3. Granskas regelspråk

Ett viktigt övergripande mål inom Granska-projektet var att skapa ett system där vi hade full kontroll över samtliga moduler. Vi önskade oss full kontroll för experiment och utveckling av idéer från tokenisering, taggning, regelspråk, regelmatchning till olika typer av användargränssnitt. Det var därför naturligt att utveckla ett eget regelspråk för våra syften. Detta kapitel kommer att behandla detta regelspråk som är en central del i Granska.

Den följande beskrivningen riktar sig främst till den som avser att skriva regler eller är intresserad av hur regelspråket fungerar vilket är viktigt för den fortsatta läsningen av avhandlingen. För en kort beskrivning av regelspråkets implementation se Carlberger, Domeij, Kann & Knutsson (2000). I Appendix B presenteras en BNF-definition av regelspråket, skriven av Viggo Kann, för att definitivt undanröja alla eventuella oklarheter i detta kapitel. Utvecklingen av regelspråket har utförts i nära samarbete med dess implementatörer Viggo Kann och Johan Carlberger. Den flexibla och robusta implementationen av regelspråket har möjliggjort en ständig och friktionsfri utveckling, där nya idéer snabbt, enkelt och konsekvent, utan återvändsgränder, har införts i regelspråket och regelmatcharen.

3.1 Ett generellt regelspråk

Ett av huvudsyftena med det nya regelspråket är att det kan användas till olika lingvistiska operationer:

- Grammatikkontroll. Det finns tydliga och generella sätt att ange vad som skall detekteras och markeras och hur det skall korrigeras.
- Identifiering av fraser och satsgränser. Det är möjligt att detektera språkliga fraser, t.ex. nominalfraser (NP). Det går också att söka efter dessa för att förbättra detektionen av andra språkliga konstruktioner. En regel som detekterar ett språkligt mönster kan återanvändas av andra regler.
- Lingvistisk sökning och redigering. Det går att utföra sökningar efter lingvistiska enheter, samt utföra redigering av dessa t.ex. förändra tempus hos verb eller att flytta konstituenter inom en mening.
- Förbättrad taggning. Granskas taggare väljer i vissa fall ”fel” tolkning av ett ord. Dessa fel kan identifieras med regler som kan ange en ny taggning av det detekterade området, samt be om en ny taggning av hela meningen som tar hänsyn till den angivna taggen.

3.1.1 Förbättrad och utökad funktionalitet

I texten nedan följer en genomgång av målsättningarna med regelspråket, längre ned i texten förklaras hur de skall förverkligas. Ett viktigt delmål är att reglerna skall få en tydlig notation och ökad uttryckskraft. Inspiration till reglernas notation har hämtats från objektorienterade programmeringsspråk som C++ och Java.

Den lingvistiska beskrivningens utformning har främst påverkats av Constraint Grammar (Karlsson, 1995b) och olika typer av Finite State Parsing (se t.ex. Karttunen et al, 1996), se avsnitt 3.5.5 och 3.5.7 för jämförelser av regelsyntax. Även den möjlighet till lingvistisk abstraktion som finns inom frasstrukturgrammatik och särdragsbaserad grammatik har bidragit till utvecklingen av regelspråket, se framförallt avsnittet om *Hjälpregler*.

Granskas regelspråk utvecklades med följande målsättningar:

- Det ska vara möjligt att konstruera granskningsregler som med god täckning och hög precision hittar de feltyper som eftersträvas.
- Regelkonstruktören styr den information som skall presenteras för användaren, till exempel markering av felen i texten, ersättningsförslag och kommentarer och förklaringar till det signalerade felet.
- Reglerna skrivs i en objektorienterad notation med attribut och värden för att förenkla generaliseringar och öka läsbarheten.
- En regel kan utnyttja språklig information från tecken- och tokennivå till enklare frasstruktur.
- En god beskrivning av en konstruktion skall återvändas i andra regler.
- Regelspråket skall enkelt kunna utvidgas till andra tillämpningar än redan nämnda.
- Regelkonstruktören skall inte belastas med krav på effektiva regler, utan detta skall lösas av det program som tolkar reglerna.

Bra detektion och diagnos uppnås genom:

- statistisk kvantifiering (se 3.2.2).
- jämförelser av särdragens värden (se 3.3.5)
- sekvensvariabler (se 3.3.10)
- villkorliga hopp, för att åstadkomma regelprioritet (se 3.4.10), och accepterande regler (se 3.5.4).
- detektion av lingvistiska enheter med hjälpregler, t.ex. nominalfraser (se 3.5.1).

- detektion och korrektion av felaktig taggning (se 3.5.3)

Bra korrektion skall uppnås genom att:

- det som är fel skall markeras med teckenprecision i texten (se 3.3.1 och 3.4.8)
- genererade ersättningsförslag skall sättas in i texten (se 3.4.9)

3.1.2 Definitioner och förklaringar

Token	En enhet som har ett språkligt innehåll, vanligtvis är ett token ett ord, t.ex. *bilen*, men även ett datum som *den 26 juli 2000*. Vad som är ett token bestäms av Granskas tokeniserare.
Element	Ett samlingsnamn för enskilda tecken, teckensekvenser och token, det vill säga något som går att matcha i indata.
Regelmatcharen	Regelmatcharen är det program som tolkar regelspråket och ser till att reglerna appliceras och tillämpas.

3.1.3 En regel består av ett vänsterled och ett högerled

En regel består av ett vänster- och högerled. I vänsterledet skall det anges vad som skall matchas i indata. I högerledet anges vad som skall göras med det matchade i vänsterledet. För att det skall fungera väl är det viktigt att relationen mellan vänsterled och högerled är väl utarbetad.

I beskrivningen av en regels uppbyggnad kommer vänsterledets beståndsdelar och funktionalitet att beskrivas i avsnitt 3.3. När detta är gjort kommer högerledet att beskrivas i avsnitt 3.4. I detta kapitel sätts också vänster- och högerled ihop för att bilda regler. I avsnitt 3.5 beskrivs andra typer av regler, framförallt hjälpregler som påminner om regler i en frasstrukturgrammatik. Först av allt måste dock de operatorer, kvantifikatorer och konstanter som finns i regelspråket beskrivas; detta sker i nästa avsnitt.

3.2 Operatorer, kvantifikatorer och konstanter i regelspråket

3.2.1 Operatorer

&	logiskt *och* mellan villkor
|	logiskt *eller* mellan villkor
!	logisk negation av villkor
=	lika med
!=	inte lika med
<	mindre än

>	större än
<=	mindre än eller lika med
>=	större än eller lika med
+	binärt plus
-	binärt och unärt minus. Används också i intervall mellan matchningsvariabler i högerledet.
:=	tilldelningsoperator
,	kommatecken används för att avskilja matchningsvariabler
;	semikolon innebär ett logiskt *eller* mellan regler
~	tilde innebär subtraktion mellan två regler enligt mängdteorin, tilde används också vid matchning av reguljära uttryck på ordnivå
-->	en konsekvenspil som skiljer vänsterled från högerled, vilket innebär att högerledet är en konsekvens av vänsterledets premisser
{	regelbörjan, en regel inleds alltid med {
}	regelslut, en regel avslutas alltid med }
{{	inleder en regel med subtraktion
}}	avslutar en regel med subtraktion

3.2.1.1 Operatorer i fallande ordning

Operatorer binder olika hårt och nedan följer en uppräkning av hur hårt de binder i fallande ordning.

```
+-
~
=, !=, <, >, >=, <=
!
&
|
:=
```

3.2.2 Kvantifikatorer

För att kunna hantera olika tolkningar av orden använder vi en statistisk kvantifierare som anger ett tröskelvärde för en viss given tolkning.

`P(Sannolikhet, Taggkombination)`	Statistisk kvantifierare som medför att en regel tillämpas endast om sannolikheten för att tolkningen Taggkombination, är lika med eller större än värdet för Sannolikhet. Sannolikheten anges som ett reellt tal `X` och det gäller att `0 <= X <= 1`.
`A(Taggkombination)`	Allkvantifierare, ett förkortat skrivsätt för `P(1, Taggkombination)`.

`E(Taggkombination)`	Existenskvantifierare, ett förkortat skrivsätt för `(0.0001,Taggkombination)`.

3.2.3 Konstanter

Det behövs två booleska konstanter i regelspråket: `true`, `false`. Det behövs också numeriska konstanter, det vill säga alla heltal och reella tal. De reella talen skrivs med decimalpunkt, till exempel `0.34`.

3.3 Notationen i vänsterledet

En matchningsvariabel, t.ex. `X`, som anges i vänsterledet, beskriver det område som skall matchas i indata när matchningen sker mot ett token; matchning på teckennivå särbehandlas, se nedan.

3.3.1 Matchning av reguljära uttryck på teckennivå

Regler där vänsterledet består av reguljära uttryck särbehandlas i regelspråket eftersom dessa matchar över token- och meningsnivå. I vänsterledet anges endast det reguljära uttryck som skall matcha indata. Högerledet kommer dock att se likadant ut för regler som endast innehåller reguljära uttryck på teckennivå. I vänsterledet skrivs det reguljära uttrycket inom "...":

```
/"regexp"/
```

där regexp utgörs av ett reguljärt uttryck på teckennivå med samma notation som i programmet *grep* under Unix.

