En metod för svensk rättstavning baserad på bloomfilter

Joachim Hollman Viggo Kann[1]

Teoretisk datalogi
Institutionen för numerisk analys och datalogi (NADA)
KTH
S–100 44 Stockholm

TRITA–NA–P9213

IPLAB 54

9 september 1993

[1]Datorpost: joachim@nada.kth.se, viggo@nada.kth.se

Sammanfattning

Vi beskriver en metod för att hitta stavfel och generera rättstavningsförslag i en svensk text. Metoden grundar sig på en probabilistisk algoritm och skiljer sig avsevärt från traditionella metoder för stavningskontroll.

Ordlistan lagras mycket kompakt med hjälp av bloomfilter som bygger på upprepad hashning i en och samma tabell. Med denna lagring går det mycket snabbt att avgöra om ett ord finns med i ordlistan.

Ytterst sällan, med en sannolikhet som kan väljas i förväg, gör algoritmen fel och godkänner ett ord som inte finns i ordlistan. Detta sker så sällan att det inte betyder något i praktiken, men det gör att det inte går att utvinna ursprungsordlistan. Vi visar att enda sättet att skydda ursprungsordlistan helt är att stavningsalgoritmen svarar fel ibland.

För att kunna ge rättstavningsförslag och ytterligare minska sannolikheten för fel använder vi en grafotaktisk tabell som anger vilka fyrbokstavsföljder som finns i språket. Vi diskuterar hur en metrik för stavfel kan se ut och hur metoden automatiskt kan godkänna böjningsformer och sammansättningar. Slutligen beskriver vi hur STAVA, vår implementation av metoden, fungerar, och ger resultat av tester.

1 Inledning

Vi har i flera år intresserat oss för språkverktyg, från början bara för användning i vårt dagliga arbete men efter hand alltmer på djupet. Med vår bakgrund som forskare inom teoretisk datalogi har det fallit sig naturligt att framför allt koncentrera sig på algoritmerna bakom dessa verktyg.

Vi vill förskjuta tyngdpunkten mot metoder som har en stabil matematisk och datalogisk grund, metoder anpassade för verkligt stora datamängder och moderna datorarkitekturer. Vi har tagit fram en metod för att upptäcka stavfel i en text. Metoden i sig är språkoberoende och kan därför användas för vilket naturligt språk som helst. Det enda som krävs är en ordlista.

STAVA, en preliminär implementation av metoden med en svensk ordlista om ca 200 000 ord, har använts på KTH under ett par år och fungerat mycket bra. Programmet finns dessutom allmänt tillgängligt över världens forskningsdatornät via s k anonym ftp[1].

Vid en oberoende utvärdering av program för svensk stavningskontroll [4] befanns STAVA vara näst bäst, klart bättre än tex alla stavningskontrollerare som är inbyggda i ordbehandlare. STAVA var klart snabbast och mest flexibelt. Programmet CAPS ansågs dock bäst eftersom det har hela SAOL (Svenska Akademiens ordlista) som databas. Det är dock inte odelat positivt med en stor orddatabas, se avsnitt 8.

Vår metod grundar sig på en probabilistisk algoritm och skiljer sig avsevärt från traditionella metoder för stavningskontroll. Den har följande utmärkande egenskaper.

- Metoden ger mycket snabb kontroll av om ett ord finns i ordlistan.
- Tiden för stavningskontrollen är oberoende av ordlistans storlek.
- Storleken på ordlistan reduceras till mindre än en femtedel.
- Ordlistan lagras i krypterad form som *omöjligt* kan dekrypteras, vilket är en förutsättning för kommersiell spridning. Ett rättstavningsprogram som godkänner exakt de ord som finns i ordlistan kan aldrig skydda sin ordlista, hur bra krypterad den än må vara. En modern dator kan nämligen kontrollera alla rimliga bokstavskombinationer och därigenom återskapa hela ordlistan på ett par timmar.
- Trots den valda lagringsformen (en tabell med ettor och nollor, se vidare avsnitt 3) kan rättstavningsförslag ges.
- Det är enkelt att utöka ordlistan med egna ordlistor.
- Sammansättningar och böjda former av orden i ordlistan kan automatiskt godkännas om så önskas.
- Sannolikheten för att ett felstavat ord godkänns kan enkelt varieras. Med ovan angivna prestanda är sannolikheten 0,007 procent. Ett jordnära exempel: vår metod tillämpad på en vanlig roman där så mycket som vart femte ord är felstavat skulle endast släppa igenom ett enda felstavat ord.

För att ge rättstavningsförslag på felstavade ord genereras alla ord som ligger nära det felstavade ordet. Rimligheten hos dessa ord kontrolleras med hjälp av två algoritmer, dels den probabilistiska, dels en algoritm som bygger på tillåtna bokstavskombinationer. De ord som

[1] Anonym ftp: `ftp.kth.se`, `/kth/misc`

klarar båda testerna presenteras som rättstavningsförslag. För att avgöra vad som är *nära det felstavade ordet* har vi infört en metrik, ett avståndsmått mellan ord.

Vi har många idéer om hur vår metod för stavningskontroll och rättstavningsförslag ska förbättras. Bland annat vill vi ägna mer forskning åt att förfina metriken för att ge bättre rättstavningsförslag. Vi anser att det är viktigt att komma ifrån de ad hoc-mässiga metoder som normalt sett används för att generera förslag. Genom att bygga vår algoritm på matematisk grund når vi ett resultat som kan användas även för andra språk än svenska.

