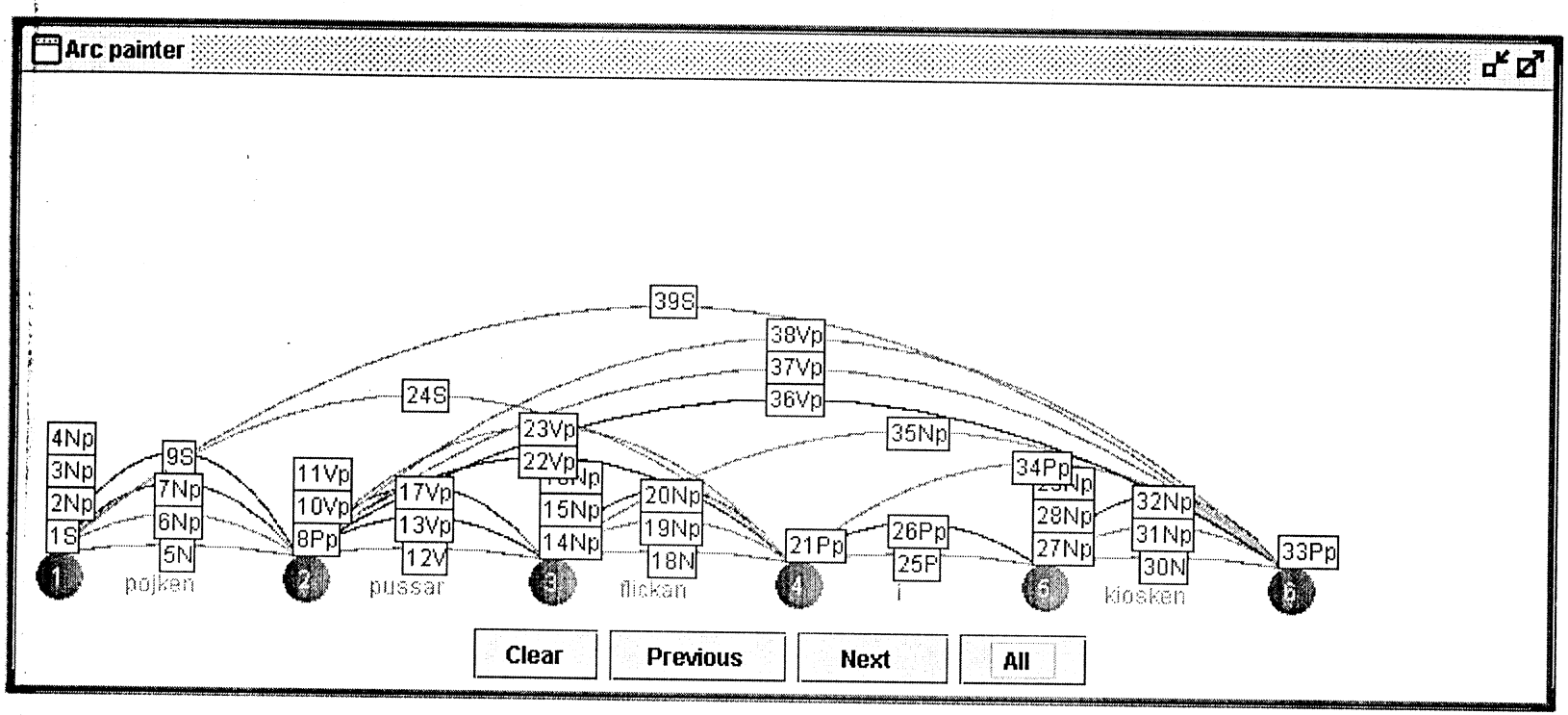
```
cling.cl5psven.util.metalthemes.Bluetheme
public class Bluetheme extends DefaultMetaltheme{
//Allmänna konstruerare
  public Bluetheme();
//Allmänna metoder
  public String getName() ;
//Skyddade metoder
  protected ColorUIResource getPrimary1();
  protected ColorUIResource getPrimary2();
  protected ColorUIResource getPrimary3();
  protected ColorUIResource getSecondary1();
  protected ColorUIResource getSecondary2();
  protected ColorUIResource getSecondary3();
}
```

```
cling.cl5psven.util.metalthemes.Greentheme
public class Greentheme extends DefaultMetaltheme{
//Allmänna konstruerare
  public Greentheme();
//Allmänna metoder
  public String getName() ;
//Skyddade metoder
  protected ColorUIResource getPrimary1();
  protected ColorUIResource getPrimary2();
  protected ColorUIResource getPrimary3();
  protected ColorUIResource getSecondary1();
  protected ColorUIResource getSecondary2();
  protected ColorUIResource getSecondary3();
}
```

```
cling.cl5psven.util.metalthemes.Redtheme
public class Redtheme extends DefaultMetaltheme{
//Allmänna konstruerare
  public Greentheme
//Allmänna metoder
  public String getName() ;
//Skyddade metoder
  protected ColorUIResource getPrimary1();
  protected ColorUIResource getPrimary2();
  protected ColorUIResource getPrimary3();
  protected ColorUIResource getSecondary1();
  protected ColorUIResource getSecondary2();
  protected ColorUIResource getSecondary3();
```

Bilaga 2.1a

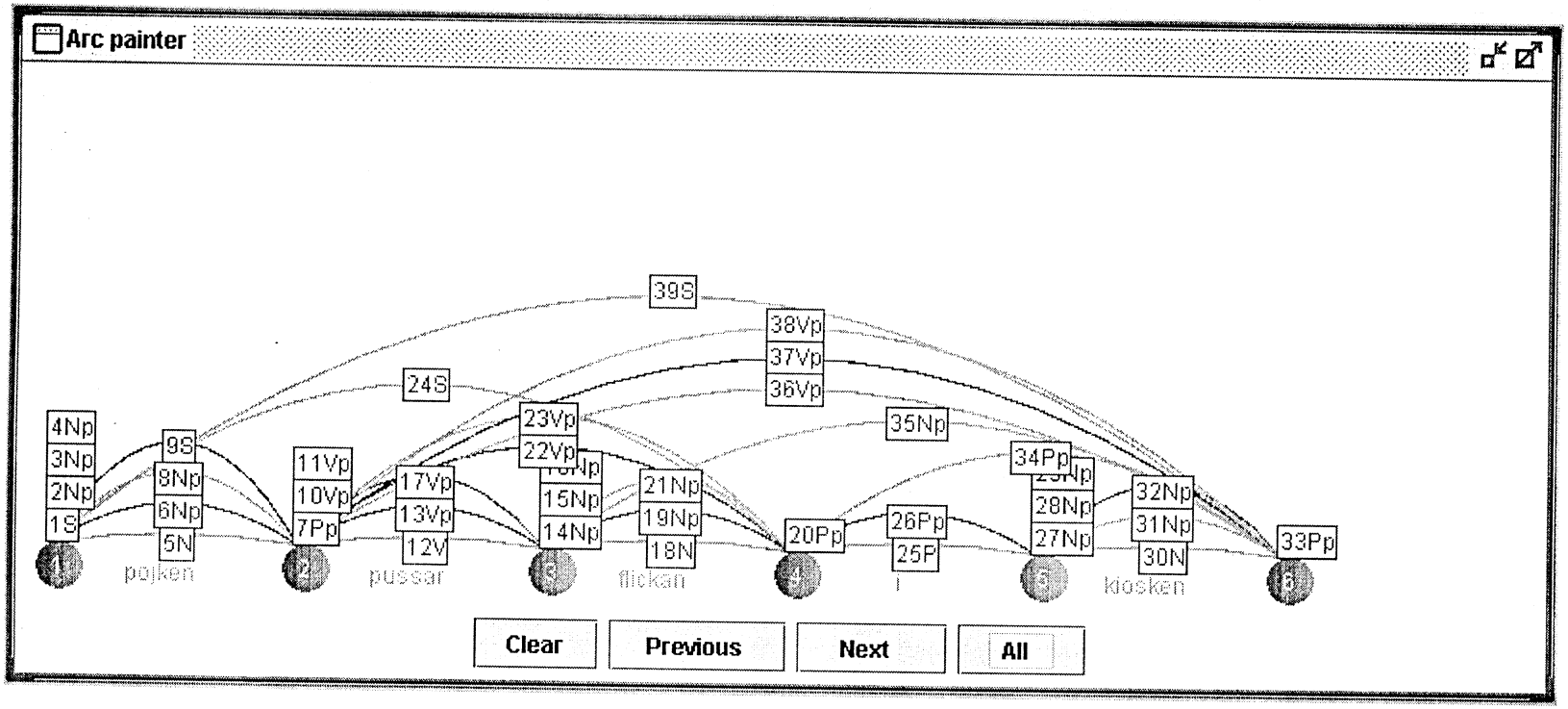
Topdown djupetförst



1 S->. Np Vp
2.Np->.N
3.Np->.N Pp
4.Np->.Det A Np
5.N-> pojken.
6.Np->N.
7. Np->N. Pp
8. Pp ->.P Np
9. S -> Np.Vp
10. Vp ->.V NpPp
11. Vp ->.V Np
12. V->pussar.
13.Vp->V.NpPp
14.Np->.N
15.Np->.NPp
16.Np->.DetANp
17.Vp->V.Np
18.N->flickan.
19.Np->N.Pp
20.Pp->.P Np
21.Np->N.
22.Vp->V Np.Pp
23.Vp->V Np.
24.S->NpVp.
25.P->i.
26.Pp->P.Np
27.Np->.N
28.Np->.N Pp
29.Np->.Det A Np
30.N ->kiosken.
31.Np->N.
32.Np->N.Pp
33.Pp->.P Np
34.Pp->P Np.
35.Np->N Pp.
36.Vp->V Np.
37.Vp->V Np.Pp
38.Vp->V Np Pp.
39.S -> Np Vp.