3.3.1.1 Markering och korrigering i det reguljära uttrycket

För att markeringar och korrigeringar ska kunna göras i regler med reguljära uttryck måste det finnas sätt att uttrycka vad som skall markeras och vad som skall vara med i korrigeringen. Ett förslag är att använda den notation som finns i editeringsprogrammet *sed*. I *sed* finns det metatecken som anger vad som skall sparas i det reguljära uttrycket. Dessa är \(och \). För att ange vad som skall finnas med i den korrigerade strängen används referenserna \1 till \n. \1 refererar till den första parentesen som gjorts med \(och \) och så vidare.

Exempel:
I *sed* blir regeln för att ta bort dubbla mellanslag i filen dubbla.txt enligt följande:

Indata: `jag har  en boll som är grön.`

Utdata: `jag har en boll som är grön.`

```
sed 's/\([a-zåääö][a-zåääö]*\)[ ][ ]\([a-zåääö][a-zåääö]*\)/\1 \2/'
dubbla.txt
```

Eftersom reglerna i det nya regelspråket består av ett vänster- och högerled är det naturligt att metatecknen `\(` och `\)` anges i vänsterledet som matchar indata och `\1` till `\n` anges i högerledet i fälten för att markera och korrigera, se avsnitten **mark** och **corr**. `\1` till `\n` anges inom citattecken i mark- och corrfältet.

I ett senare avsnitt visas också hur sed-uttrycket ovan kommer att formuleras i Granskas regelnotation. Där visas även hur det matchade området skall markeras i texten. Regler och regelmatchning med reguljära uttryck på teckennivå är under implementation (Baltatzis, forthcoming).

3.3.2 Matchning på tokennivå

Varje token som skall matchas beskrivs med en matchningsvariabel som innehåller olika attribut för att ange vad som skall matchas. Det matchade ses som ett objekt med attribut och metoder för manipulation. Attributen kombineras med logiska operatorer. Attributen består av två grupper – textattribut och särdragsattribut; dessa presenteras nedan.

Objektens textattribut:

`real_text` Detta attribut innehåller objektets textsträng. För att matcha ett objekts textsträng används dels textsträngar och dels reguljära uttryck på teckennivå. Textsträngar och reguljära uttryck anges inom "...". För att matcha ord används `real_text="ord"` och för att matcha med reguljära uttryck används `real_text ~ "regexp"`.

`text` Detta attribut innehåller objektets lexikala textsträng, som skiljer sig mot real_text som innehåller den textsträng som finns i själva texten. Om textsträngen i texten är *öl- och vinrättighter* så är `real_text="öl och vinrättigheter"` medan `text="vinrättigheter"`. Attributet `text` används för att avgöra ordets morfosyntaktiska tolkning.

`lemma` Detta attribut innehåller objektets lemma, det vill säga ordets grundform. För matchning av detta attribut anges en textsträng. I vänsterledet skriver man `lemma="lemma"` för att matcha ordens lemma.

`token` Granskas tokeniserare klassificerar token med hjälp av reguljära uttryck. Denna klassificering är åtkomlig från reglerna med attributet `token`. Några exempel på tokenklassificeringar är t.ex. webbadresser som får värdet `TOKEN_URL`, enkla ord får värdet `TOKEN_SIMPLE_WORD`. Konstruktioner som *öl- och vinrättigheter* får värdet `TOKEN_SPLIT_WORD`.

Objektens särdragsattribut:

Följande särdragsattribut används i denna text (komplett lista över samtliga särdragsklasser och särdragsvärden finns i Appendix A). I regelspråket skiljer vi

mellan särdragsklasser som utgör objektens attribut och särdragsvärden som utgör de värden som objektens attribut är satta till. Särdragsklasserna och särdragsvärdena som man vill använda i regelspråket definieras i en separat fil, för det är information som behövs redan vid taggningen.

Särdragsklasser som används i detta kapitel:

`wordcl`	ordklass
`gender`	genus
`num`	numerus
`spec`	species
`vbf`	verbform
`case`	kasus
`style`	stil
`sed`	meningsavgränsare

Särdragsvärden som används i detta kapitel:

`dt`	determinerare (artikel)
`jj`	adjektiv
`nn`	nomen (substantiv)
`def`	definit (bestämd)
`utr`	utrum
`vb`	verb
`pn`	pronomen
`vard`	vardagligt ord
`pc`	particip
`foal`	formella och ålderdomliga ord
`datm`	dataterm
`ie`	infinitivmärke

Alla matchningsvariabler består av två fält oavsett om det finns några matchningsvillkor:

```
variabelnamn(matchningsvillkor)
```

En matchningsvariabel kan se ut enligt följande:

```
X(attribut1=värde1 & attribut2=värde2 & ... & attribut_n=värde_n)
```

Exempel 1:

En matchningsvariabel som matchar ett godtyckligt token i texten skrivs enligt:

```
X()
```

Exempel 2:

En matchningsvariabel i vänsterledet som matchar ordet *bilen* kommer att se ut som nedan (med en viss redundans):

```
X(text="bilen" & lemma="bil" & wordcl=nn & gender=utr & num=sin &
spec=def & case=nom)
```

3.3.3 Operatorer mellan matchningsvariabler och regler

Matchningsvariabler kan kombineras för att bilda fraser och andra konstruktioner. Det skall också vara möjligt att kombinera regler för att konstruera mer generella regler. Det finns tre operatorer för detta ändamål. Dessa operatorer kan kombineras och ganska komplexa uttryck kan konstrueras.

`A , B` Operatorn **,** skiljer matchningsvariabel `A` från matchningsvariabel `B` vilket innebär att `A` skall följas av `B`. `B` beskriver ett nytt ord som följer direkt efter `A` i indata. `A` och `B` måste gälla för att en regel skall tillämpas.

`A ; B` Operatorn **;** innebär ett logiskt *eller* mellan regler vilket innebär att regel `A` eller regel `B` måste gälla för att den sammansatta regeln skall tillämpas. Logiskt *eller* kan uttrycka mängdbegreppet `A` union `B`.

`A ~ B` Operatorn `~` innebär subtraktion mellan två regler, vilket innebär att villkoren i regel `A` skall gälla medan villkoren i `B` inte skall gälla för att regeln skall matcha. Denna operator får endast användas i en speciell syntax, se avsnittet *Union och subtraktion mellan regler* (3.5.5).

En sekvens av matchningsvariabler anges med avskiljaren ”,” mellan matchningsvariablerna enligt nedan:

```
X1(text="ordform" operator lem="lemma" operator matchningskrav),
X2(text="ordform" operator lem="lemma" operator matchningskrav),
...,
Xn(text="ordform" operator lem="lemma" operator matchningskrav)
```

Exempel på vad man kan uttrycka med dessa operatorer i samverkan finns i avsnittet *Union och subtraktion mellan regler* (3.5.5).

3.3.4 Punktnotation

Om varje matchningsvariabel ses som ett objekt med attribut och metoder skapas en regelnotation som är mycket tydlig och kraftfull. Varje objekt kommer att motsvara ett element i texten, det vanligaste fallet blir att ett objekt motsvarar ett ord i texten. Från objektorienterade programmeringsspråk hämtas punktnotationen för att komma åt objektets attribut och metoder. En viktig sak är att man kan få fram objektens värden var som helst inom regeln och inte bara i den lokala matchningsvariabeln utan i andra matchningsvariabler samt i högerledet.

Ett sätt att undvika att man gör punktnotation på objekt som saknar ett attribut vore att man alltid måste matcha ordklassen först, men då förlorar man möjligheten att kunna söka direkt på morfosyntaktiska särdrag. Det enklaste är om man endast behöver ange det enskilda särdrag som man är ute efter. I regelspråket löser vi detta genom att låta objektet returnera värdet `undef` när ett attribut saknas. Det är enkelt att uttrycka att det måste finnas ett språkligt värde på attributet genom att skriva att särdraget skall vara skilt från `undef`:

```
X.särdragsattribut!=undef
```

Detta är praktiskt när man söker efter något som inte skall ha ett visst värde men som skall ha ett värde på det angivna attributet:

```
X.särdragsattribut!=undef & X.särdragsattribut!=värde
```

Exempel:

Matcha alla ord som inte är i utrum (`utr`) men som har ett nominalt innehåll.

```
X(gender!=undef & gender!=utr)
```

matchar:	**matchar inte:**
huset, litet, det, ett	till, bilen, går, spelar

3.3.5 Att jämföra värden på särdrag

För att man skall kunna jämföra värden på särdrag krävs det att den lexikala informationen ordnas upp och värden sätts till attribut. Det måste finnas en lexikonpost för varje token. Underspecificerade taggar måste kunna matcha specificerade när det är möjligt, t.ex. utr/neu skall matcha både utr och neu.

För att kunna jämföra två eller flera matchningsvariabler behöver vi olika särdragsattribut f1, f2, ..., fn som sätts till värden F1, F2, ..., Fn.