En av anledningarna till att utveckla en snabb och säker stavningskontroll, som inte klagar på för många rättstavade ord, är att det i framtiden kan förväntas bli vanligt med överföring av stora mängder text från papper till dator med hjälp av OCR, optisk teckenigenkänning. Vid denna typ av stavningskontroll krävs en radikalt annorlunda metrik eftersom felläsningar ger upphov till en annan sorts fel än då en människa skriver maskin.

Liknande problem finns vid tolkning av handskriven text, t ex i de nya pennbaserade användargränssnitten.

Ibland räcker det inte att använda endast en ordlista. Hur stor den än är så finns det alltid ord som saknas, nämligen ovanliga böjningsformer och långa sammansättningar. När man inte vill drunkna i ord som stavningsprogrammet tror kan vara felstavade, t ex vid rättstavning av mycket stora mängder text, kan det vara önskvärt att programmet klarar av att känna igen alla böjningsformer och sammansättningar. Visserligen kan det då slinka igenom några felstavade ord, men tiden för att utföra stavningskontrollen blir betydligt kortare. Ifall inte någon enda felstavning kan accepteras kan man avslutningsvis göra en stavningskontroll med enbart ordlistan.

2 Krav på datastrukturen och algoritmen

Vid konstruktionen av vår metod för stavningskontroll hade vi följande egenskaper i åtanke.

- Snabbt — det ska gå mycket fort att avgöra om ett ord är med i ordlistan.
- Utrymmessnålt — ordlistan får inte ta för stor plats, även om den innehåller hundratusentals ord.
- Utbyggbart — nya ordlistor ska kunna läggas till.
- Skyddad ordlista — originalordlistan ska inte kunna utvinnas i klartext.
- Ord behöver aldrig tas bort ur ordlistan.
- Vi kan tillåta fel någon gång ibland.

De fyra första punkterna är krav som vi tycker att ett bra program för stavningskontroll ska uppfylla, även om inget svenskt kommersiellt program ännu tycks göra det. De två sista punkterna är begränsningar som vi utnyttjar för att kunna tillfredsställa de fyra kraven.

Kravet att ordlistan ska vara skyddad leder automatiskt till att metoden måste göra fel ibland, se avsnitt 7

```
var v: array [0..tabellstorlek] of boolean;

function kollastavning(ordet:ordtyp; langd:integer):boolean;
var bar, nr, index:integer;
begin
     bar:= antalhashfunktioner+1;
     nr := 1;
     while nr < bar do begin
          index:= hashning(nr, ordet, langd);
          if v[index]=true then nr:= nr+1
          else bar:= nr;
     end;
     kollastavning:= (bar = antalhashfunktioner+1);
end (* kollastavning *);
```

Figur 1: *Funktion som kontrollerar om ordet* `ordet` *finns i ordlistan som lagras med bloomfilter i vektorn* v.

3 Bloomfilter

Den sedan länge dominerande metoden för snabb sökning kallas hashning (söknyckelordstransformation). Idén är att tilldela varje söknyckel ett heltal som anger var i minnet motsvarande post lagras. Funktionen som transformerar söknycklar till tal måste gå snabbt att beräkna och ge god spridning för att resultatet ska bli bra.

Om problemet bara består i att testa medlemskap i en mängd (t ex om ett ord finns med i en ordlista) så kan en snarlik metod, s k bloomfilter, användas [1]. Det är främst två punkter som skiljer: varje post utgörs av en nolla eller en etta och flera hashningar görs i samma tabell med olika transformationsfunktioner.

Ett ord införs genom att alla transformationsfunktioner tillämpas på ordet och motsvarande poster ettställs. Man kontrollerar sedan om ett ord finns med i ordlistan genom att beräkna transformationsfunktionerna och se efter om alla poster är ettställda, se figur 1.

Det kan tänkas att ord som inte finns med i ordlistan ändå accepteras om alla ordets poster råkar vara ettställda (på grund av kollisioner med andra ord). Sannolikheten för detta kan enkelt regleras genom att ändra antalet transformationsfunktioner och storleken på tabellen. För en perfekt balanserad tabell är sannolikheten för fel $1/2^k$, där k är antalet transformationsfunktioner, se avsnitt 6. Som synes är detta en mycket liten sannolikhet redan för små värden på k.

Den lilla risken för fel gör dock att det både praktiskt och teoretiskt är omöjligt att återvinna den ursprungliga ordlistan ur den lagrade tabellen.

Som hashalgoritm används Knuths välkända metod, se figur 2.

4 Rättstavning, metrik för ord

Många undersökningar (se t ex Damerau [2] och Peterson [7]) visar att 80 till 90 procent av alla skrivfel vid maskinskrivning kan hänföras till någon av följande kategorier:

```
var primtal: array [1..antalhashfunktioner] of integer;

function hashning(nr:integer; ordet:ordtyp; langd:integer):integer;
var sum, i: integer;
begin
     sum:= kodabokstav(ordet[1]) * 256 * 256;
     for i:= 2 to langd do
        sum:= (sum*128 + kodabokstav(ordet[i])) mod primtal[nr];
     hashning:= sum;
end (* hashning *);
```

Figur 2: *Funktion som transformerar ett ord till ett index i vektorn* `v` *på olika sätt beroende på värdet av argumentet* `nr`*. Vektorn* `primtal` *innehåller de* `antalhashfunktioner` *primtalen som är närmast mindre än* `tabellstorlek`*.*

1. omkastning av två efter varandra följande tecken,
2. ett extra tecken,
3. ett utelämnat tecken,
4. ett felaktigt tecken.