Bilaga 2.1b

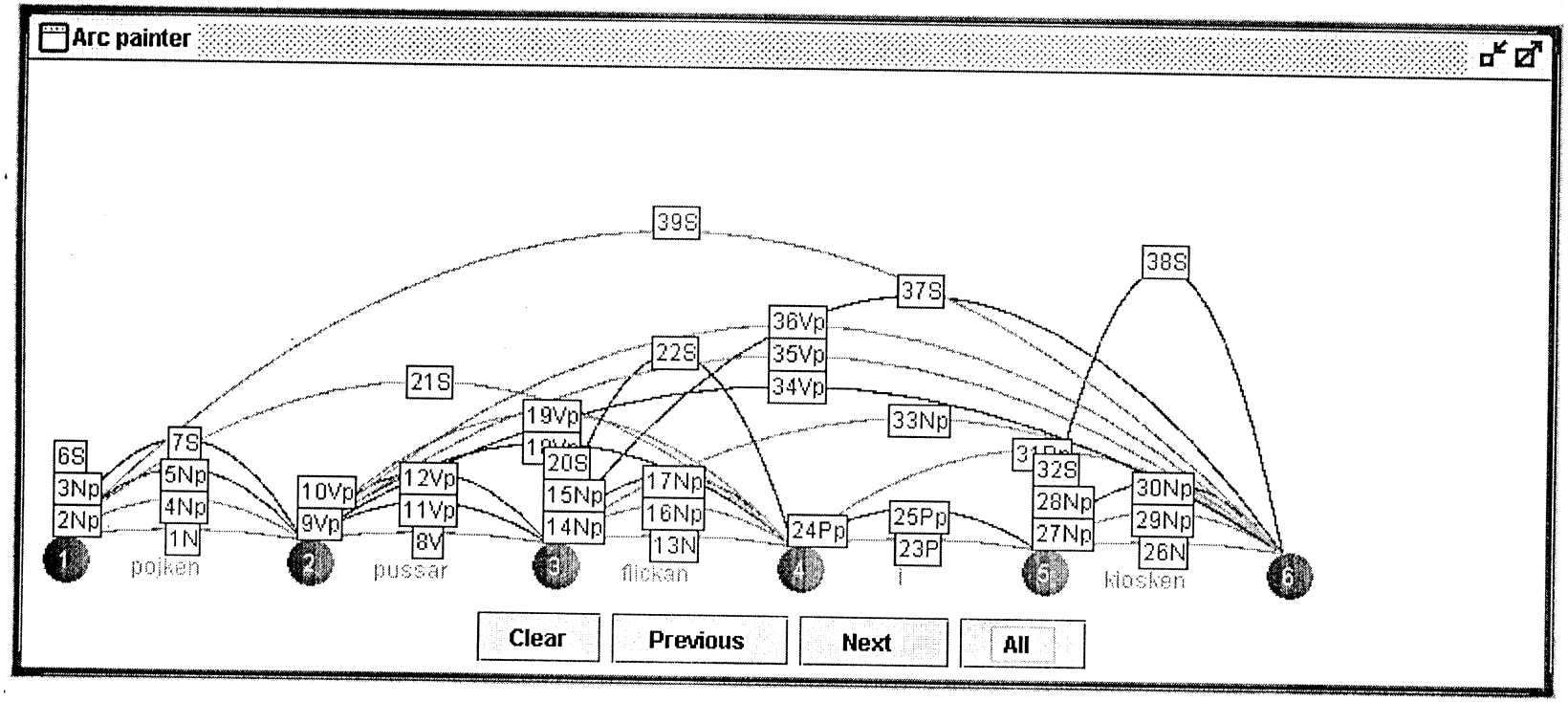
Topdown breddenförst



1 S->. Np Vp
2.Np->.N
3.Np->.N Pp
4.Np->.Det A Np
5.N-> pojken.
6.Np->N. Pp
7.Pp ->.P Np
8 Np->N.
9.S->Np.Vp
10.Vp->.V Np Pp
11.Vp->.V Np
12.V-> pussar.
13.Vp->V.Np Pp
14.Np->.N
15.Np->.N Pp
16.Np->.Det A Np
17.Vp->V.Np
18.N->flickan.
19.Np->N.Pp
20.Pp->.P Np
21.Np->N.
22.Vp->V Np.Pp
23.Vp->V Np.
24.S->Np Vp.
25.P->i.
26.Pp->P.Np
27.Np->.N
28.Np->.N Pp
29.Np->.Det A Np
30.N->kiosken.
31.Np->N.
32.Np->N.Pp
33.Pp->.P Np
34.Pp->P Np.
35.Np->N Pp.
36.Vp->V Np.
37.Vp->V Np.Pp
38.Vp->V Np Pp.
39.S-> Np Vp.

Bilaga 2.2a

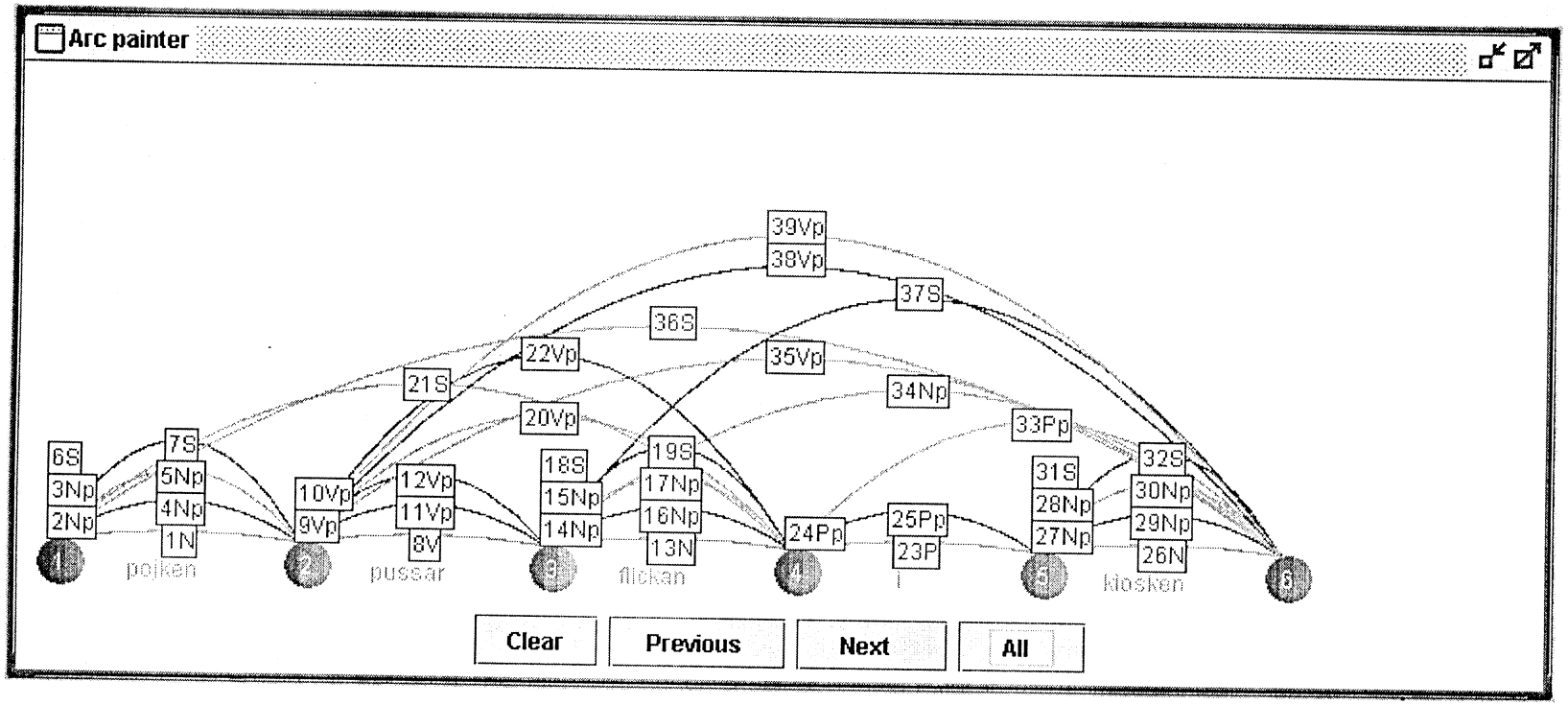
Bottomup depthfirst



1.N-> pojken.
2. Np->.N
3.Np-> .N Pp
4. Np->N.
5.Np->N.Pp
6.S->.Np Vp
7.S->Np.Vp
8.V-> pussar.
9.Vp->.V Np Pp
10.Vp->.V Np
11.Vp->V.Np Pp
12.Vp->V.Np
13.N->flickan.
14.Np->.N
15.Np->.NPp
16.Np->N.
17.Np->N.Pp
18.Vp->V Np.Pp
19.Vp->V Np.
20.S->.Np Vp
21.S->Np Vp.
22.S->Np.Vp
23.P->i.
24.Pp->.PNp
25.Pp->P.Np
26.N->kiosken.
27.Np->.N
28.Np->.N Pp
29.Np->N.
30.Np->N.Pp
31.Pp->P Np.
32.S->.Np Vp
33.Np->N Pp.
34.Vp->V Np.Pp
35.Vp->V Np Pp.
36.Vp->V Np.
37.S->Np.Vp
38.S->Np.Vp
39.S->Np Vp.