Uppordning av den lexikala informationen sker enligt följande:

```
text=Ord & lemma=Lemma & f1=F1 & f2=F2 & ...& fn=Fn
```

Exempel:

Ordet *bilen* får följande utseende:

```
(text="bilen" & lemma="bil" & wordcl=nn & gender=utr & num=sin &
spec=def & case=nom)
```

Med punktnotation på de olika objekten jämförs värdena. Det är inte nödvändigt att göra punktnotation på det objekt man befinner sig i, det vill säga om vi gör en jämförelse i objektet X så gäller att `särdrag=X.särdrag & särdrag=särdrag`

Vid jämförelser får endast kända variablers särdragsattribut jämföras. Det är inte tillåtet att göra punktnotation på variabler som inte är introducerade, det vill säga

endast bakåtreferens är tillåten. Jämförelse av särdragsattribut i olika matchningsvariabler måste ske enligt:

```
X1(särdragsattribut_1=värde),
X2(särdragsattribut_1=X1.särdragsattribut_1),
...,
Xn-1(särdagsattribut_1),
Xn(särdragsattribut_1=Xn-1.särdragsattribut_1)
```

3.3.6 Namngivning och kategorisering av regler

En regel måste kunna referera till andra regler; därför behöver vi kunna namnge regler. Regelmatcharen måste också kunna hålla reda på vilken regelkategori som en regel tillhör så att användaren kan plocka bort och lägga till regelkategorier efter sina önskemål. Vi anger regelnamn och regelkategori enligt följande notation. Man inleder deklarationen med det reserverade namnet `category` följt av den regelkategori man vill skapa. Fälten `info` och `link` måste också anges, `info` innehåller en kommentar som kan vara generell för en hel regelkategori, `link` innehåller en hyperlänk till en text med utförligare information. Fälten `info` och `link` beskrivs utförligare i 3.4 (*Notationen i högerledet*).

```
category regelkategori {
info(”kategorikontroll”)
link(”hyperlänk” ”länktext”)
}
```

En regel namnges med ett regelnamn följt av `@` följt av regelkategorin, enligt:

```
regelnamn@regelkategori {
X()
```

3.3.7 Vänsterledets syntax

Vänsterledet måste bestå av följande element för att tillhöra regelspråket.

`regelnamn@regelkategori`	`regelkategori krävs utom i s.k. hjälpregler`
`{`	`regelbörjan`
`X_0(),`	`matchningsvariabeln X_0. Om det kommer fler matchningsvariabler efter X_0 följs X_0 av avskiljaren ,`
`…,`	
`X_n()`	`ev. ytterligare matchningsvariabel.`
`-->`	`pilen avskiljer vänsterledet från högerledet`

Exempel 1:

Exemplet visar hur detektion av kongruenta nominalfraser går till. De tre matchningsvariablerna `X1`, `X2` och `X3` måste ha samma värde i respektive särdrag för att matchning skall ske. I det här fallet måste de olika särdragsattributen ha följande värden: `gender=utr`, `num=sin` och `spec=ind`. Regeln kommer dessutom att matcha fraser som *det stora huset* eller *de snälla pojkarna*, det vill säga alla korrekta nominalfraser som består av en artikel följt av ett adjektiv följt av ett substantiv. Felaktiga fraser som *en litet bil*, *det stort huset* och *den snälla pojkarna* kommer däremot inte att matchas.

```
ex1@kong    {
  X1(wordcl=dt),
  X2(wordcl=jj & gender=X1.gender & num=X1.num & spec=X1.spec),
  X3(wordcl=nn & gender=X2.gender & num=X2.num & spec=X2.spec)
-->
```

Exempel 2:

Exemplet visar ett vänsterled för detektion av den inkongruenta nominalfrasen *en litet bil*. Notera att vänsterledet dessutom kommer att detektera nominalfraser som *ett liten hus*.

```
ex2@kong    {
  X1(wordcl=dt),
  X2(wordcl=jj & num=X1.num & spec=X1.spec ),
  X3(wordcl=nn & gender!=X2.gender & gender=X1.gender & num=X2.num &
      spec=X2.spec)
-->
```

3.3.8 Andra tolkningar än den taggaren har gett

Det kan vara intressant att veta om ett token har ytterligare tolkningar än den som taggaren gett, t.ex. vid omtaggning av en mening. Informationen om ett tokens olika tolkningar finns i objektets lexikala information, som nås genom attributet `lex`. För att undersöka vilka tolkningar som existerar använder man tre kvantifikatorer med följande notation:

Existenskvantifikatorn	`E(lex.särdrag=värde)`
Allkvantifikatorn	`A(lex.särdrag=värde)`
Statistisk kvantifierare	`P(sannolikhet, lex.särdrag=värde)`

`E`, `A` och `P` är reserverade ord i regelspråket och får inte användas som namn på matchningsvariabler.

Exempel 1:

Vi vill ange som krav att det måste finnas en adjektivtolkning förutom den substantivtolkning som taggaren gett för att en matchning skall ske. Regeln matchar *svenska* men inte *bilen*

```
ex1@kvant   {
X1(wordcl=nn & E(lex.wordcl=jj))
-->
```

Exempel 2:

Vi vill ange som krav att det endast får finnas substantivtolkningar förutom den substantivtolkning som taggaren gett för att en matchning skall ske. Regeln matchar: *bilen* men inte *svenska*.

```
ex2@kvant   {
X1(wordcl=nn & A(lex.wordcl=nn))
-->
```

Exempel 3:

Vi vill ange som krav att det måste finnas en tolkning som determinerare med lexikal sannolikhet som är lika med 0.6 förutom den pronomentolkning som taggaren gett för att en matchning skall ske. Regeln matchar *den* men inte *det*.

```
{
X1(wordcl=pn & P(0.6, lex.wordcl=dt))
-->
```

3.3.8.1 Stiltaggar

En del ord i lexikon är uppmärkta med stiltaggar för att olika regler skall kunna detektera dessa som stilistiska avviklelser. Stiltaggarna hos orden nås också genom attributet `lex`.

Exempel:
Indata: *Jag såg dej på stan igår.*
Ordet *dej* matchas.

```
ex1@stil    {
X(lex.style=vard) -->
```

3.3.9 Definition av konstanter

Det är möjligt att definiera konstanter i regelspråket; detta måste göras överst i regelsamlingen. Vi använder tilldelningsoperatorn för att tilldela dem värden. Man inleder deklarationen med det reserverade ordet `const` och avslutar deklarationen med `;`

```
const konstantnamn:=värde;
```

Exempel 1:
Vi vill ha en konstant för kommentaren för dubbelt supinum.

```
const supkom:="Dubbelt supinum. Undvik s.k. dubbelt supinum, t.ex.
>>kunnat gjort>>. Skriv istället >>kunnat göra>> som är grammatiskt
korrekt. Myndigheternas skrivregler 4.2.8";
```

3.3.10 Sekvensvariabler

En sekvensvariabel innebär att samma krav för matchning gäller över flera token. Genom att sätta en räknare efter matchningsvariabeln så anger man att denna variabel är en sekvensvariabel som sträcker sig över noll eller flera token. Kraven på denna variabel anges på samma sätt som för en vanlig matchningsvariabel. Det går endast att använda sekvensvariabler för matchning mot token. Samtliga möjliga matchningar utförs av regelmatcharen. Flera användare[11] av regelspråket efterlyser en operator för en styrning av denna matchning, till exempel till längsta möjliga matchning eller kortast möjliga matchning. Detta skulle öka kontrollen vid användningen av sekvensvariabler och ge mer överblickbara regler. Regelspråket bör i framtiden innehålla kvantifierare för att styra hur lång matchning som skall utföras.

Operator (räknare)	Beskrivning av antal token som matchas	Operator med matchningsvariabel
`*`	`noll eller flera`	`X(matchningskrav)*`
`?`	`noll eller ett`	`X(matchningskrav)?`
`+`	`ett eller flera`	`X(matchningskrav)+`
`n`	`noll eller maximalt n (heltal)`	`X(matchningskrav)n`

Tabell 3.1. Fyra olika räknare för sekvensvariabler.

Exempel 1:
Detta vänsterled visar hur en regel matchar olika varianter av en enkel nominalfras med optionellt adjektiv. Några fraser som regeln täcker in är *en bil, en liten bil, en stor röd bil.*

```
ex1@kong    {
X1(wordcl=dt),          % determinerare
X2(wordcl=jj)*,         % noll eller flera adjektiv
X3(wordcl=nn)           % substantiv
-->
```

[11] Jag är tacksam för förslaget från Martin Hassel, doktorand på Nada, KTH; om att föra in kvantifierare för längsta möjliga matchning och kortast möjliga matchning.

3.3.10.1 Att komma åt ett speciellt token i en sekvensvariabel

För att komma åt ett speciellt token i en sekvensvariabel eller hjälpregel används en vektornotation `X[nth token]`. Vilket token i ordningen man är ute efter anges med en variabel `i`, det blir: `X[i]`. Det första elementet i vektorn är `X[0]`. Exempel på hur denna notation används, presenteras när högerledet är specificerat. Samma notation gäller för att komma åt ett speciellt token i en hjälpregel (se avsnittet *Hjälpregler*).

3.3.11 Att kunna ange meningsbörjan och meningsslut

Vi vill kunna skriva regler som tar hänsyn till meningsbörjan och meningsslut. Det blir enklast om vi har en tagg för meningsbörjan/meningsslut, `sen`, som kan matchas av reglerna.

Exempel:

Med information om meningsbörjan och meningsslut kan vi skriva regler som matchar en mening som precis består av en sats, varken mer eller mindre.