En väl fungerande metod för att gissa sig till hur ett felstavat ord egentligen skulle ha sett ut är att generera alla ord som ges av de fyra kategorierna ovan och kontrollera vilka av dessa som är riktiga.

Vi säger att de ord som bildas på detta sätt ligger på *avstånd ett* från ursprungsordet. Om inga riktiga ord finns på avstånd ett kan man tänka sig att fortsätta leta efter ord på avstånd två, d v s ord som bildas genom två av ovanstående skrivfel, och så vidare.

Denna metrik fungerar alltså väl för skrivfel som uppstår vid maskinskrivning. Andra metriker måste användas vid rättstavning av text som lästs in med hjälp av optisk teckenigenkänning [8] eller på annat sätt, t ex genom igenkänning av tal eller handskrift.

Vi kan räkna hur många ord som kan genereras med ovanstående fyra regler utgående från ett ord med b bokstäver. Anta att alfabetet består av a bokstäver.

1. Omkastning kan göras på $b - 1$ sätt.
2. En bokstav kan tas bort på b sätt.
3. En bokstav kan läggas till på $a \cdot (b + 1)$ sätt.
4. En bokstav kan ersättas på $(a - 1) \cdot b$ sätt.

Sammantaget ger detta $2ab + a + b + 1$ genererade ord. Om vi räknar med ett alfabetet innehåller 29 bokstäver, d v s $a = 29$, får vi $59b + 28$ ord. För ett tiobokstavsord genereras alltså 618 ord. Även om sannolikheten för att ett godtyckligt tiobokstavigt felstavat ord ska godkännas är så liten som 0,007 procent så kommer sannolikheten för att ett ord som inte

```
function grafkoll(ordet:ordtyp; langd:integer):boolean;
label 999;
var i:integer; kodatord:kodatordtyp;
begin
     kodatord[0]:= specialtecken;
     for i:= 1 to langd do kodatord[i]:= kodabokstav(ordet[i]);
     kodatord[langd + 1]:= specialtecken;
     for i:= 0 to langd-2 do
        if not gtabell[kodatord[i],kodatord[i+1],kodatord[i+2],kodatord[i+3]]
        then begin grafkoll:= false; goto 999; end;
     grafkoll:= true;
     999:
end (* grafkoll *);
```

Figur 3: *Funktion som kontrollerar om alla fyrbokstavsföljder i ordet* `ordet` *finns i den grafotaktiska tabellen* `gtabell`.

finns med i ordlistan ska ges som rättstavningsförslag att bli nästan 4 procent, vilket är oacceptabelt högt. Vi bringar ned denna sannolikhet genom att för varje genererat ordförslag kolla att det bara innehåller bokstavskombinationer som finns i ordlistan, se nästa avsnitt.

5 Grafotaktisk tabell

En grafotaktisk tabell är en tabell som anger vilka bokstavsföljder som är rimliga i ett språk. Att använda en grafotaktisk tabell är ett alternativt sätt att kolla stavningen av ett ord.

Mullin och Margoliash [6] föreslår att man använder en tabell som för varje följd av tre bokstäver anger om den förekommer i något ord i ordlistan. Vi har funnit att en tabell över fyrbokstavsföljder fungerar bättre utan att ta för stor plats om den lagras på ett genomtänkt sätt. Förutom de 29 bokstäverna har vi tagit med ett specialtecken "`-`" som står för såväl ordbörjan, ordslut som bindestreck. Exempelvis ger ordet *g-klav* fyrbokstavsföljderna `-g-k`, `g-kl`, `-kla`, `klav` och `lav-`. Tabellen lagras som ett bitfält av storleken $30 \times 30 \times 30 \times 30$, d v s 810 000 bitar eller 101 250 byte.

```
var gtabell: array [1..30,1..30,1..30,1..30] of boolean;
```

Den grafotaktiska tabellen initieras med de fyrbokstavsföljder som finns i ordlistan. Den måste uppdateras då nya ord läggs till ordlistan. En svensk ordlista med 200 000 ord visade sig ge en tabell fylld till endast 7 procent, se avsnitt 17.

Ett ord kontrolleras genom att man för varje fyrbokstavsföljd ser efter om följden finns med i den grafotaktiska tabellen, se figur 3. Vid rättstavning kontrolleras alla genererade ordförslag dels med bloomfilter och dels med den grafotaktiska tabellen. För att få en extra noggrann kontroll av stavningen i en text kan man också använda båda metoderna, men normalt ger enbart bloomfiltret tillräckligt bra resultat.

Vid rättstavning kan en inledande kontroll av det felstavade ordet med den grafotaktiska tabellen minska arbetet med att generera rättstavningsförslag avsevärt. Om den grafotaktiska

analysen säger att den första fyrbokstavsföljden i ordet som inte finns i tabellen börjar på position p vet vi nämligen att felstavningen måste finnas mellan position p och $p+3$ i ordet.

1. Omkastning kan göras på 4 sätt.
2. En bokstav kan tas bort på 4 sätt.
3. En bokstav kan läggas till på $a \cdot 3$ sätt.
4. En bokstav kan ersättas på $(a-1) \cdot 4$ sätt.

Detta ger $7a + 4$ genererade ord, oberoende av ordets längd, eller 207 ord om $a = 29$, vilket ska jämföras med $59b + 28$ ord om vi inte vet var felet ligger.