Bilaga 2.2b

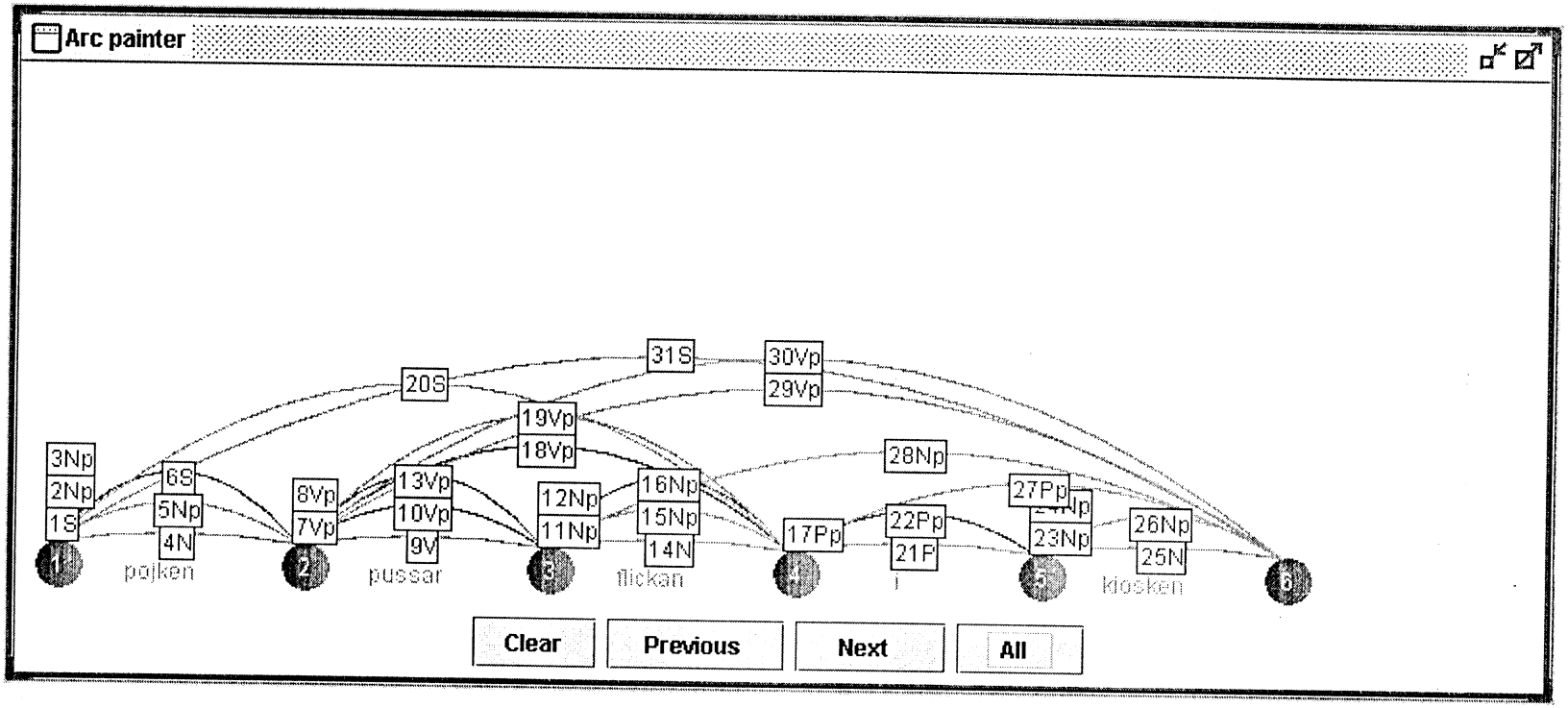
Bottomup breadthfirst



1.N-> pojken.
2. Np->.N
3.Np-> .N Pp
4. Np->N.
5.Np->N.Pp
6.S->.Np Vp
8.V->pussar.
9- Vp->.V Np Pp
10.Vp->.VNp
11.Vp->V.Np
12.Vp->V.Np Pp
13.N->flickan.
14.Np->.N
15.Np->.N Pp
16.Np->N.Pp
17.Np->N.
18.S->.NpVp
19.S->Np.Vp
20.Vp->V Np.
21.S->Np Vp.
22.Vp->V Np.Pp
23.P->i.
24.Pp->.P Np
25.Pp->P.Np
26.N->kiosken.
27.Np->.N
28.Np->.N Pp
29.Np->N.Pp
30.Np->N.
31.S->.Np Vp
32.S->Np.Vp
33.Pp->P Np.
34.Np->N Pp.
35.Vp->V Np.
36.S->Np Vp.
37.S->Np.Vp
38.Vp->V Np.Pp
39.Vp->V Np Pp

Bilaga 2.3a

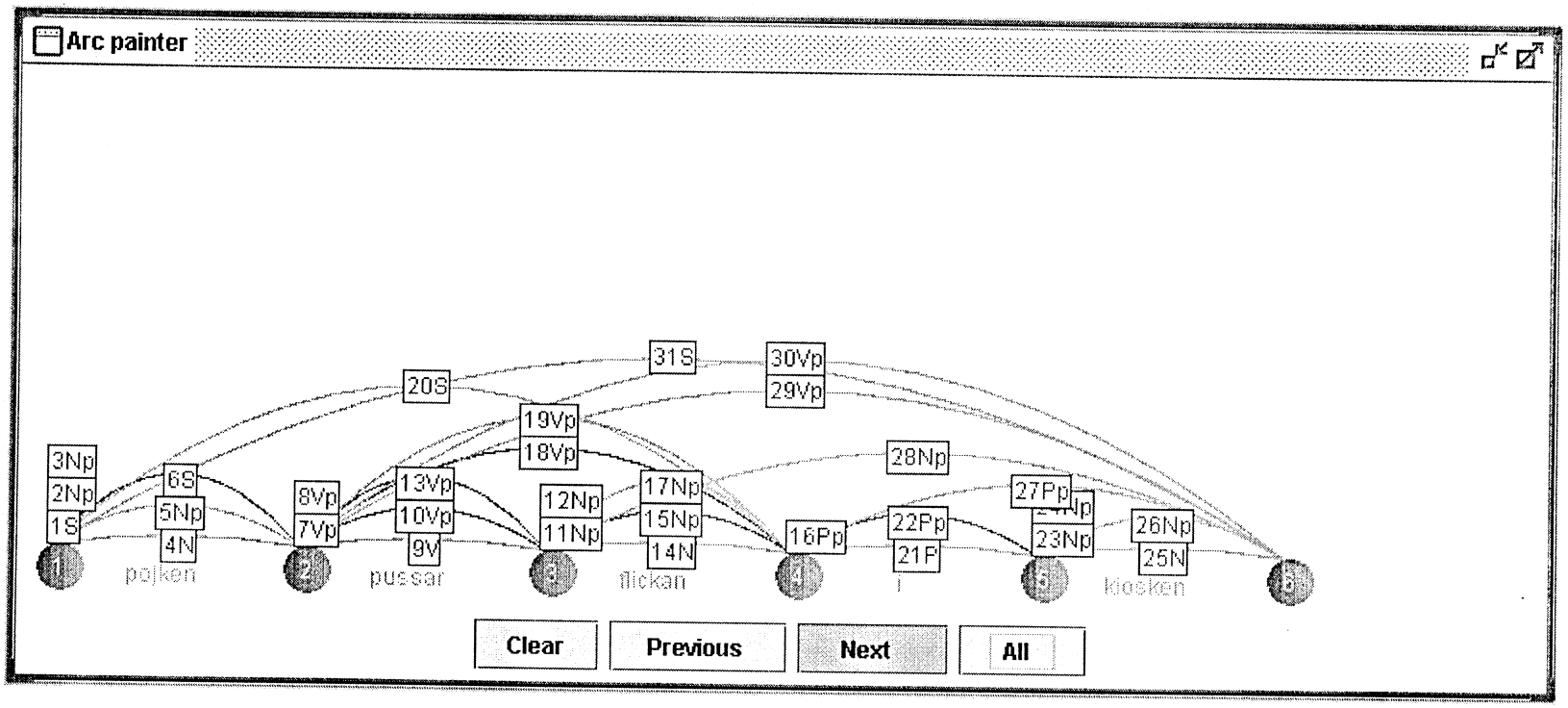
Topdown med bottomupfilter depthfirst



1.S->.Np Vp
2.Np->.N
3.Np->.N Pp
4.N->pojken.
5.Np->N.
6.S->Np.Vp
7.Vp->.V Np Pp
8.Vp->.V Np
9.V->pussar.
10.Vp->V.Np Pp
11.Np->.N
12.Np->.N Pp
13.Vp->V.Np
14.N->flickan.
15.Np->N.
16.Np->N.Pp
17.Pp->.P Np
18.Vp->V Np.Pp
19.Vp->V Np.
20.S->Np Vp.
21.P->i.
22.Pp->P.Np
23.Np->.N
24.Np->.N Pp
25.N->kiosken.
26.Np->N.
27.Pp->P Np.
28.Np->N Pp.
29.Vp->V Np Pp.
30.Vp->V Np.
31.S->Np Vp