Indata: Jag såg den lilla hunden.

```
ex1@topk    {
  X1(sed=sen),          % meningsbörjan
  X2(wordcl=pn),        % Jag
  X3(wordcl=vb),        % såg
  X4(wordcl=dt),        % den
  X5(wordcl=jj),        % lilla
  X6(wordcl=nn),        % hunden
  X7(),                 % behövs för att ange skiljetecknets plats
  X8(sed=sen)                 % meningsslut
-->
```

3.4 Notationen i högerledet

I högerledet behövs det möjligheter att utföra konsekvenser av matchningen. Högerledet består av olika fält; i dessa fält anges vad man vill göra med de matchningsvariabler som finns i vänsterledet. Alla fält utom **action** är optionella i högerledets syntax. Följande fält anger vad som skall utföras:

`action` I detta fält anges vilken typ av regel det är, det vill säga om det är en granskningsregel, sök- eller redigeringsregel, omtaggningsregel, accepterande regel eller hjälpregel. Se vidare i avsnittet **action**.

`corr` Här anges de matchningsvariabler som skall förändras. Se vidare i avsnittet **corr**.

`info` I detta fält skall den kommentar som skall presenteras vid utförd granskning anges. Se vidare i avsnittet **info**.

`jump` I detta fält anropas procedurer för hoppsatser som innebär att regelmatcharen skall fortsätta från ett specifikt avsnitt i regelsamlingen eller i indata. Se vidare i avsnittet **jump**.

`mark` Detta fält fylls i med det som skall markeras i texten för just denna matchning. Se vidare i avsnittet **mark**.

`link` I detta fält anges en länk till ett dokument som beskriver feltypen ytterligare. Se vidare i avsnittet **link**.

`detect` I detta fält anges meningar som skall detekteras av reglerna. Detta är användbart för att kontrollera om en regel fortfarande matchar det avsedda trots ändringar i regelkoden.

`accept` I detta fält anges meningar som inte skall detekteras av den regel som den anges i. Detta är användbart för att upptäcka om ändringar i regeln orsakar nya falska alarm.

3.4.1 Högerledets syntax

Högerledet, som börjar efter avskiljaren `-->`, måste innehålla följande element för att godkännas i regelspråket:

`-->` avskiljaren mellan vänster- och högerledet

`action()` action anger vilken regeltypen är, övriga fält är optionella.

`}` regelslut

3.4.2 Metoder för att manipulera objekten i högerledet

`no_of_tokens` `no_of_tokens` är en metod som räknar antalet token i en sekvensvariabel eller hjälpregel. Metoden returnerar ett heltal mellan 0 och n.

`substr (X, Y)` `substr` är en metod som returnerar delsträngar. Delsträngen anges med en startteckenposition `(X)` och en längd `(Y)` i strängen. En sträng i regelspråket börjar i position 0.

`length`	`length` returnerar längden i antal tecken för objektets textsträng.
`form(L)`	Metoden `form` tar en lista med lemma- och särdragsattribut satta till värden som det genererade ordet skall ha. Man behöver endast ange de särdrag som skall sättas om. Det genererade ordet returneras. Om listan är: `lemma:="bil", wordcl:=nn, gender:=utr, num:=sin, spec:=def, case:=nom` så returneras ordet *bilen*. Ursprungsformen tas bort automatiskt.
`join(Y)`	Metoden `join` tar text som argument enligt: `X.join(Y.real_text)` och sätter ihop `X` och `Y:s` textsträngar (`X` följt av `Y`) till en textsträng som returneras. Ursprungsträngarna tas bort automatiskt.
`insert(X)`	Metoden `insert` tar text, `X.text`, och sätter in det före det angivna objektet, enligt `Y.insert(X.text)`. `X` sätts in före `Y`.
`delete`	Metoden `delete` tar bort det angivna objektet.
`replace(X)`	Metoden `replace` tar text, `X.text`, som argument och ersätter det angivna objektet med `X.text`.
`donothing`	Metoden `donothing` gör ingenting med det angivna objektet, vilket är användbart i olika villkorsuttryck.

3.4.3 Funktioner i regelspråket

`spell_OK(str, token)`	Funktionen tar en sträng (ett ord) som första argument och returnerar `true` om ordet är rättstavat och `false` om ordet är felstavat. Det andra argumentet är `token`, vilket ger stavningskontrollen information om vilken typ av token som skall stavningskontrolleras.
`spell_corr(str)`	Funktionen tar en sträng (ett felstavat ord) som argument och returnerar ett ersättningsförslag. Om det saknas ersättningsförslag returneras tomma strängen `""`.
`smart_concat (X, Y)`	Funktionen sätter ihop två strängar och ser samtidigt till att korrigera för fler än två likadana konsonanter i rad.
`toupper (str)`	Funktionen tar en sträng som argument och returnerar alla bokstäver i strängen som versaler.

`tolower (str)`	Funktionen tar en sträng som argument och returnerar alla bokstäver i strängen som gemener.
`italics (w)`	Funktionen `italics` returnerar ordet kursiverat.
`bold (w)`	Funktionen `bold` returnerar ordet i fet stil.

3.4.4 If then else

För att kunna undersöka objekten ytterligare i högerledet vill man kunna skriva if-then-else-uttryck. If-then-else skrivs enligt nedan:

```
if expression then expression else expression end
```

3.4.5 Ett typfel tillåts i högerledet

För att underlätta regelskrivandet kommer vi att tillåta ett typfel, det skall räcka att endast ange matchningsvariabeln i fälten. Egentligen skall den textsträng som objektet innehåller anges med punktnotation. Det korrekta vore att skriva X.text, men vi förenklar, eftersom objekten alltid innehåller text, till att endast ange namnet på objektet för att få fram texten i högerledet: `X`. Observera att detta gäller endast på den yttersta nivån. Som parametrar till metoder och funktioner måste textattributet anges som brukligt är, det vill säga `X.text`.

3.4.6 Regeltypen anges i action

Vi vill använda regelspråket till att lösa flera olika uppgifter och vi delar därför in reglerna i olika regeltyper. I fältet **action** skall man ange vilken regeltyp det är. Följande typer finns:

`scrutinizing`	Granskningsregler för stavnings- och grammatikkontroll.
`tagging`	Regler för att göra om taggningen av en mening. I detta fält kan man sätta värden på attribut som skall gälla vid omtaggningen. Se vidare i avsnittet *Samarbete mellan taggaren och reglerna*.
`searching`	Regler för att söka efter lingvistiska enheter.
`editing`	Regler för att förändra, flytta eller ta bort lingvistiska enheter.
`help`	Hjälpregler som kan anropas av andra regler för att förbättra träffsäkerheten hos dessa. I detta fält sätts hela hjälpregelns attribut. Notationen blir: `action(help, särdragsattribut_1 :=värde_1 & särdragsattribut :=värde_2 & ... & särdragsattribut_n :=värde_n).` Se vidare i avsnittet *Hjälpregler*.

`accepting` Regler som används för att godkänna en struktur och som sedan gör ett hopp förbi ett antal felregler.

3.4.7 Reglernas syntax

En viktig skillnad mot Granskas tidigare regelspråk (Domeij et al, 1998) är att i detta nya regelspråk så finns det en tydlig relation mellan vänster- och högerled. Variablerna i vänster- och högerledet refererar till samma objekt. En regel som detekterar fraser med avvikande substantiv eller determinerare, till exempel *ett bilen* eller *den lilla huset*, kan formuleras enligt följande:

```
exregel@kongruens                    % regelnamn och regelkategori
{                                    % regelbörjan
  X(wordcl=dt),                      % en determinerare
  Y(wordcl=jj)*,                     % noll eller flera adjektiv
  Z(wordcl=nn & gender!=X.gender)    % ett substantiv
-->
  mark(X Y Z)                        % ord som skall markeras
  corr(X.form(gender:=Z.gender))     % determineraren ändras med
                                       genusvärdet från substantivet.ord
                                       som skall ändras anges.
  jump(kongruens_slut)               % hopp till lägen eller förbi token
  info("Kongruensfel")               % kommentarer till användaren
  link("www.kongruens.se" "Mer om")  % länk till mer information
  detect("Jag ser ett bilen")        % Regeln skall detektera
                                       denna konstruktion
  accept("Jag ser en bil")           % Regel skall släppa förbi denna
                                       konstruktion
  action(scrutinizing)               % regeltypen är scrutinizing
                                       (granskning)
}                                    % regelslut
```

3.4.8 Markering av orden anges i mark

Detta fält är till för att kunna markera ord eller tecken som är fel i den matchade sekvensen. Vi vill kunna göra flera saker i detta fält:

a) Markera alla element som har matchats i regeln – inget mark-fält anges i regeln.

b) Markera ett eller flera ord – de matchningsvariabler som skall markeras anges.

c) Markera endast en del av ett token, metoden för detta är `substr`, se ovan.

d) Markera en delsträng av det reguljära uttryck som matchats. Det görs genom att man anger vilken del av det reguljära uttrycket som skall markeras med hjälp av parenteser i det matchande uttrycket i vänsterledet. Dessa kan man sedan referera till i högerledet genom att de numreras från vänster till höger. Detta är under implementering (Baltatzis, forthcoming).