I tidigare metoder för automatisk generering av rättstavningsförslag, se t ex Takahashi m fl [8], har det av tidsskäl ansetts orimligt att använda en ordlista som är större än ca 10 000 ord. Med våra metoder kan man med bibehållen snabbhet utöka ordlistans storlek praktiskt taget obegränsat.

En variant av den grafotaktiska metoden i kombination med associativa nät har utvecklats av Ekeberg [3]. Metoden ger sämre resultat för fel inom avstånd ett, men kan tänkas vara användbar när det felstavade ordet ligger på större avstånd från det rättstavade.

6 Sannolikhet för fel

Låt oss beräkna sannolikheten för att ett ord som inte finns med i ordlistan ska godkännas av bloomfiltret. Anta att ordlistan innehåller n ord, att tabellen rymmer m bitar och att vi använder k olika, oberoende och jämnt fördelande transformationsfunktioner. Vi ska alltså beräkna sannolikheten för att k stycken på varandra följande uppslagningar i tabellen alla ska ge svaret 1.

I tabellen har n ord lagrats och för varje ord har k bitar satts till 1. Sannolikheten för att en viss bit i tabellen därefter fortfarande är noll är

$$\left(1 - \frac{1}{m}\right)^{k \cdot n}$$

om vi antar att de kn tabelltilldelningarna är oberoende av varandra. Sannolikheten för att k slumpvis valda bitar i tabellen samtidigt ska vara ettställda blir då

$$f(k) = \left[1 - \left(1 - \frac{1}{m}\right)^{k \cdot n}\right]^k.$$

För att minimera sannolikheten sätter vi derivatan till noll:

$$f'(k) = 0 \Rightarrow \left(1 - \frac{1}{m}\right)^{k \cdot n} = \frac{1}{2}$$

vilket innebär att den optimalt utnyttjade tabellen är till hälften fylld med ettor. Om vi löser ut antalet funktioner k får vi

$$k = -\frac{\ln 2}{n \cdot \ln\left(1 - \frac{1}{m}\right)} \approx \ln 2 \cdot \frac{m}{n} \approx 0,69 \cdot \frac{m}{n}$$

och den minimala sannolikheten för fel blir

$$f(k) = 2^{-k}.$$

Exempel 1 Om ordlistan innehåller $n = 200\,000$ ord och vi väljer storleken på tabellen $m = 4\,000\,000$ bitar (0,5 Mbyte) så bör vi välja

$$k = \ln 2 \cdot \frac{4\,000\,000}{200\,000} \approx 13,9 \approx 14$$

d v s vi ska använda 14 transformationsfunktioner i bloomfiltret. Sannolikheten för fel med $k = 14$ är $f(14) \approx 6,7 \cdot 10^{-5} = 0,0067\%$. □

7 Försök att återskapa ordlistan

Om man har tillgång till ett program för stavningskontroll men inte till dess ordlista kan man alltid försöka återskapa ordlistan genom att generera och stavningskontrollera alla tänkbara ord. De ord som godkänns av programmet måste ingå i ordlistan om programmet alltid svarar rätt. Detta kan tyckas vara ogörligt eftersom det t ex finns över 17 miljarder kombinationer av högst sju svenska bokstäver, men med en rekursiv algoritm går det att bara konstruera de ord som är grafotaktiskt rimliga, d v s där alla fyrbokstavsföljder finns med i den grafotaktiska tabellen, och därmed kommer man ner i 15 miljoner konstruerade ord, dvs bara en tusendel. Det går därför med en rimlig datorinsats att rekonstruera ordlistan hos varje stavningsprogram som alltid svarar rätt.

Hur blir då resultatet för vår probabilistiska algoritm? Jo, av 15 miljoner konstruerade ord godkänns 43 tusen, varav knappt ett tusen är ord som inte finns. Detta är precis vad man kan förvänta sig, ty ett tusen är $0,0067\%$ av 15 miljoner. Vi hittar alltså dels alla 42 tusen ord i ordlistan som är högst sju bokstäver men får ett tusen felaktiga ord på köpet. Detta gör att resultatet är oanvändbart som ordlista. Tydligen är en probabilistisk algoritm det enda säkra skyddet av originalordlistan.

8 Nackdelar med för stor ordlista

Det kan tyckas vara bra om stavningsalgoritmen bygger på en mycket stor ordlista. Vissa stavningsprogram (t ex CAPS [4]) har som filosofi att alla ord i svenska språket ska finnas med i ordlistan så att programmet bara ska klaga på felstavade ord. Men det är faktiskt inte odelat positivt med en stor ordlista. Den främsta anledningen till detta är att ett felstavat ord kan godkännas på grund av att det råkar vara ett ovanligt ord eller ett fackord. Risken för sådana fel är av naturliga skäl större för korta ord. Därför bör standardordlistan inte innehålla ovanliga eller fackspecifika korta ord (t ex med mindre än åtta bokstäver), medan det går bra att ha ovanliga långa ord med (om de inte råkar sammanfalla med en enkla felstavningar av vanliga ord). Vid stavningskontroll av text inom något visst fackområde kan man till standardordlistan lägga en extra ordlista med fackord, se nästa avsnitt.

En matematisk analys av varför det kan vara önskvärt att begränsa orddatabasens storlek ges av Peterson [7].