Bilaga 2.3b

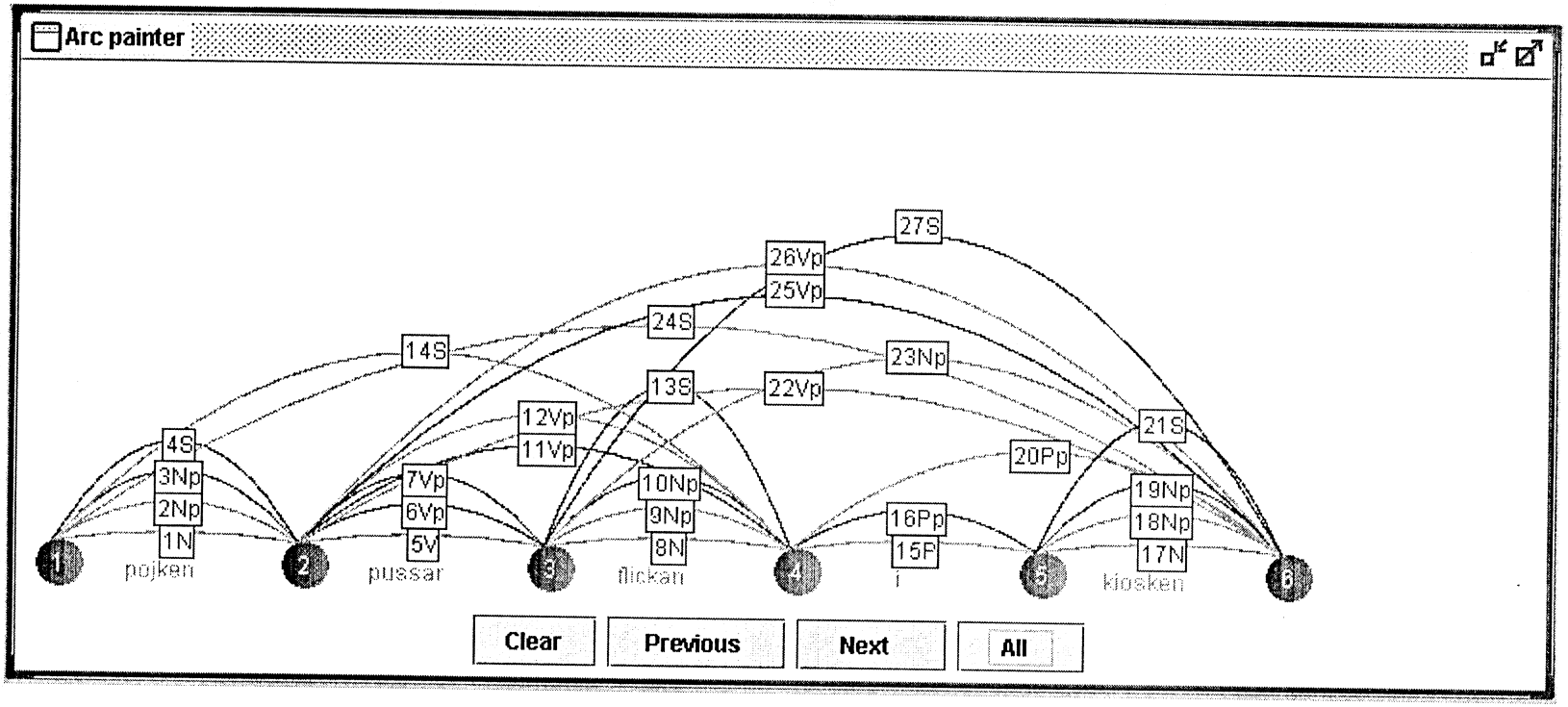
Topdown med bottomupfilter breddenförst



1.S->.Np Vp
2.Np->.N
3.Np->.N Pp
4.N->pojken.
5.Np->N.
6.S->Np.Vp
7.Vp->.V Np Pp
8.Vp->.V Np
9.V->pussar.
10.Vp->V.Np Pp
11.Np->.N
12.Np->.N Pp
13.Vp->V.Np
14.N->flickan.
15.Np->N.Pp
16.Pp->.P Np
17.Np->N.
18.Vp->V Np.Pp
19.Vp->V Np.
20.S->Np Vp.
21.P->i.
22.Pp->P.Np
23.Np->.N
24.Np->.N Pp
25.N->kiosken.
26.Np->N.
27.Pp->P Np.
28.Np->N Pp.
29.Vp->V Np.
30.Vp->V Np Pp.
31.S->Np Vp.

Bilaga 2.4a

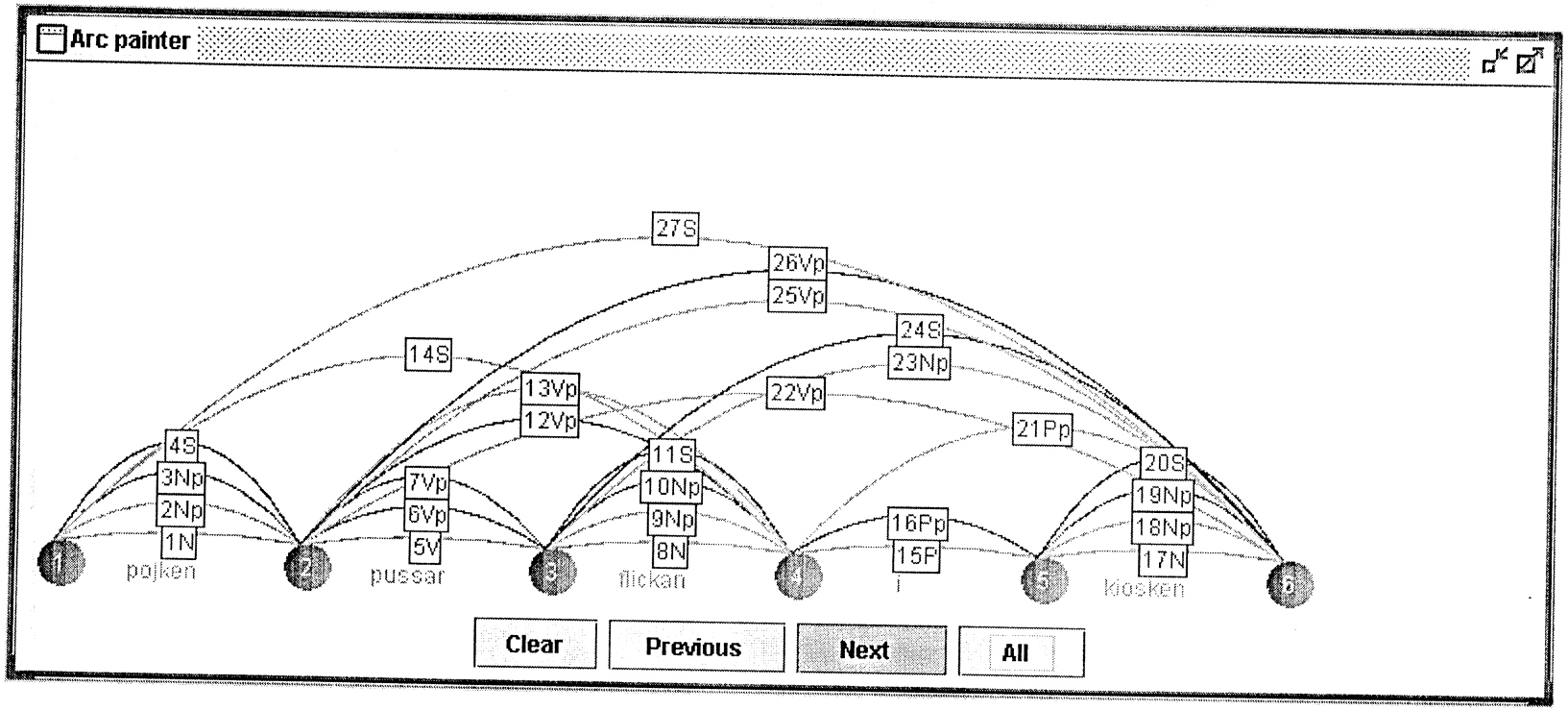
Bottomup á la Kilbury djupetförst



1.N->pojken.
2.Np->N.
3.Np->N.Pp
4.S->Np.Vp
5.V->pussar.
6.Vp->V.Np Pp
7.Vp->V.Np
8.N->flickan.
9.Np->N.
10.Np->N.Pp
11.Vp->V Np.Pp
12.Vp->V Np.
13.S->Np.Vp
14.S->Np Vp.
15.P->i.
16.Pp->P.Np
17.N->kiosken.
18.Np->N.
19.Np->N.Pp
20.Pp->P Np.
21.S->Np.Vp
22.Vp->V Np Pp.
23.Np->N Pp.
24.S->Np Vp.
25.Vp->V Np.Pp
26.Vp->V Np.
27.S->Np.Vp

Bilaga 2.4b

Bottomup á la Kilbury breddenförst



1.N->pojken.
2.Np->N.
3.Np->N.Pp
4.S->Np.Vp
5.V->pussar.
6.Vp->V.NpPp
7.Vp->V.Np
8.N->flickan.
27.S->NpVp.