3.4.9 Ersättningsförslag i corr

Om man vill ändra något i det som har matchats i vänsterledet så anges det här genom att matchningsvariabeln anges med metoderna `form`, `delete`, `replace`, `insert` eller `join`. Endast de matchningsvariabler som skall ändras behöver anges.

I reglerna kan det se ut enligt följande:

```
intro@corr{
  X1(matchningskrav),
  X2(matchningskrav),
  ...,
  Xn(matchningskrav)
-->
  mark()
  corr(X1.delete() X2.replace(X1.text)… Xn.insert(X2.text)
  action()
}
```

3.4.9.1 Olika sätt att förändra texten

Det finns flera olika sätt att förändra det som har matchats i texten. Det som matchas i vänsterledet kan förändras genom att man:

a) tar bort matchningsvariabeln i högerledet, det vill säga `X Y Z --> X Z`

b) byter plats på variablerna i högerledet, det vill säga `X Y Z --> X Z Y`

c) sätter in textsträngar, det vill säga `X Y Z --> X Y "ny sträng" Z`

d) sätter in ersättningsförslag, det vill säga `X Y Z --> X "ersättningsförslag" Z`

Exempel 1:

En regel för att förändra något som matchat och endast det som är fel markeras. Exemplet visar hur felet dubbla mellanslag detekteras och korrigeras.

Indata: *Jag har vunnit på tipset.*

Utdata 1 (felet markeras): Jag har vunnit_på tipset.

Utdata 2 (felet åtgärdas): Jag har vunnit på tipset.

```
ex1@regexp  {
  ("\([a-zääöA-ZÅÄÖ]\)\([ ][ ]\)\([a-zääöA-ZÅÄÖ]\)"/
-->
  mark(”\2”)
  corr(”\1 \3”)
  action(scrutinizing)
}
```

Exempel 2:

Regeln skall ta bort element. Exemplet visar hur två *och* i rad tas bort:

Indata: *Jag kan inte spela och och det kan inte hon heller.*

Utdata 1 (felet markeras): *Jag kan inte spela och och det kan inte hon heller.*

Utdata 2 (felet åtgärdas): *Jag kan inte spela och det kan inte hon heller*.

```
ex2@ordregler     {
  X1(),
  X2(text=X1.text)
-->
  mark(X1 X2)
  corr(X1.delete())
  action(scrutinizing)
}
```

Exempel 3:

Ett exempel som visar hur två token sätts ihop till ett.

Indata: *Skolan har köpt ett cykel ställ.*

Utdata 1 (mark): Skolan har köpt ett cykel ställ.

Utdata 2 (corr): Skolan har köpt ett cykelställ.

```
ex3@saer    {
  X1(wordcl=dt),
  X2(wordcl=nn & gender!=X1.gender),
  X3(wordcl=nn & gender=X1.gender)
-->
  mark(X2 X3)
  corr(X2.join(X3.real_text))
  action(scrutinizing)  }
```

Exempel 4:

En regel som detekterar ett kongruensfel och tar fram ett ersättningsförslag som sätts in i texten.

Indata: Vi bor i den stora huset.

Utdata 1 (mark): Vi bor i den stora huset.

Utdata 2 (corr): Vi bor i det stora huset.

```
ex4@kong    {
  X1(wordcl=dt),
  X2(wordcl=jj),
  X3(wordcl=nn & gender!=X1.gender & gender=X2.gender & num=X2.num &
  spec=X2.spec)
-->
  mark(X1 X2 X3)
  corr(X1.form(gender:=X3.gender))
  action(scrutinizing)
}
```

3.4.9.2 Alternativa ersättningsförslag

Ibland vill man kunna ange flera olika ersättningsförslag i en regel. Det går därför att ge flera alternativa corr-fält i högerledet.

Exempel:

Ofta är det svårt att avgöra om det är artikeln eller substantivet som skall ersättas vid en feldetektion. Följande regel ger två olika förslag när artikeln och substantivet inte stämmer överens vad det gäller numerus.

Indata: *Jag såg en män som gick mot rött.*

Utdata från det första corr-fältet: *Jag såg några män som gick mot rött.*

Utdata från det andra corr-fältet: *Jag såg en man som gick mot rött.*

```
altcorr@kong
{
  X(wordcl=dt),
  Y(wordcl=nn & num!=X.num)
-->
  corr(X.form(num:=Y.num))          % byter ut artikeln
  corr(Y.form(num:=X.num))          % byter ut substantivet
  action(scrutinizing)
}
```

3.4.10 Regelprioritet och villkorliga hopp i jump

Med jump-satser kan vi gruppera regler och ordna dem inbördes. Reglerna exekveras i den ordning som de står i regelsamlingen och med jump-satser kan vi styra vilka regler som skall exekveras. Det går också att hoppa över regler som beskriver en mindre trolig tolkning av en språklig konstruktion. Med villkorliga hopp kan vi använda oss av så kallade accepterande regler som anger en sekvens som är helt korrekt och därför inte behöver undersökas mer (se Accepterande regler). Alla hopp måste ske framåt i regelsamlingen för att undvika oändliga loopar. Allt detta är möjligt i regelspråket med hjälp av jump-satserna **jump,** samt möjligheten att kunna sätta lägen (labels) i regelsamlingen, detta görs enligt:

```
läge:
```

Vänsterledet utgör det villkor som skall uppfyllas för att hopp skall ske till en regel, till regelsamlingens slut eller till ett token i indata.

Det går att hoppa förbi alla regler till slutet av regelsamlingen till det fördefinerade läget `endlabel` eller till ett egendefinerat läge. I båda fallen anges jump med ett argument, enligt:

```
jump(läge)
```

Det finns också ett fördefinierat läge som heter `beginlabel` som betecknar regelsamlingens början.

3.4.10.1 Exempel på användning av jump

Exempel 1:
Regelprioritet, när en regel är viktigare än en annan. Regelmatcharen tillämpar regeln för felaktig särskrivning, hoppar förbi alla inkongruensregler och fortsätter att exekvera regler.

Indata: *Skolan har köpt ett cykel ställ*

Utdata: *Skolan har köpt ett cykelställ*.

```
ex1@jump    {
  X1(wordcl=dt),
  X2(wordcl=nn & gender!=X1.gender),
  X3(wordcl=nn & gender=X1.gender)
-->
  mark(X2 X3)
  corr(X2.join(X3.real_text))
  jump(inkongruens_slut)
  action(scrutinizing)
}
....
```

```
Kongruensregler
...
inkongruens_slut:
...
andra regler
```

3.4.10.2 Att kunna hoppa förbi token i indata

Om man har en regel som matchat en följd av token är det ofta användbart att kunna hoppa förbi dessa vid nästa regelapplicering. För att kunna ange till vilket läge och hur många token fram i indata vi vill hoppa, använder vi **jump** med två argument `jump(läge, antal_token)`. Argumentet `läge` anger läget i regelsamlingen och `antal_token` anger hur många token vi vill hoppa förbi i indata.

Det går alltså att hoppa framåt i regelsamlingen samtidigt som regelmatcharen hoppar förbi token i indata, hur många token anges i jumps andra argument enligt nedan. Det första tokenet i matchningssekvensen har dock position 0, vilket medför:

```
jump(endlabel) <==> jump(endlabel, 0)
```

och det blir därför nödvändigt att dra bort ett token för att hamna rätt i indata:

```
jump(läge, antal_token - 1)
```

Exempel 1:

Indata: Vi har en ny bil som heter Volvo.

Utdata: "en ny bil" matchas och nästa regel appliceras på "som heter Volvo". Nästa regel som skall appliceras finns efter läget `inkongruens_slut`.

```
ex1@jump    {
  X0(wordcl=dt),
  X1(wordcl=jj),
  X2(wordcl=nn)
-->
  jump(inkongruens_slut, 2)
  action(accepting)
}
```

Exempel 2:

Med hjälp av metoden `no_of_tokens` kan vi skriva regler som kan hoppa över ett flexibelt antal token.

```
ex2@jump          {
  X0(wordcl=dt),
  X1(wordcl=jj)*,
  X2(wordcl=nn)
```

```
-->
  jump(endlabel, 1 + X1.no_of_tokens())
  action(accepting)     }
```

3.4.11 Kommentarer till användaren i info

En ytterligare förändring från tidigare versioner av Granskas regelspråk är att vi vill kunna ange kommentarer till användaren direkt i reglerna. Kommentaren som skall presenteras för användaren utgör också en utmärkt kommentar till själva regeln. Notationen är:

```
info("kommentar")
```

3.4.11.1 Generering av kommentarer

Man vill kunna generera olika kommentarer till de regler som är generella genom att sätta in olika ersättningsförslag i kommentarerna. Kommentaren byggs upp av strängar och matchningsvariabler som tillsammans bildar kommentartexten.

Exempel 1:

Utdata (i dialogruta): ”Stela och ålderdomliga ord. Ordet x är ålderdomligt, byt ut det mot y. Svarta listan, 1988.”

```
ex1@kom           {
VL()
-->
info("Stela och ålderdomliga ord. Ordet " X.text " är ålderdomligt byt
ut det mot" Y.text ". Källa: Svarta listan, 1988")
action(scrutinizing)}
```

Det finns också ett behov att få fram ordunika kommentarer till exempel för datatermer. Metoden `get_comment` som hämtar kommentarer kopplade till ord, kan användas för detta; den är dock inte implementerad.