9 Tilläggsordlistor, dokumentordlistor

Istället för att ha en mycket stor grundordlista har vi i STAVA valt att ha ett antal tilläggsordlistor. Vi har ett antal små, specialiserade, tilläggsordlistor som vid behov kan läggas till grundordlistan. För närvarande har vi i STAVA endast tilläggsordlistor för datatermer, namn,

förkortningar samt styrkoder för TEX och LATEX (se avsnitt 16) men det är enkelt att själv skapa nya ordlistor.

En annan typ av ordlista är den som hör ihop med ett specifikt dokument. Denna typ av ordlista, som vi valt att kalla dokumentordlista, är tänkt att innehålla ord som av en eller annan anledning inte hör hemma i en egen ordlista men som ändå ska godkännas i precis det här sammanhanget. Det kan t ex röra sig om slangord, ovanliga namn eller utländska ord som normalt inte hör hemma i en svensk text.

10 Ändelser

Vårt sätt att ta hand om ändelser är verkligen ad hoc-mässigt. Normalt så godkänns enbart ord som finns med i ordlistan men om så önskas kan ord som inte ingår i ordlistan, men som kan skapas genom att byta ut svansen på ett ord i ordlistan enligt en suffixtabell, godkännas. Det är naturligtvis så att felaktiga ord kan slinka igenom med denna enkla metod.

Exempel 2 *Låt oss anta att en del av suffixtabellen ser ut på följande sätt*

$$
\begin{array}{lcl}
-ad & \longrightarrow & -a \\
-ade & \longrightarrow & -a \\
-ades & \longrightarrow & -a \\
-aktig & \longrightarrow & \lambda \text{ (tomma strängen)} \\
-man & \longrightarrow & \lambda \\
-erat & \longrightarrow & -era \\
-na & \longrightarrow & -en \\
-nde & \longrightarrow & -nna
\end{array}
$$

och att ordet spatsera *ingår i ordlistan men däremot inte* spatserat*. Om vi valt att godkänna ord som kan skapas med hjälp av suffixtabellen kommer* spatserat *att godkännas eftersom* spatserat − erat + era *finns med i ordlistan.* □

För att stavningskontrollen ska vara extra säker bör böjningsformer av alla ord ingå i ordlistan i stället. I STAVA innehåller ordlistan de vanligaste böjningsformerna, men man kan välja att använda ovanstående metod för att säkert godkänna alla böjningsformer av ord i ordlistan.

11 Sammansättningar, bindebokstäver

Svenska språket skiljer sig i ett för stavningskontroll mycket viktigt avseende från t ex engelskan, nämligen hur nya ord kan bildas genom sammansättningar. Det finns i svenskan ingen gräns för hur många ord man kan bilda genom att sätta ihop flera ord. Det finns ingen möjlighet att göra en ordlista med alla dessa sammansättningar eller att kunna förutse vilka sammansättningar som kan komma att användas. De enda lösningarna tycks vara att antingen klaga på alla sammansatta ord som inte finns i ordlistan (där alla vanliga sammansättningar kan finnas, t ex alla sammansättningar som finns i SAOL) eller att godkänna alla sammansättningar av ord i ordlistan.

I programmet STAVA har vi valt den första lösningen som standardalternativ, men man kan enkelt välja den andra lösningen istället för att slippa få så många klagomål på riktiga

ord. Om ett ord inte finns i ordlistan och inte är en böjningsform av ett ord i ordlistan (se förra avsnittet) så antar vi att det är ett sammansatt ord och letar upp sammansättningspunkterna. Om man inte begränsar sammansättningsmöjligheterna på något sätt visar det sig att felstavade ord stundom anses vara riktiga eftersom de kan bildas genom sammansättning av många korta ord i ordlistan. Det verkar vara naturligt att begränsa antalet sammansättningsled till tre eller fyra och att begränsa minsta tillåta ordlängd i sammansättningslederna till två eller tre bokstäver beroende på hur många ord som finns i ordlistan.

Ett problem som uppstår vid vissa sammansättningar är tillägg av bindebokstäver och vokalbyten, t ex *tid+nöd=tidsnöd*, *tid+varv=tidevarv*, *binda+bokstäver=bindebokstäver*, *stuga+gemak=stugugemak*, *äga+del=ägodel*, *äta+pinne=ätpinne*. Binde-s är så vanligt att det bör godkännas vid alla sammansättningspunkter där föregående ordled inte slutar på vokal, *s*, *x* eller *z*. De övriga bindebokstäverna och vokalbytena är svårare att behandla korrekt och måste specialbehandlas. Vissa regler kan ställas upp, t ex *-ie* → *-ium* i sammansättningar som *mediearkiv*.

Om första leden i ett sammansatt ord slutar med dubbelskriven konsonant och nästa ord börjar med samma konsonant skrivs i svenskan bara två konsonanter. Vi kan komplicera vår algoritm ytterligare genom att ta hänsyn även till detta och därför testa att sätta in ytterligare en konsonant vid dubbelskrivna konsonanter.

12 Ändelser vid sammansättningar, prefix

För att även böjningsformer av sammansatta ord ska godkännas måste vi samtidigt tillämpa idéerna i avsnitt 10 och 11. I slutet på ordet kan vi göra precis som vanligt när vi kontrollerar om ordet finns i grundform. Men i de tidigare sammansättningsleden ska bara vissa ändelser tillåtas, t ex *-ande*, *-ende*, *-else*, *-ing*, *-ning*, *-isering*, *-het*.