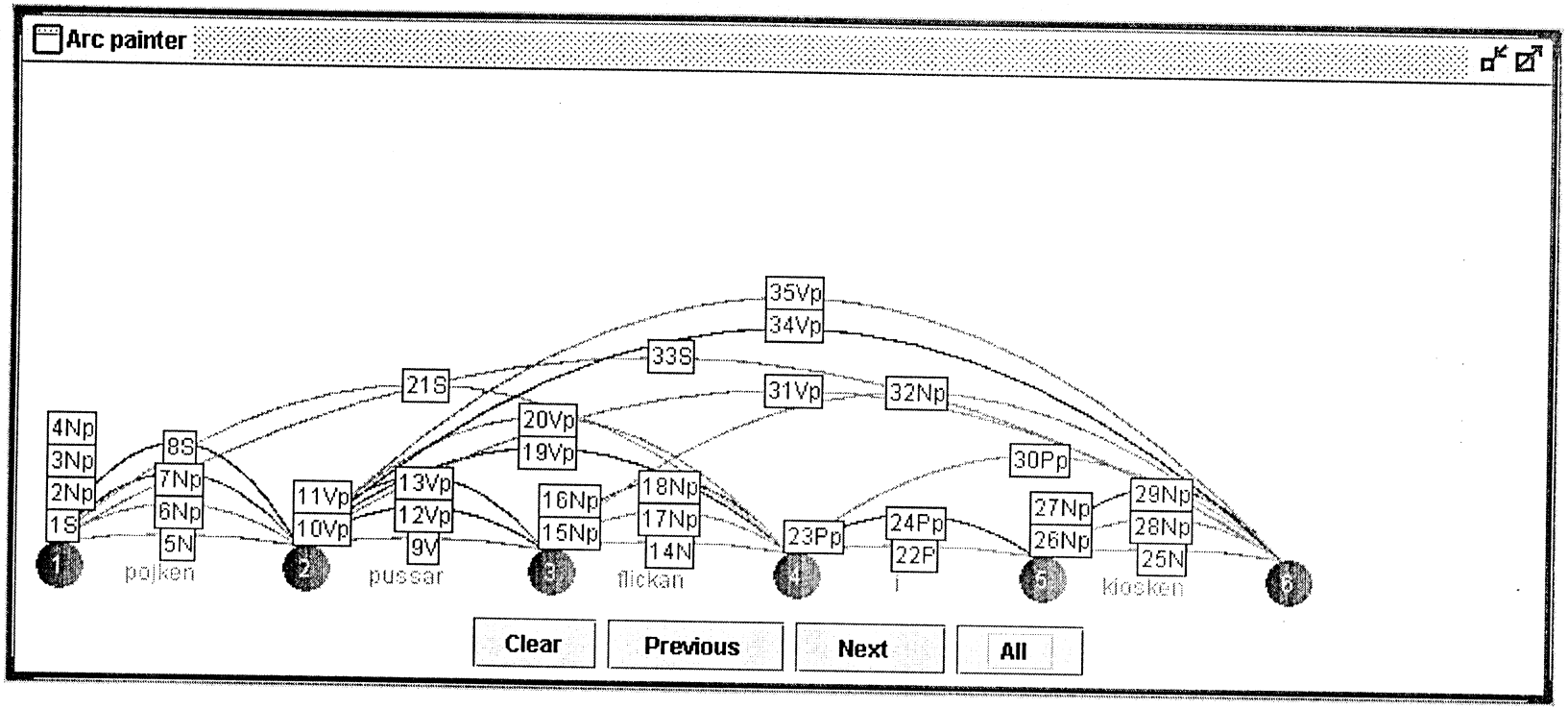
9.Np->N.
10.Np->N.Pp
11.S->Np.Vp
12.Vp->V Np.Pp
13.Vp->V Np.
14.S->Np Vp.
15.P->i.
16.Pp->P.Np

17.N->kiosken.
18.Np->N.
19.Np->N.Pp
20.S->Np.Vp
21.Pp->P Np.
22.Vp->V Np Pp.
23.Np->N Pp.
24.S->Np.Vp

25.Vp->V Np.
26.Vp->V Np.Pp
27.S->Np Vp.

Bilaga 2.5a

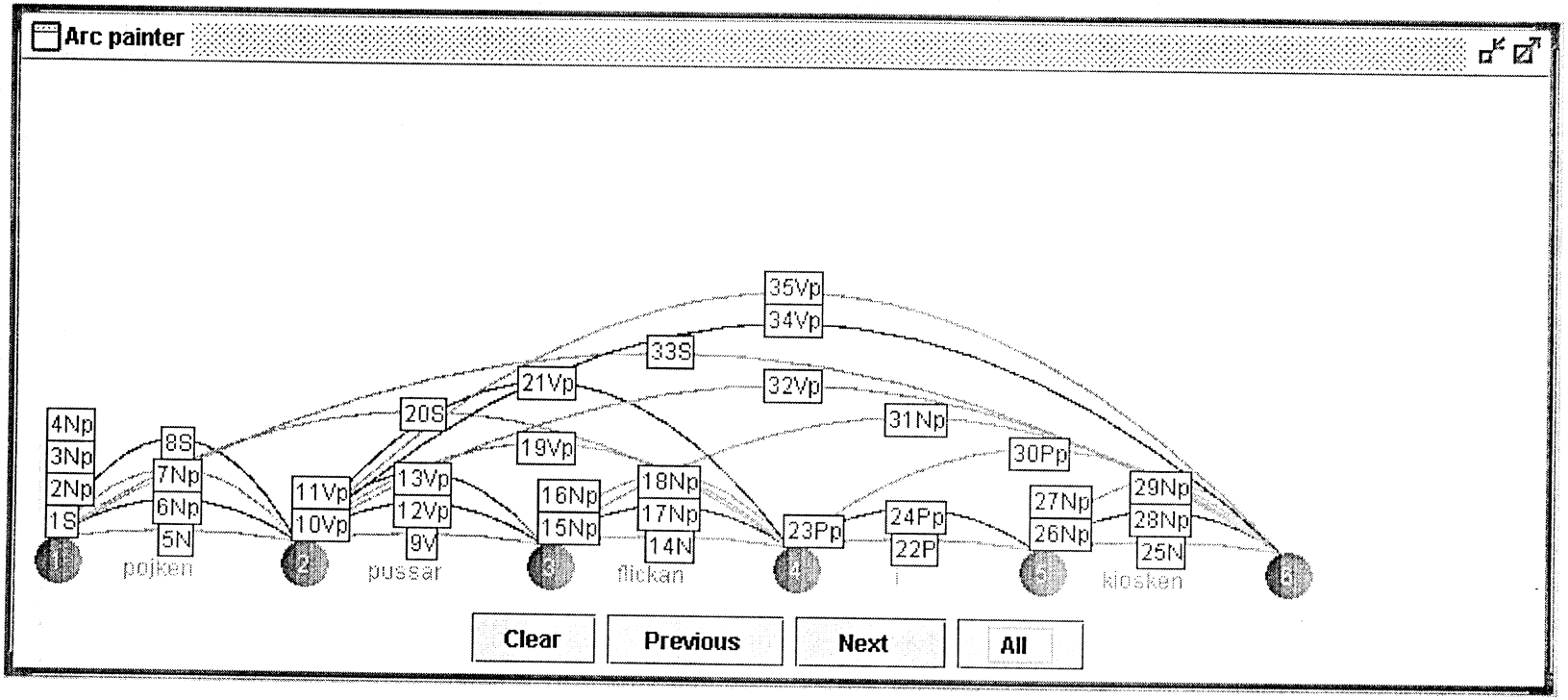
Bottomup med topdownfilter djupetförst



1. S-> .Np Vp
2. Np->.N
3.Np->.N Pp
4.Np->.Det A Np
5.N->pojken.
6.Np->N.
7.Np->N.Pp
8.S->Np.Vp
9.V->pussar.
10.Vp->.VNpPp
11.Vp->.VNp
12.Vp->V.NpPp
13.Vp->V.Np
14.N->flickan.
15.Np->.N
16.Np->.NPp
17.Np->N.
18.Np->N.Pp
19.Vp->VNp.Pp
20.Vp->VNp.
21.S->NpVp.
22.P->i.
23.Pp->.PNp
24.Pp->P.Np
25.N->kiosken
26.Np->.N
27.Np->.NPp
28.Np->N.
29.Np->N.Pp
30.Pp->PNp.
31.Vp->VNpPp.
32.Np->NPp.
33.S->NpVp.
34.Vp->VNp.Pp
35.Vp->VNp.