3.4.12 Källhänvisningar till användaren i link

Mer omfattande kommentarer eller hänvisningar till böcker i hypertext, till exempel Svenska skrivregler (1991), sker med hyperlänkar i fältet link, enligt

```
link("hyperlänk" "hyperlänksnamn")
```

Exempel:

Programmet visar en länk till den aktuella skrivregeln.

```
ex1@link              {
VL()
-->
link("/granska/skrivregler/regel_120.html" "Skrivregel 120")
action(scrutinizing)    }
```

3.5 Olika typer av regler

I fältet action som beskrivits ovan skall man ange vilken typ av regel som skall appliceras. Nedan följer en genomgång av några av de regeltyper som finns förutom de granskningsregler som vi är vana vid. Ett avsnitt kommer också att visa hur regler skrivna i Finite State Grammar (Karttunen et al, 1996) och Constraint Grammar (Karlsson et al, 1995) kan implementeras i Granskas regelspråk. Sist beskrivs också något om hur reglerna skall exekveras.

3.5.1 Hjälpregler

Med hjälpregler menas regler som endast är till för att öka den generella lingvistiska analysen. En hjälpregel hjälper en annan regel med analysen, det kan till exempel vara en granskningsregel eller en annan hjälpregel.

Hjälpreglerna anger en sekvens av token i indata och kan ses som kontextfria regler på formen `A --> B C` (`A` skrivs om till `B` och `C`) oavsett hur kontexten ser ut till vänster och höger om `B` och `C`.

Det finns dock en möjlighet att ange en intern kontextkänslighet i reglerna genom

```
A --> B C(krav=B.krav)
```

det vill säga `A` skrivs om till `B` och `C`; om `C` har samma värde i `krav` som `B` har.

Det finns dock fall då denna uttryckskraft inte är tillräcklig för regelkonstruktören och därför finns det begränsade kontextkänsliga regler, dessa presenteras i avsnitt 3.5.2. En bra överblick med definitioner av kontextfri- och kontextkänslig grammatik finns t.ex. i Jurafsky & Martin (2000).

Exempel:

Vi vill skriva en hjälpregel på formen `NP --> dt nn,` vi anger också att kongruens måste råda mellan determineraren (`dt`) och substantivet (`nn`). Regeln matchar till exempel *en bil*, *den bilen*, *några bilar*, *ett hus*, *det huset*. Regeln matchar dock inte inkongruenta fraser som *en hus*, *ett huset* eller *några bil*.

```
NP@   {
X(wordcl=dt),
Y(wordcl=nn & gender=X.gender & num=X.num & spec=X.spec)
-->
action(help)
}
```

Det är i många fall praktiskt att kunna ange attribut och värden för en hel matchad sekvens (ofta fras). Vilka gemensamma attribut denna sekvens skall ha sätts i fältet `action` i högerledet, t.ex. om en nominalfras skall ha särdraget species och om detta i sin tur skall ha värdet bestämd form eller obestämd form. De attribut som kan användas är de som definieras för taggningen. Detta är en liten svaghet då det

ofta uppstår ett behov av att kunna ange lokala attribut i hjälpreglerna. Möjligheten att kunna ange lokala attribut bör beaktas vid en utvidgning av regelspråket; främst för att förenkla för regelkonstruktören.

Hjälpreglerna kan anges i andra regler. Det är dock inte tillåtet att använda operatorerna `+` och `*` tillsammans med en hjälpregel när de anges i andra regler, `?` går däremot bra. En hjälpregel skall kunna hanteras som ett objekt i övriga regler, det vill säga de metoder och funktioner som finns för objekten fungerar även för hjälpreglerna.

För att namnge regler används `@` efter regelnamnet, regelkategori är dock inte nödvändigt för hjälpregler.

```
hjälpregel@       {
  X1(),
  ...,
  Xn()
-->
  action(help)    }
```

När en hjälpregel anges i en annan regel anges den inom en parentes, enligt:

```
ex@help     {
  (hjälpregel)(),
  X(),
  Y()
-->
```

För att ytterligare öka möjligheterna att använda hjälpregler går det att ge dessa ett lokalt namn inom en regel enligt:

```
(Hjälpregel/Lokalt_namn)()
```

Med denna lokala namngivning kan man använda flera hjälpregler med samma namn men med olika innehåll i en regel. Hjälpregeln matchar till exempel kongruenta nominalfraser, vilket innebär att till exempel `X` skulle kunna matcha *den lilla hunden* och `Z` skulle kunna matcha *det stora huset* i regeln nedan:

```
exlokal@help {
  (hjälpregel/X)(),
  Y(),
  (hjälpregel/Z)(Z.attribut!=X.attribut)
-->
```

För att ange ett speciellt token i en hjälpregel gör man på samma sätt som i sekvensvariablerna och använder en vektornotation, `hjälpregel[n]`. Först dock ett exempel på hur man gör i sekvensvariabler.

Exempel 1:

Vi vill komma åt ett specifikt element i en sekvensvariabel:

Indata: Jag har sålt den lilla röda stugan i skogen

Utdata: Jag har sålt den röda stugan i skogen

```
ex1@help          {
  X1(wordcl=jj)+
-->
  mark(all)
  corr(if X1.no_of_tokens=2 then X1[1] else ”” end)
  action(editing) }
```

Exempel 2:

Vi vill komma åt ett specifikt element i en hjälpregel som anropas i en sökregel:

Indata: Jag har sålt den röda stugan i skogen.

Utdata: Jag har sålt stugan i skogen.

```
ex2@help                {
  (NP)()
-->
  mark(all)
  corr(if NP.no_of_tokens=3 then NP[2] else ”” end)
  action(searching)     }
```

Exempel 3:

Exemplet visar en regel som söker efter prepositionsfraser.

Indata: *Vi har köpt bensin till vår vita bil.*

Utdata: *Vi har köpt bensin till vår vita bil.*

Vi använder hjälpregeln som beskriver en nominalfras. I åtgärdsfältet sätter vi värden för hela nominalfrasen.

```
NP@   {
  X1(wordcl=ps),
  X2(wordcl=jj & gender=X1.gender & num=X1.num & spec=X1.spec),
  X3(wordcl=nn & gender=X2.gender & num=X2.num)
-->
  action(help, gender:=X1.gender, num:=X1.num, spec:=X1.spec)
      }
```

Sökregeln blir:

```
ex3@help                {
  X1(wordcl=pp),              % prepositionen
```

```
  (NP)()                        % nominalfrasen
-->
  mark(X1 NP)
  action(searching)      }
```

Exempel 4:

Nominalfraser som samordnas med konjunktion skall vanligen vara symmetriska. Vi skriver en regel som letar upp nominalfraser som inte är symmetriska.

Indata: *Sportaffären säljer dyra drag och billigt spö.*

Utdata 1 (orden markeras): *Sportaffären säljer dyra drag och billigt spö.*

Utdata 2 (efter korrektion): *Sportaffären säljer dyra drag och billiga spön.*

Först en hjälpregel:

```
NP@   {
  X(wordcl=jj),
  Y(wordcl=nn & gender=X.gender num=X.num & spec=X.spec)
-->
  action(help, gender:=Y.gender, num:=Y.num, spec:=X.spec)
      }
```

Granskningsregeln blir:

```
ex4@help                {
  (NP/X)(),
  Y(wordcl=kn),
  (NP/Z)(num!=X.num | spec!=X.spec)
-->
  mark(X Y Z)
  corr(Z[0].form(gender:=X.gender, num:=X.num, spec:=X.spec)
  Z[1].form(gender:=X.gender, num:=X.num, spec=X.spec))
  action(scrutinizing)  }
```

3.5.1.1 Optionella hjälpregler

Optionella hjälpregler är mycket användbart om man vill skriva generella regler. Det handlar ofta om fraser som ibland finns med en i sekvens men inte alltid, som till exempel adjektiven i en nominalfras (se exempel nedan). En optionell hjälpregel anges med `?`, enligt:

```
(Hjälpregel)()?
```

Exempel:

Vi vill skriva en granskningsregel som detekterar fraser där artikeln och substantivet är inkongruenta. Vi vill också att det skall kunna förekomma adjektiv

mellan artikeln och substantivet. Vi skall detektera fraser med adjektiv och fraser utan adjektiv. Om fraserna innehåller adjektiv skall dessa kongruera med substantivet.