Det finns också prefix som inte kan stå som egna ord utan bara kan ingå i sammansättningar, t ex *auto-*, *hyper-*, *infra-*, *meta-*, *mikro-*, *pseudo-* och *semi-*. Dessa ord bör därför inte stoppas in i den ordinarie ordlistan. Istället kan de utgöra en egen specialordlista som bara används för att hitta förled i sammansättningar. Denna ordlista blir mycket liten (mindre än hundra ord) och kan därför med fördel lagras sorterad i bokstavsordning i en vektor i vilken man söker med binärsökning.

13 Grafotaktisk koll vid sammansättningar

Om grafotaktisk kontroll (se avsnitt 5) ska göras av bokstavsföljderna i ett sammansatt ord måste man tänka på att det i sammansättningspunkterna kan uppstå nya bokstavskombinationer som inte förekommer inuti ord. Därför bör den grafotaktiska kontrollen göras på varje ordled för sig och inte på hela det sammansatta ordet. Men en grafotaktisk kontroll av hela ordet kan utnyttjas för att hitta platser som måste vara sammansättningspunkter och därigenom minska det annars relativt krävande arbetet att söka igenom alla tänkbara sammansättningspunkter.

14 Stor bokstav

Att avgöra var det ska vara stor bokstav är ett problem som ligger nära problemet med stavningkontroll. I ordlistan kan utan problem ord lagras med angiven bokstavsstorlek — det

gäller bara att transformationsfunktionerna klarar av att transformera såväl stora som små bokstäver. Därmed klarar algoritmen av att godkänna *Sverige* men klaga på *sverige*. Problem uppstår dock när ett ord får stor begynnelsebokstav först i en mening eller skrivs helt med versaler i t ex en rubrik.

Antingen kan vi alltid godkänna att ett ord som ska stavas med små bokstäver skrivs med inledande stor bokstav eller helt med versaler, eller också försöker vi gissa när det börjar en ny mening eller rubrik. Exempelvis kan vi säga att det får förekomma stor begynnelsebokstav efter skiljetecknen . ! ? – " ' och först på raden.

15 Tecknen åäö

De svenska bokstäverna åäö ställer till med en del problem eftersom de i ISO-standarden för 7-bitarstecken krockar med tecknen }{| i den amerikanska teckenuppsättningen. Normalläget för STAVA är att använda den gamla vanliga sjubitskoden för svenska bokstäver. Om så önskas kan STAVA använda åttabitskoder för svenska bokstäver enligt ISO 8859-1. En svaghet är att accenter skalas bort så att t ex *idé* och *ide* behandlas som samma ord. Ännu värre är det naturligtvis att *succe* godkänns eftersom det inte finns något sådant svenskt ord (däremot finns ju *succé*). Förklaringen till detta beteende är att åttabitskoden ISO 8859-1, som man behöver för att kunna skriva alla bokstäver med accenter, inte har slagit igenom ännu. Det går lätt att modifiera alfabetet och lägga till t ex é och à som egna bokstäver. Därefter kan den vanliga algoritmen användas för att skilja *succé* från *succe*.

16 Text med styrkoder

Det finns åtminstone två olika sätt att behandla text som innehåller styrkoder[2]. En metod är att först skala bort alla styrkoder med hjälp av ett s k textfilter innan stavningkontrollen påbörjas. Ett annat alternativ, som vi med viss framgång använt för TEX och LATEX, är att skapa en tilläggsordlista med alla fördefinierade styrkoder. Detta fungerar bara fullt ut om inga egna namn definieras i TEX-filen.

17 Test av en implementation av metoden

För att ge ett hum om hur snabb vår implementation STAVA är har vi utfört några enkla tester på en Sparc Station ELC. Vi vill inte göra sken av att dessa tester är särskilt väl valda eller "vetenskapliga" — ta dem med en nypa salt.

Testfilen innehöll 14923 ord om totalt 120615 tecken (det blir ca 8 tecken per ord). Det vanligaste sättet att använda STAVA är att använda väljaren `-k` (`k` står för kort lista med fel; för en förklaring av väljarnas innebörd se listan nedan) som är en kombination av väljarna `-eafnsd`. Om rättstavade förslag till varje felstavat ord önskas går kontrollen naturligtvis mycket långsammare men å andra sidan är tiden för att presentera förslagen försumbar jämfört med den tid en användare måste lägga ned för att utföra en ändring i texten. Resultaten från testerna återfinns i figur 4.

[2]Styrkoder används ofta för att välja teckensnitt och för att tala om var på sidan texten ska placeras. Det kan också vara så att bilder och text blandas på en sida och att styrkoder talar om var en bild börjar och slutar.

väljare	real	user	system	antal fel	ord/s
ingen	6,0	5,6	0,2	620	2487
`-k`	6,5	6,2	0,2	356	2296
`-r`	23,8	23,2	0,3	620	627
`-kr`	14,3	13,5	0,3	356	1043

Figur 4: *Ett enkelt test av* STAVA. *Tiderna* real, user *och* system *är de som rapporteras av programmet* `/usr/bin/time` *då utmatningen sker till* `/dev/null` *och är angivna i sekunder.*

`e` rapportera felstavade ord bara en gång även om samma stavfel förekommer på flera ställen,

`a` godkänn ändelser,

`f` använd tilläggsordlista med förkortningar,

`n` använd tilläggsordlista med namn,

`s` godkänn ord som kan bildas genom sammansättningar av i ordlistan förekommande ord,

`d` använd tilläggsordlista med datatermer,

`r` ge rättstavade förslag till varje felstavat ord.