Bilaga 2.5b

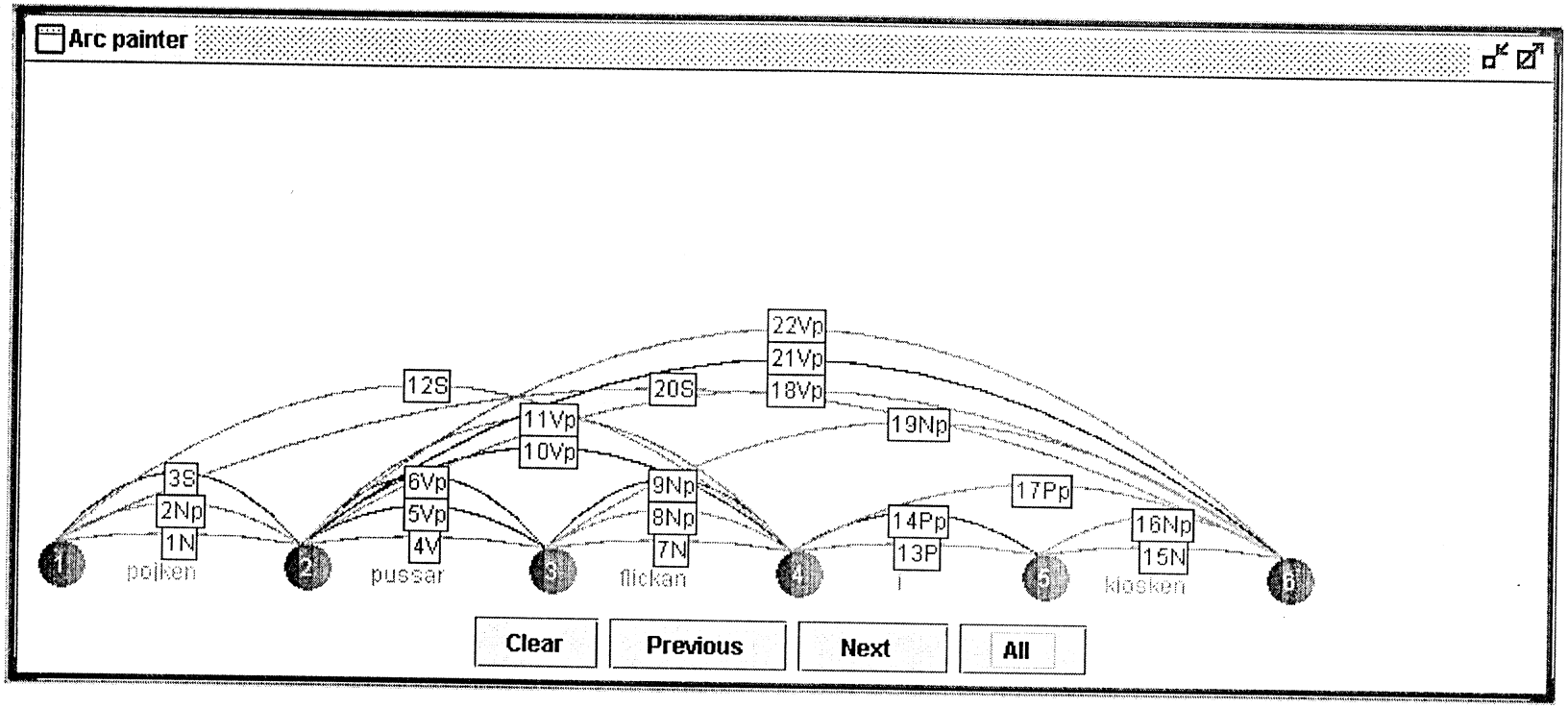
Bottomup med topdownfilter breddenförst



1. S-> .Np Vp
2. Np->.N
3.Np->.N Pp
4.Np->.Det A Np
4.Np->.Det A Np
5.N->pojken.
6.Np->N.Pp
7.Np->N.
8.S->Np.Vp
9.V->pussar.
10.Vp->.V Np Pp
11.Vp->.V Np
12.Vp->V.Np
13.Vp->V.Np Pp
14.N->flickan.
15.Np->.N
16.Np->.N Pp
17.Np->N.Pp
18.Np->N.
19.Vp->V Np.
20.S->Np Vp.
21.Vp->V Np.Pp
22.P->i.
23.Pp->.P Np
24.Pp->P.Np
25.N->kiosken.
26.Np->.N
27.Np->.N Pp
28.Np->N.Pp
29.Np->N.
30.Pp->P Np.
31.Np->N Pp.
32.Vp->V Np.
33.S->Np Vp.
34.Vp->V Np.Pp
35.Vp->V Np Pp.

Bilaga 2.6a

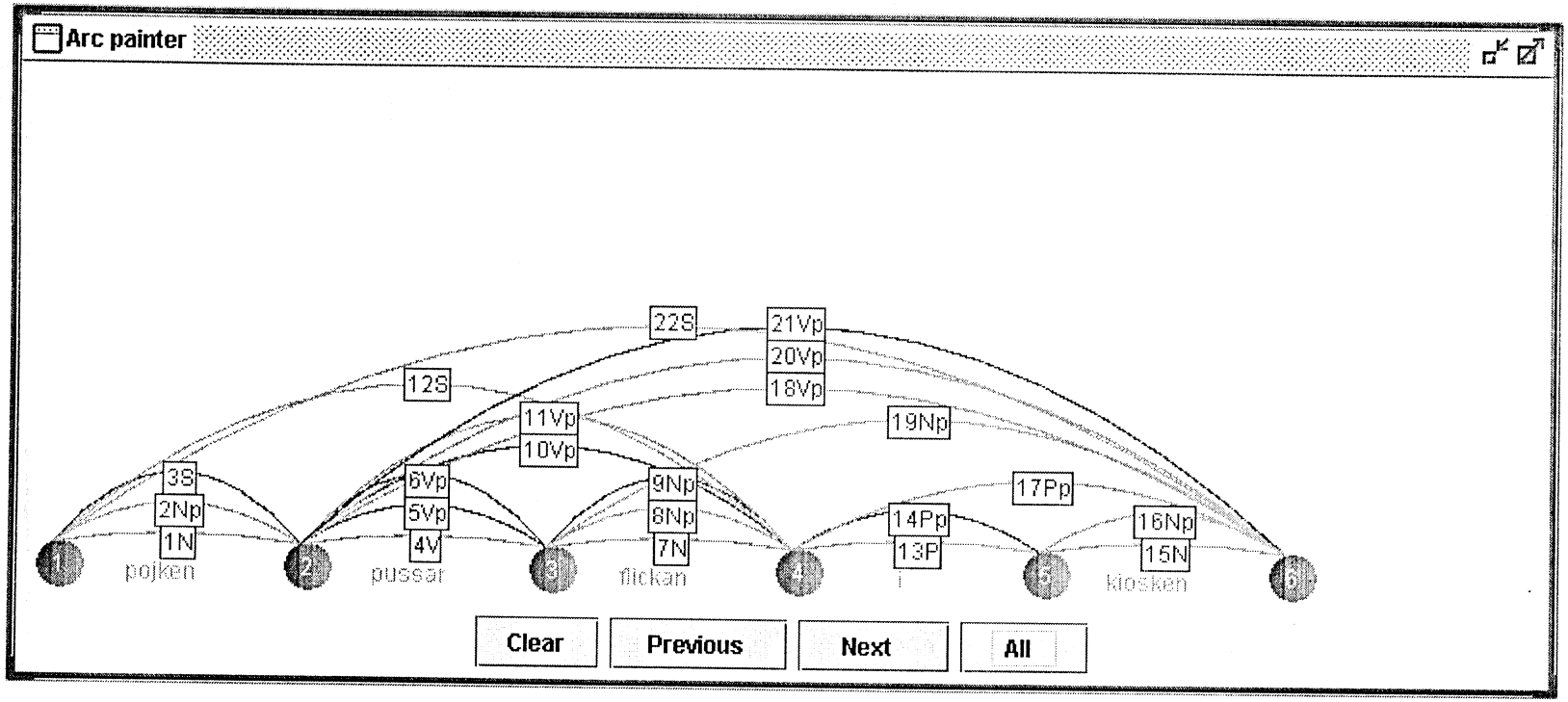
Bottomup á la Kilbury med bottomupfilter djupetförst



1.N->pojken.
2.Np->N.
3.S->Np.Vp
4.V->pussar.
5.Vp->V.Np Pp
6.Vp->V.Np
7.N->flickan.
8.Np->N.
9.Np->N.Pp
10.Vp->V Np.Pp
11.Vp->V Np.
12.S->Np Vp.
13.P->i.
14.Pp->P.Np
15.N->kiosken.
16.Np->N
17.Pp->P Np.
18.Vp->V Np Pp.
19.Np->N Pp.
20.S->Np Vp.
21.Vp->V Np.Pp
22.Vp->V Np.