Detektion av kongruensfel med adjektiv:

Indata : *Jag såg en litet hus i skogen.*

Utdata: *Jag såg ett litet hus i skogen.*

Detektion av kongruensfel utan adjektiv:

Indata : *Jag såg en hus i skogen.*

Utdata: *Jag såg ett hus i skogen.*

Först anger vi hur hjälpregeln för adjektivfrasen skall se ut. Adjektivfrasen skall innehålla ett eller flera kongruenta adjektiv:

```
JJ@
{
X(wordcl=jj),
Y(wordcl=jj & (gender=X.gender & num=X.num & spec=X.spec))*,
-->
action(help, gender:=X.gender, num:=X.num, spec:=X.spec)
}
```

Därefter skriver vi granskningsregeln:

```
kong22@inkongruens
{
  X(wordcl=dt),
  (JJ/Y)()?,
  Z(wordcl=nn & (gender!=X.gender | num!=X.num | spec!=X.spec) &
      (Y.no_of_tokens=0 | (gender=Y.gender & num=Y.num & spec=Y.spec))
-->
  mark(all)
  corr(X.form(gender:=Z.gender, num:=Z.num, spec:=Z.spec) Y Z)
  action(scrutinizing)
}
```

3.5.2 Kontextkänsliga hjälpregler

Många hjälpregler bör kunna skrivas som kontextfria regler, men i vissa fall är det opraktiskt att hela det matchade området från en hjälpregel konkateneras med det matchade området från en annan regel. Man vill ofta ”tjuvkika” på kontexten utan att den tas med i själva matchningen. En stor fördel är att kontexten därmed kan användas i matchningen av en annan regel, vilket är mycket användbart.

Kontextkänsliga regler kan också användas på Constraint Grammar-liknande sätt för att skapa nya lingvistiska kategorier på ord eller flera ord (mer om detta i slutet av detta avsnitt). Det finns möjligheter att skriva begränsade kontextkänsliga hjälpregler i regelspråket. Den begränsning som finns är att vänsterkontexten måste vara bestämd i antalet ord. Anledningen är att Granskas matchningsalgoritm är optimerad för statisk vänsterkontext – om sekvensvariabler vore tillåtna i vänsterkontexten skulle matchningen ta betydligt längre tid.

Vänsterkontexten anges mellan regelbörjan `{` och det reserverade ordet `ENDLEFTCONTEXT`. Det som skall vara synligt för de andra reglerna anges mellan `ENDLEFTCONTEXT` och det reserverade ordet `BEGINRIGHTCONTEXT`, detta kan vi kalla regelkropp. Regelkroppen kan bestå av en eller flera matchningsvariabler. Högerkontexten anges mellan `BEGINRIGHTCONTEXT` och avskiljningspilen `-->`.

Om vi har en klassisk kontextkänslig regel som: `A --> B /LC _ RC`, så blir den översatt till regelspråket:

```
A@                          % A, regelhuvud
{
  vänsterkontext,           % LC
  ENDLEFTCONTEXT,
  B(),                      % B, regelkropp

  BEGINRIGHTCONTEXT,
  högerkontext              % RC

-->
action(help)
}
```

Exempel:

I svenskan finns det adjektiv som används som självständiga nominalfraser med såväl framförställda som efterställda attribut. En vanlig adjektivform bland dessa är den som slutar på *-a*, till exempel *unga* och *gröna*. Följande förenklade regel skiljer ut dessa från de adjektiv som ingår i nominalfraser med substantiv som huvudord.

```
NP_jj@      {
X(wordcl!=dt & wordcl!=ps),                % vänsterkontexten får inte
                                           % innehålla determinerare
                                           % eller possessiv
ENDLEFTCONTEXT,
Y(wordcl=jj & substr(length-1, 1) = "a"), % regelkroppen utgörs av ett
                                           % adjektiv som slutar på -a
BEGINRIGHTCONTEXT,
Z(wordcl!=jj & wordcl!=nn & wordcl!=kn)    % högerkontexten får inte
                                           % innehålla adjektiv,
                                           % substantiv
                                           % eller en konjunktion.
-->
action(help)
}
```

Kontextkänsliga hjälpregler är ett kraftfullt sätt att avgöra t.ex. en eller flera ords konstituenttillhörighet eller grammatiska funktion. Angreppssättet påminner mycket om det i Constraint Grammar. I Constraint Grammar finns det dock färre begränsningar av hur vänsterkontexten får anges, olika typer av räknare är till exempel tillåtna. Å andra sidan är regelkroppen (domain) i Constraint Grammar begränsad till ett ord, medan den i Granskas regelspråk kan bestå av ett eller flera ord.

Med kontextkänsliga regler kan den lingvistiska analysen i Granska förbättras och förfinas. Regler för satsgränsidentifikation har redan implementerats. Syntaktiska funktioner som subjekt och objekt bör också implementeras med kontextkänsliga regler för en generellare och ”högre” beskrivning. Ord eller sekvenser av ord kan få nya syntaktiska taggar (det vill säga namnet på kontextkänsliga hjälpregler) som t.ex. subjekt eller objekt. Den enkelhet som finns i Constraint Grammar med regler som i stort sett är oberoende av andra regler och där flertydighet successivt kan elimineras saknas i Granskas regelspråk. I regelspråket måste en stegvis flertydighetseliminering utföras av antingen mer komplexa regler som tar hänsyn till fler fall eller av regler som ”känner varandra” och som kopplas ihop. Detta område är ganska outforskat i regelspråket och kräver en realistisk tillämpning för att mer generella uttalanden skall kunna göras.

3.5.3 Samarbete mellan taggaren och reglerna

Vi har upptäckt att den probabilistiska taggaren taggar fel enligt återkommande mönster. Dessa mönster vill vi fånga upp och tagga om. Vi kan göra det enligt följande:

1. En regel detekterar en sekvens som vi vet att taggaren ofta taggar fel.
2. Vi vill kunna ange att det skall finnas andra tolkningar än den taggaren har kommit fram till för att regeln skall tillämpas.
3. Vi väljer en tolkning som vi vet är mer riktig och vi anger den och den skall gälla vid omtaggningen.
4. Meningen taggas om.
5. Ny regelexekvering.

Vi uppnår det genom:

1. Vi matchar den taggning som vi vet är fel i vänsterledet.
2. Vi undersöker det felaktigt taggade objektet ytterligare genom att ange att den förhoppningsvis korrekta tolkningen skall existera i strukturen lex.
3. Genom att ange i fältet **action** att regeln är en taggningsregel så kan vi sätta värden på det feltaggade objektets attribut, för att kunna utgå ifrån denna vid nästa taggning. Med notationen X.särdragsklass:=särdragsvärde sätts den tagg som skall gälla vid omtaggningen.
4. Vi väljer tagging i fältet **action** i regeln, så tolkas det som att meningen skall taggas om.
5. När taggningen är färdig appliceras de resterande reglerna på den omtaggade meningen och resten av texten.

Vi vill också kunna förändra en mening genom att ta bort, lägga till eller byta plats på ord för att sedan tagga om den förändrade meningen. Den förändrade delen av meningen anges i fältet **corr** och meningen taggas om utifrån denna förändring.

Notationen för fältet **action** vid taggning blir:

```
action(tagging, särdrag_1:=värde_1,…, särdrag_n:=värde_n)
```

Exempel 1:

Felaktig taggning av artikeln i en nominalfras.

Indata: Taggaren har taggat *den* i *den lilla pojken* som pronomen.

Utdata: Taggaren taggar *den* i *den lilla pojken* som determinerare.

```
ex1@tagg    {
  X1(wordcl=pn & E(lex.wordcl=dt)),
  X2(wordcl=jj & gender=X1.gender & num=X1.num & spec=X1.spec),
  X3(wordcl=nn & gender=X2.gender & num=X2.num & spec=X2.spec )
-->
  action(tagging, X1.wordcl:=dt)
}
```

Exempel 2:

Särskrivningen skrivs ihop och vi vill tagga om meningen.

Indata: *Skolan har köpt ett cykel ställ*

Utdata: Taggaren taggar meningen *Skolan har köpt ett cykelställ.*

```
ex2@tagg    {
  X1(wordcl=dt),
  X2(wordcl=nn & gender !=X1.gender),
  X3(wordcl=nn & gender=X1.gender)
-->
  corr(X2.join(X3.real_text))
  action(tagging)
}
```

3.5.4 Accepterande regler

Accepterande regler är ofta användbara för att acceptera ord och fraser som är korrekta och inte behöver granskas med olika felregler.

Exempel:

En korrekt nominalfras undviker regelapplicering med regler för inkongruenta nominalfraser.

Indata: *Jag bor i det lilla huset*.

Utdata: *Jag bor i det lilla huset*. % Oförändrat

```
ex1@acc     {
  X1(wordcl=dt),
  X2(wordcl=jj & gender=X1.gender & num=X1.num & spec=X1.spec),
  X3(wordcl=nn & gender=X2.gender & num=X2.num & spec=X2.spec)
-->
  corr(all)
  jump(inkongruens_slut)
  action(accepting)
```

```
}
...
inkongruensregler
...
inkongruens_slut:
...
andra regler
```

3.5.5 Union och subtraktion mellan regler

Vi vill uttrycka union och subtraktion för att kunna implementera generella och kraftfulla regler i regelspråket. Inspiration till dessa två operatorer har hämtats genom det samarbete som Granska-projektet hade med Institutionen för lingvistik vid Göteborgs Universitet[12].

Union kan vi uttrycka med `;` och för subtraktion mellan två regler använder vi tilde `~`. `A` och `B` är regler.

Subtraktion, `A-B <==> A ~ B` vilket betyder: om uttrycket skall bli sant gäller det att det som detekteras av regel `A` inte skall detekteras av regel `B`.

Union, `A | B <==> A ; B` vilket betyder att elementen måste detekteras av regel `A` eller regel `B` för att uttrycket skall bli sant.