I de fall där det var 356 felstavade ord var totala antalet tecken 3077 (ca 8,6 tecken per ord) och för fallen med 620 felstavade ord var motsvarande siffra 6094 (ca 9,8 tecken per ord).

18 Användarvänligt gränssnitt

Förutom STAVA, som endast har ett textbaserat gränssnitt i god Unixtradition, finns en implementation, kallad XSTAVA under X-windows på SUN arbetsstationer samt en implementation kallad MACSTAVA för Macintosh. Både XSTAVA och MACSTAVA har utförts som projektarbeten av elever på KTHs datatekniklinje och har betydligt trevligare användargränssnitt än STAVA. MACSTAVA klarar av dokument från så gott som samtliga populära ordbehandlare. De filter för styrkoder som nämndes i avsnitt 16 ingår alltså som en del av MACSTAVA.

19 Användning i ordbehandlare

Förutom det normala förfarandet att först skriva in texten med en ordbehandlare och sedan kontrollera stavningen med hjälp av STAVA, XSTAVA eller MACSTAVA kan man på Unixdatorer använda sig av STAVA som en del av texteditorn GNU Emacs. Fördelen med detta är att delar av ett dokument kan kontrolleras utan att man behöver lämna Emacs vilket gör att kontrollen går fortare och smidigare. Det hela fungerar så att Emacs, med hjälp av ett litet program skrivet i *elisp*, anropar STAVA med den markerade texten som indata.

Litteraturförteckning

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. Commun. ACM, **13**:7, (1970), s 422–426.

[2] F. J. Damerau. A technique for computer detection and correction of spelling errors. Commun. ACM, **7**:3, (1964), s 171–176.

[3] Ö. Ekeberg. Robust dictionary lookup using associative networks. Int. J. Man–Machine Studies, **28**, (1988), s 29–43. Föreligger även som teknisk rapport vid NADA, KTH.

[4] E. Malm och K. Waller. Datorstödd stavningskontroll för svenska språket. Examensarbete vid institutionen för grafisk teknik, KTH, (1991).

[5] J. K. Mullin. A second look at Bloom filters. Commun. ACM, **26**:8, (1983), s 570–571.

[6] J. K. Mullin och D. J. Margoliash. A tale of three spelling checkers. Software–Practice and Experience, **20**:6, (1990), s 625–630.

[7] J. L. Peterson. A note on undetected typing errors. Commun. ACM, **29**:7, (1986), s 633–637.

[8] H. Takahashi, N. Itoh, T. Amano och A. Yamashita. A spelling correction method and its application to an OCR system. Pattern Recognition, **23**:3/4, (1990), s 363–377.

Bilaga A

Källkod

1 stava.h

```
/* stava.h r en inkuderingsfil till stava.c */
/*
  Copyright (C) 1990,1991,1992
  Joachim Hollman och Viggo Kann
  joachim@nada.kth.se viggo@nada.kth.se
 */

/* Datafilsnamn om inget annat anges */
#define LISTFIL        "/usr/local/lib/stava/list"
#define KLISTFIL       "/usr/local/lib/stava/klist"
#define XFORKORTNINGAR "/usr/local/lib/stava/forkortningar"
#define XNAMN          "/usr/local/lib/stava/namn"
#define XDATATERMER    "/usr/local/lib/stava/datatermer"
#define XTEX           "/usr/local/lib/stava/tex"
#define XFYRGRAFER     "/usr/local/lib/stava/fyrgraf"
#define DOKUMENTORDLISTEEFTERNAMN ".ord"
#define STORLEK 250000       /* Antal 16-bitarselement i bitvektorn */
#define FYRSTORLEK 50625     /* Antal 16-bitarselement i fyrgrafsbitvektorn */
#define LANGD 50             /* Lngsta tilltna ord */
#define DELORDMIN 3          /* Minsta tilltna ordlngd i sammansttningar */
#define ORDMIN 2             /* Ord som r kortare n ORDMIN hoppas ver */
#define VSHIFT 7             /* Hur mycket nsta pow2 r vnstershiftad */
#define PTAL 14              /* Antal hashningar */
#define ETT14 1<<14
long p[PTAL] = { /* De PTAL primtalen nrmast mindre n 16*STORLEK */
  3999971,
  3999949,
  3999929,
  3999923,
  3999917,
  3999901,
  3999893,
  3999871,
  3999859,
  3999851,
  3999803,
  3999791,
  3999781,
  3999773
};
```