Bilaga 2.6b

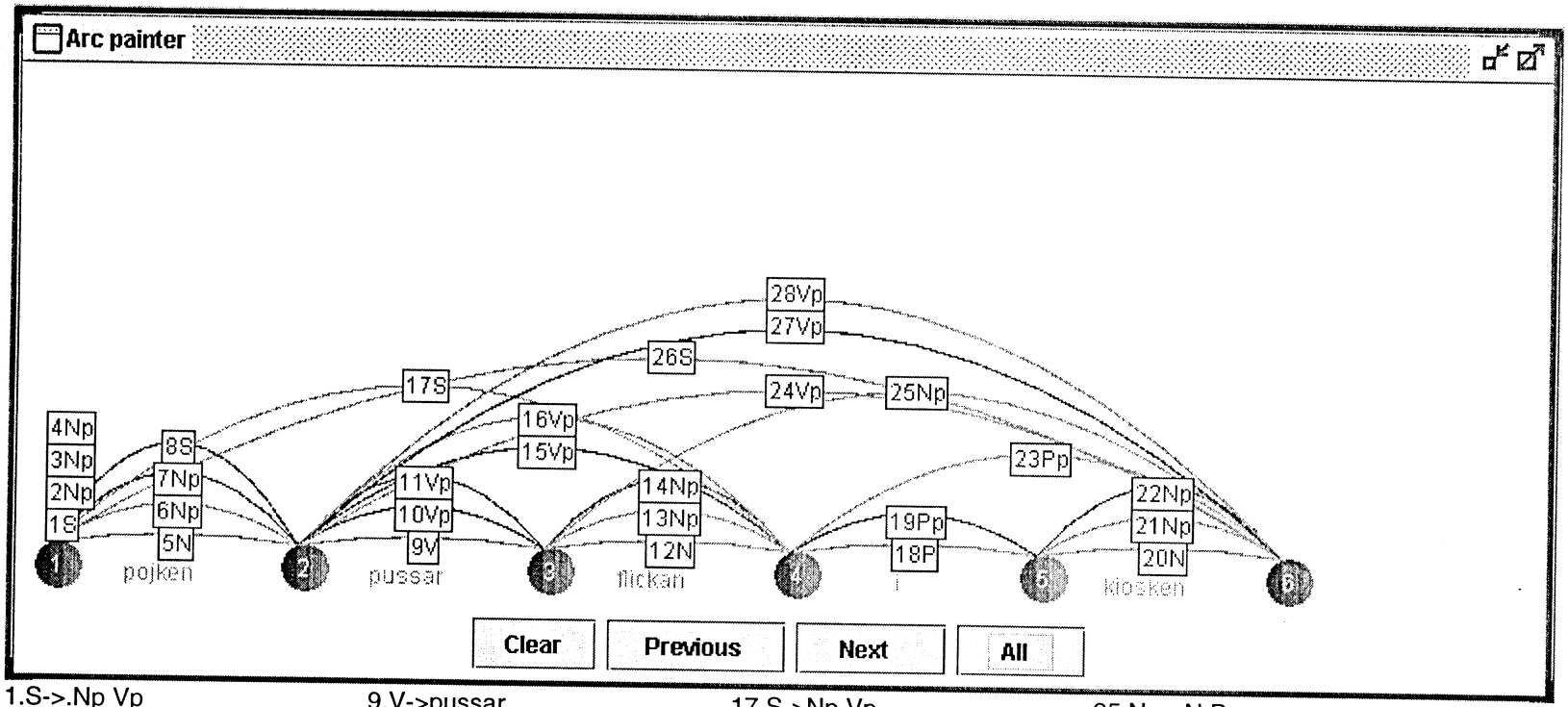
Bottomup á la Kilbury med bottomupfilter breddenförst



1.N->pojken.
2.Np->N.
3.S->Np.Vp
4.V->pussar.
5.Vp->V.Np Pp
6.Vp->V.Np
7.N->flickan.
8.Np->N.
9.Np->N.Pp
10.Vp->V Np.Pp
11.Vp->V Np.
12.S->Np Vp.
13.P->i.
14.Pp->P.Np
15.N->kiosken.
16.Np->N.
17.Pp->P Np.
18.Vp->V Np Pp.
19.Np->N Pp.
20.Vp->V Np.
21.Vp->V Np.Pp
22.S->Np Vp.

Bilaga 2.7a

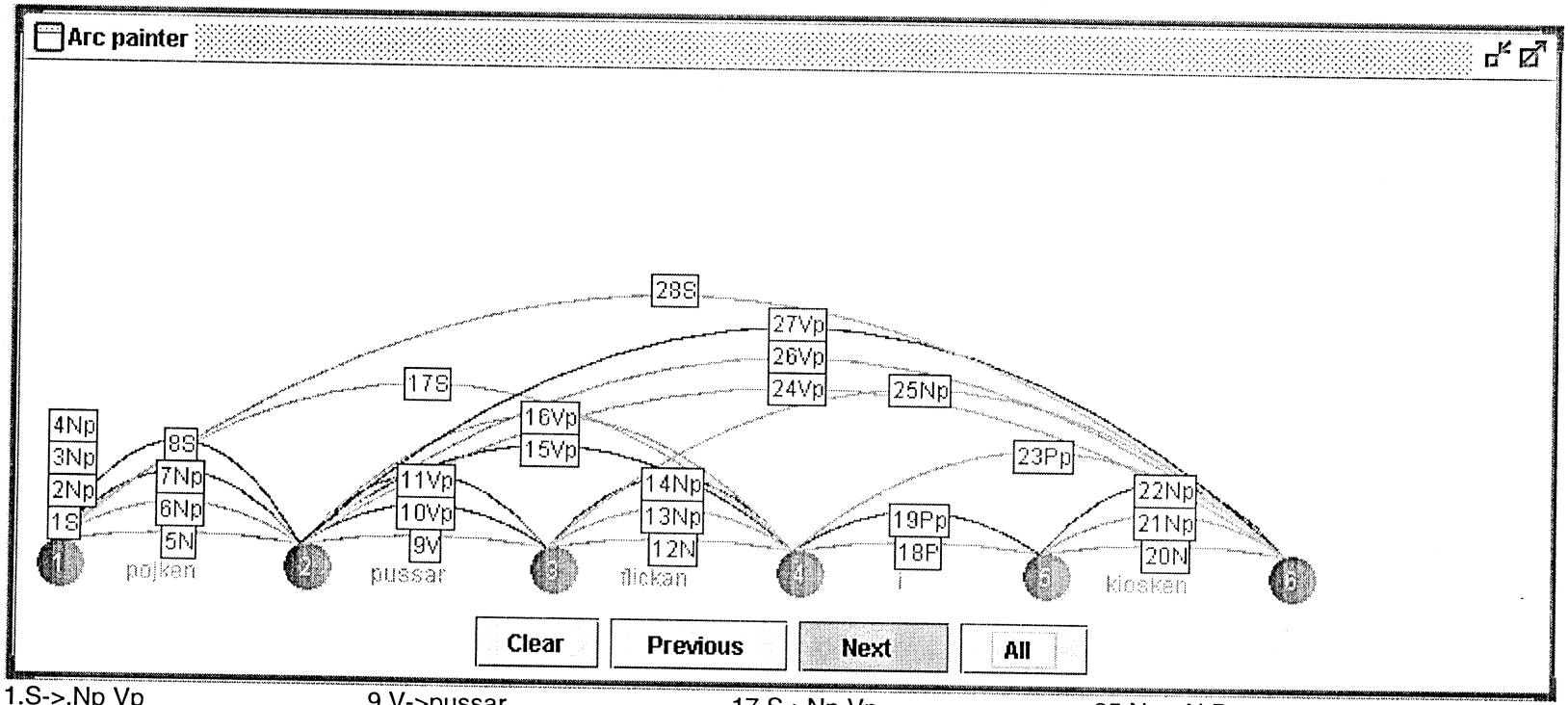
Bottomup á la Kilbury med topdownfilter djupetförst



1.S->.Np Vp
2.Np->.N
3.Np->.N Pp
4.Np->.Det A Np
5.N->pojken.
6.Np->N.
7.Np->N.Pp
8.S->Np.Vp
9.V->pussar.
10.Vp->V.Np Pp
11.Vp->V.Np
12.N->flickan.
13.Np->N.
14.Np->N.Pp
15.Vp->V Np.Pp
16.Vp->V Np.
17.S->Np Vp.
18.P->i.
19.Pp->P.Np
20.N->kiosken.
21.Np->N.
22.Np->N.Pp
23.Pp->P Np.
24.Vp->V Np Pp.
25.Np->N Pp.
26.S->Np Vp.
27.Vp->V Np.Pp
28.Vp->V Np.