Subtraktion mellan regler utgör ett specialfall och detta måste anges, vi gör det genom att omge reglerna med dubbla klammerparenteser enligt:

```
{{Regel_1 ~ Regel_2 --> kommentar}}
```

Högerledet består endast av ett kommentarfält och det område som matchas markeras automatiskt med rött.

Exempel:

Följande exempel är hämtat från Cooper & Sofkova (1998). Regelformalismen är Xerox Finite State Transducer-formalism (Karttunen et al, 1996), vilken i stort är en formalism för reguljära uttryck och omskrivningsregler. Exemplet visar hur avvikelser i species kan detekteras med enbart positiva regler, det vill säga regler som anger korrekta fragment i språket. Regler kan upptäcka fel som *en lilla bil* och *ett litet huset*.

% Den grova regeln (Broad grammar)

```
define      NP [dt jj* nn]
```

% De finare reglerna (Narrow grammar)

```
define      NPDef[dt_def jj_def* nn_def]
```

[12] Det projekt för automatisk språkgranskning som bedrevs vid Institutionen för lingvistik vid Göteborgs Universitet (http://www.ling.gu.se/~sylvana/FSG/) finns beskrivet i kapitel 2 i denna avhandling.

```
define      NPIndef[dt_ind jj_ind* nn_ind]
```

% Granskningsregeln, frasen (språket) skall finnas i `NP` men inte i `NPDef` eller `NPIndef`

```
NP-[NPDef | NPIndef]
```

I Granska löser vi det genom att först skriva två hjälpregler och sedan skriva själva granskningsregeln:

% Hjälpregel 1:

```
NP@               {
  X1(wordcl=dt),
  X2(wordcl=jj)*,
  X3(wordcl=nn)
-->
  action(help)    }
```

% Hjälpregel 2:

```
NPunion@          {
  X1(wordcl=dt & spec=def),
  X2(wordcl=jj & spec=def)*,
  X3(wordcl=nn & spec=def)
-->
  action(help)
;
  X1(wordcl=dt & spec=ind),
  X2(wordcl=jj & spec=ind)*,
  X3(wordcl=nn & spec=ind)
-->
action(help)      }
```

% Granskningsregeln:

```
{{(NP)() ~ (NPunion)() --> info("kommentar")}}
```

3.5.6 Rekursiva hjälpregler

En effektiv beskrivning och implementation av svenska nominalfraser kräver rekursion. I regelspråket är alla rekursiva regler tillåtna, förutom vänsterrekursiva regler (d.v.s. regler där det rekursiva anropet ligger längst till vänster). Följande minigrammatik visar hur rekursiva hjälpregler kan skrivas. Grammatiken består av reglerna `NP`, `NPmin`, `NPmPP` och `PP` som beskrivs nedan. Grammatiken matchar till exempel följande satser: *mannen*, *mannen på taket*, *mannen på taket i det gula huset*, *mannen på taket i det gula huset vid ån.*

En nominalfras kan bestå av en minimal nominalfras (NPmin) eller en nominalfras med en eller flera prepositionsfraser som attribut (NPmPP).

```
NP@          {                  % NP --> NPmin | NPmPP


  (NPmin)() --> action(help); % stoppvillkor för PP
  (NPmPP)() --> action(help)
             }
```

Den minimala nominalfrasen NPmin detekterar t.ex. *bilen*, *en bil*, *en liten bil*, *min lilla gröna bil.*

```
NPmin@             {           % NP --> (DT|PS) (JJ) NN
  X(wordcl=dt | wordcl=ps)?,  % optionell determinerare eller possessiv
  Y(wordcl=jj)*,              % noll eller flera adjektiv
  Z(wordcl=nn)                % ett substantiv
-->
action(help)       }
```

En nominalfras med en eller flera prepostionsfraser som efterställda attribut kan anges med följande regel. I denna regel sker ett rekursivt anrop; PP anropar NP som kan bestå av NPmPP.

```
NPmPP@             {     % NP --> NPmin PP
  (NPmin)(),             % det går inte att anropa NP här, då hade regeln
                         blivit vänsterrekursiv eftersom NP --> NPmPP.
  (PP)()                 % en eller flera prepositionsfraser
-->
  action(help)}
```

Prepositionsfrasen kan bestå av en eller flera prepositionsfraser, eftersom NP kan bestå av en prepositionsfras. Här sker ett rekursivt anrop; PP anropar NP som kan bestå av NPmPP som anropar PP.

```
PP@          {           % PP --> pp NP
  X(wordcl=pp),          % en preposition
  (NP/Y)()               % en nominalfras som kan innehålla en eller
                         flera PP
-->
  action(help)}
```

3.5.7 Två regler i Constraint Grammar och i regelspråket

En jämförelse med regler skrivna i Constraint Grammar (CG) är intressant främst av tre anledningar. För det första har inspiration till regelspråket hämtats från CG och för det andra är CG långt mer känt som formalism och det kan därför vara intressant att visa hur CG-regler ser ut i Granskas regelspråk. Dessutom finns det en grammatikkontroll skriven i CG för svenska (Birn, 2000). De följande reglerna

är hämtade ur Birn (2000) och visar först förenklat hur inkongruens i nominalfrasen detekteras, t.ex. *ett villa*.

```
("<ett>" =s!@ERR)        ; Utläses: För ordformen Ett/ett välj (=s!)
                         feltaggen
                         ; (@ERR),
(1C N-UTR))              ; om nästa ord till höger är ett entydigt
                         substantiv
                         ; i utrum (1C N-UTR)
```

Regeln får följande utseende i Granskas regelspråk:

```
cg1@kong    {
  X(text="ett"),                          % ("<ett>" =s!@ERR)
  Y(A(lex.wordcl=nn & lex.gender=utr))     % (1C N-UTR))
-->
  corr(X.form(gender:=utr))
  action(scrutinizing)
}
```

En regel för att hitta bestämdhetsfel som *Pelles gula bilen* ser ut enligt följande i CG:

```
(@w=s!(@ERR)
      (0 N-DEF)              ; Substantiv i bestämd form skall taggas
                             som fel,
      (-2 GEN)               ; om det andra ordet till vänster om
                             substantivet står i genitiv,
      (-1 A-DEF))            ; och om det första ordet till vänster om
                             substantivet är ett adjektiv i bestämd
                             form
```

I Granskas regelspråk kan regeln konstrueras enligt följande:

```
cg2@kong
{
  X(case=gen),                      % (-2 GEN)
  Y(wordcl=jj & spec=def)           % (-1 A-DEF))
  Z(wordcl=nn & spec=def)           % (0 N-DEF)
-->
  corr(Z.form(spec:=ind))
  action(scrutinizing)
  }
```

En stor skillnad mellan reglerna är att högerledet i CG-reglerna är i stort begränsat till om taggar skall väljas eller förkastas, medan man i Granskas regelspråk kan ange vilka ord som skall ersättas, vilket har att göra med att CG inte främst är en formalism för automatisk språkgranskning. Detta visar dock styrkan i CG-formalismen; den kan användas för andra ändamål än den designades för. Å andra sidan kan man se grammatikkontroll som ett disambigueringsproblem (vilket CG är designat för). Det gäller att välja mellan en tolkning som är rätt eller fel.

3.6 Hjälp för regelkonstruktören

Att kunna skriva kommentarer, spåra exekveringen av reglerna samt få hjälp med syntax- och semantikkontroll är sådant som underlättar arbetet för regelkonstruktören.

3.6.1 Kommentarer i reglerna

Kommentarer inleds med `(*` och avslutas med `*)` som i Pascal. En kommentar som gäller endast en rad inleds med `%`.

Exempel:
Kommentarer till en accepterande regel.

```
(*
 * regel 51
 * Accepterande regel som undviker att korrekta
 * nominalfraser tolkas som felaktiga
 *)
ex@kom            {
  X1(wordcl=dt),         % Artikeln
  X2(wordcl=jj & gender=X1.gender & num=X1.num & spec=X1.spec),
  X3(wordcl=nn & gender=X2.gender & num=X2.num & spec=X2.spec)
-->

(* Frasen markeras inte, exekveringen fortsätter efter
inkongruensreglerna *)

  jump(inkongruens_slut)
  action(accepting)
}
```

3.6.2 Syntax- och semantikkontroll

Regelmatcharen gör en fullständig syntaxkontroll av regelfilen och talar för varje fel om vilken rad i regelkoden som är fel och vilken typ av fel det är. Regelmatcharen har olika typer för regler, element, lägen, särdragsklasser,

särdragsvärden, texter, heltal, flyttal och booleska värden. Typkonflikter ger felmeddelanden. Särdrag i olika särdragsklasser kan inte jämföras med varandra.

Regelmatcharen kontrollerar också fälten `detect` och `accept` i reglerna för att förhoppningsvis upptäcka om regelkonstruktören har gjort några fel som inte beror på syntaxfel. Detta är ett stöd för att upptäcka att angivna exempelmeningar i `detect` och `accept` fortfarande detekteras respektive godkänns trots förändringar i regelkoden. I `detect` kan man ange en felaktig mening med t.ex. en särskrivning som *Vi har grupp arbete just nu* och i `accept` kan man ange en korrekt mening som *Vi har en grupp studenter på besök*. Om regelkonstruktören har förändrat en regel så att dessa inte detekteras respektive godkänns upptäcker regelmatcharen det och ett felmeddelande presenteras för regelkonstruktören.