Bilaga 2.7b

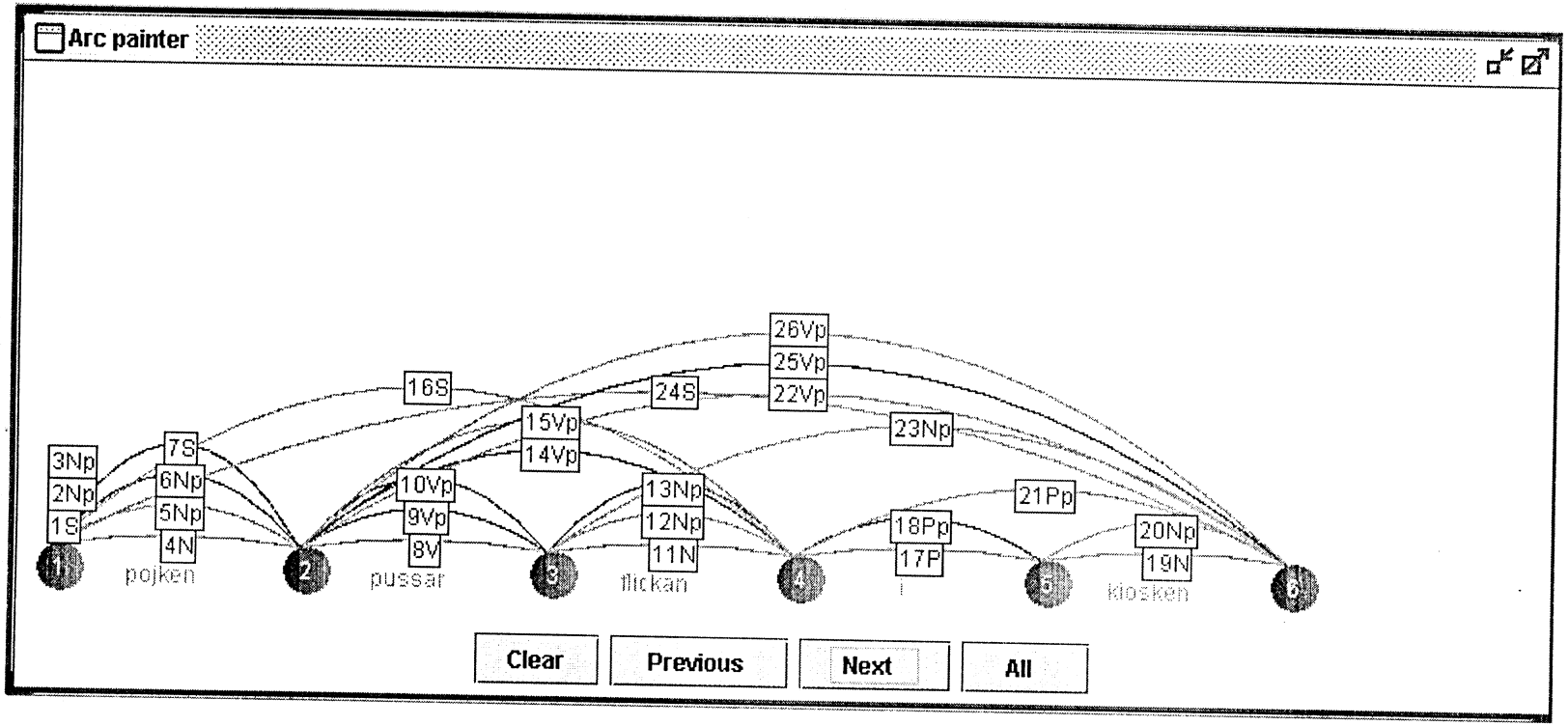
Bottomup á la Kilbury med topdownfilter breddenförst



1.S->.Np Vp
2.Np->.N
3.Np->.N Pp
4.Np->.Det A Np
5.N->pojken.
6.Np->N.
7.Np->N.Pp
8.S->Np.Vp
9.V->pussar.
10.Vp->V.Np Pp
11.Vp->V.Np
12.N->flickan.
13.Np->N.
14.Np->N.Pp
15.Vp->V Np.Pp
16.Vp->V Np.
17.S->Np Vp.
18.P->i.
19.Pp->P.Np
20.N->kiosken.
21.Np->N.
22.Np->N.Pp
23.Pp->P Np.
24.Vp->V Np Pp.
25.Np->N Pp.
26.Vp->V Np.
27.Vp->V Np.Pp
28.S->Np Vp.

Bilaga 2.8a

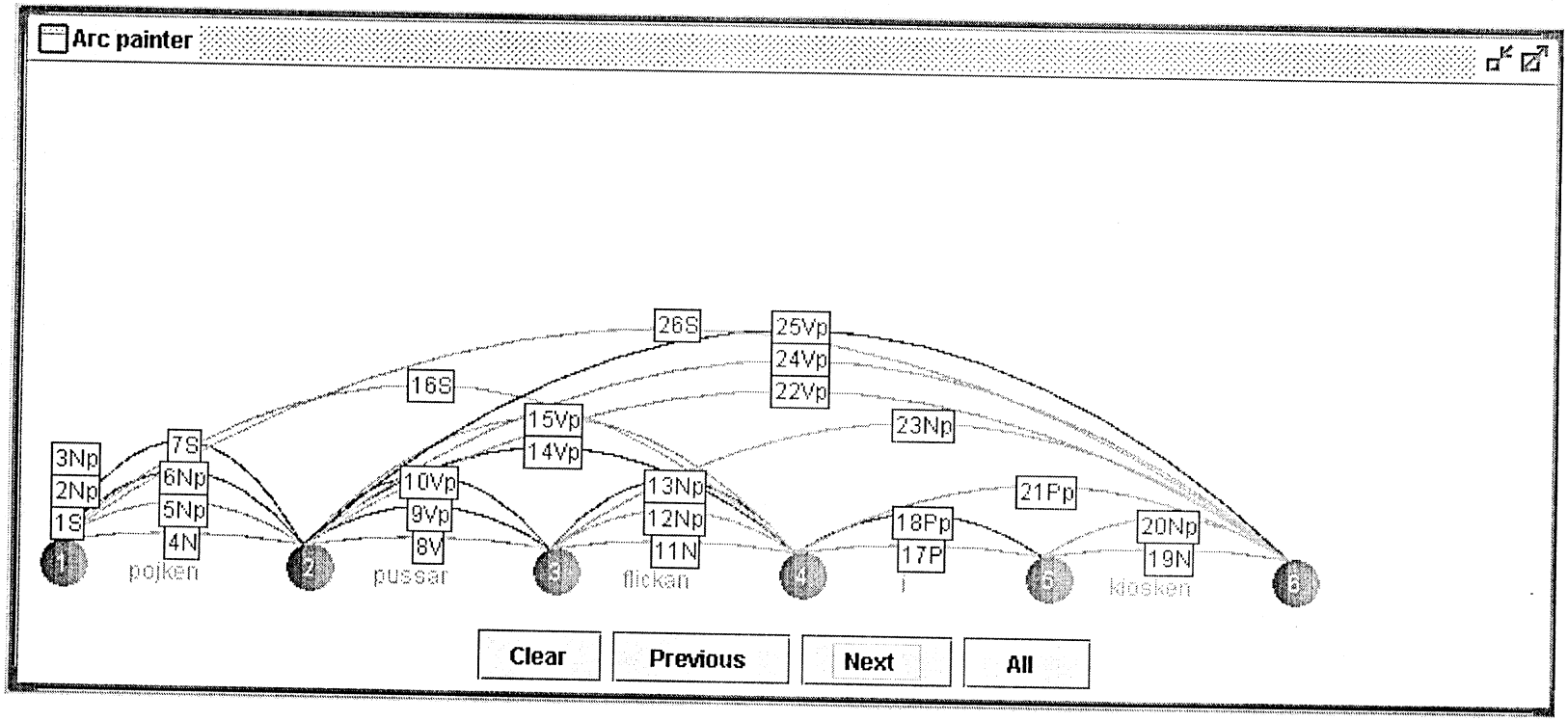
Bottomup á la Kilbury med topdown & bottomupfilter djupetförst



1.S->.Np Vp
2.Np->.N
3.Np->.N Pp
4.N->pojken.
5.Np->N.
6.Np->N.Pp
7.S->Np.Vp
8.V->pussar.
9.Vp->V.Np Pp
10.Vp->V.Np
11.N->flickan.
12.Np->N.
13.Np->N.Pp
14.Vp->V Np.Pp
15.Vp->V Np.
16.S->Np Vp.
17.P->i.
18.Pp->P.Np
19.N->kiosken.
20.Np->N.
21.Pp->P Np.
22.Vp->V Np Pp.
23.Np->N Pp.
24.S->Np Vp.
25.Vp->V Np.Pp
26.Vp->V Np.

Bilaga 2.8b

Bottomup á la Kilbury med topdown & bottomupfilter breddenförst



1.S->.Np Vp
2.Np->.N
3.Np->.N Pp
4.N->pojken.
5.Np->N.
6.Np->N.Pp
7.S->Np.Vp
8.V->pussar.
9.Vp->V.Np Pp
10.Vp->V.Np
11.N->flickan.
12.Np->N.
13.Np->N.Pp
14.Vp->V Np.Pp
15.Vp->V Np.
16.S->Np Vp.
17.P->i.
18.Pp->P.Np
19.N->kiosken.
20.Np->N.
21.Pp->P Np.
22.Vp->V Np Pp.
23.Np->N Pp.
24.Vp->V Np.
25.Vp->V Np.Pp
26.S->Np Vp.