2 stava.c

```
/* Rttstavningsprogram. Version 1.5  1992-05-14
   Copyright (C) 1990, 1991, 1992
   Joachim Hollman och Viggo Kann
   joachim@nada.kth.se viggo@nada.kth.se
 */

/* #define ISO8BITAR */
/* Definiera ISO8BITAR om ttabitarstecken ISO8859-1 ska anvndas som
   standard, dvs utan att -8 behver ges. -7 ger d sjubitarstecken.
   I infilerna kan alltid bde 7- och 8-bitarsbokstver anvndas. */

#include <stdio.h>

#ifdef __GNUC__
#define INLINE inline
#define VOLATILE volatile
#else
#define INLINE
#define VOLATILE
#endif

#include "stava.h"
#define liten(t)       (97<=t && t<=125)
#define stor(t)        (65<=t && t<=93)
#define Liten(t)       (t+32) /* Krver att t r en stor bokstav */
#define KOLLAORD(o) { if (ILexikon(o)) return(1); else\
                      if (Finns2(o)) return(1); }
#define KOLLAORDPETIG(o) { if (ILexikon(o)) return(1); else\
                           if (Finns2Petig(o,0)) return(1); }

unsigned short tabell[STORLEK];
char lexfNamn[100], ordfNamn[100], xlexfNamn[100], utlexfNamn[100];
FILE *lexf, *ordf, *xlexf;
long pow2[PTAL][LANGD];
int filter = 0;
int bindestreck;
int xLasOrdlista = 1, xInforOrdlista = 0;
int xAndelser = 0, xForkortningar = 0, xNamn = 0, xDatatermer = 0;
int xTex = 0, xSammansatta = 0, xKort = 0, xEndastEtt = 0, xSkrivLexikon = 0;
int xIntePetig = 1;
int laserOrdlista = 0, inforOrdlista = 0;
#ifdef ISO8BITAR
int x8bitar = 1;
#else
int x8bitar = 0;
#endif
char **infiler = NULL;   /* vektor med pekare till infilnamn */
int antalInfiler = 0;    /* sista anvnda texten i infiler */
int fleraInfiler = 0;    /* sant om antalInfiler > 1 */
int aktuelltFilnr = 0;   /* vilken av filerna som stavningskollas fn */
char **ordlistor = NULL; /* vektor med pekare till ordlistenamn */
int antalOrdlistor = 0;  /* sista anvnda texten i ordlistor */
char *bokstavsTabell;

char ISO_ASCII[256] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,'-',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
'P','Q','R','S','T','U','V','W','X','Y','Z',0,0,0,0,0,
0,'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o',
'p','q','r','s','t','u','v','w','x','y','z',0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
'A','A','A','A',']',']','[','C','E','E','E','E','I','I','I','I',
'D','N','O','O','O','O','\\',0,'\\','U','U','U','Y','Y',0,0,
'a','a','a','a','{','}','{','c','e','e','e','e','i','i','i','i',
'd','n','o','o','o','o','|',0,'|','u','u','u','y','y',0,'y'};

unsigned char ASCII_ISO[128] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,0,0,0,0,0,'-',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
'P','Q','R','S','T','U','V','W','X','Y','Z',
0xc4 /*   */, 0xd6 /*   */, 0xc5 /*   */, 0,0,
0,'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o',
'p','q','r','s','t','u','v','w','x','y','z',
0xe4 /*   */, 0xf6 /*   */, 0xe5 /*   */, 0,0};

char ASCII_ASCII[256] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,'-',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
'P','Q','R','S','T','U','V','W','X','Y','Z','[','\\',']',0,0,
0,'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o',
'p','q','r','s','t','u','v','w','x','y','z','{','|','}',0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

char dubbelBokstavsTabell[256] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,'-',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
'P','Q','R','S','T','U','V','W','X','Y','Z','[','\\',']',0,0,
0,'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o',
'p','q','r','s','t','u','v','w','x','y','z','{','|','}',0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
'A','A','A','A','[',']','[','C','E','E','E','E','I','I','I','I',
'D','N','O','O','O','O','\\',0,'\\','U','U','U','Y','Y',0,0,
'a','a','a','a','{','}','{','c','e','e','e','e','i','i','i','i',
'd','n','o','o','o','o','|',0,'|','u','u','u','y','y',0,'y'};

char bindebokstav[128] = {
/* 's' str fr bokstver som i sammansttningar kan f s efter sig */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,' ','s','s','s',' ','s','s','s',' ','s','s','s','s','s',' ',
's','s','s',' ','s',' ','s','s',' ',' ',' ',' ',' ',' ',0,0,
0,' ','s','s','s',' ','s','s','s',' ','s','s','s','s','s',' ',
's','s','s',' ','s',' ','s','s',' ',' ',' ',' ',' ',' ',0,0};

INLINE int ILexikon(char *ord)
{ register i,j;
  long h;

  for (i = 0; i < PTAL; i++) {
    j = 0; h = 0;
    while (ord[j] != '\0') {
      h = (h + ord[j] * pow2[i][j]) % p[i];
      j++;
    }
    if (!(tabell[h>>4] & (1<<((int) h & 017)))) return(0);
  }
  return(1);
}

/* LagraOrd hashar in ordet i lexikonet */
INLINE void LagraOrd(char *ord)
{ register int i, j;
  register long h;

  for (i = 0; i < PTAL; i++) {
    j = 0; h = 0;
    while (ord[j] != '\0' && j < LANGD) {
      h = (h + ord[j] * pow2[i][j]) % p[i];
      j++;
    }
    tabell[h>>4] |= 1<<((int) h & 017);
  }
}
```

```
int xRattstavningsforslag = 0;
unsigned short fyrtabell[FYRSTORLEK];
FILE *fyrf;
char **fyrOrd;
int fyrAntalOrd = 0, fyrMaxAntalOrd = 0;

/* FyrKollaHela kollar om ett ords alla fyrgrafer r tilltna */
INLINE int FyrKollaHela(char *ord)
{ static char buf[LANGD+4];
  char *s;
  long l;
  int plats;
  sprintf(buf, "-%s-", ord);
  for (s = buf; *s; s++)
    if (*s >= 'a' && *s <= '}') *s -= 'a'; else
    if (*s >= 'A' && *s <= ']') *s -= 'A'; else
      *s = 29;
  for (plats = 3; buf[plats]; plats++) {
    l = ((buf[plats - 3] * 30 + buf[plats - 2]) * 30 + buf[plats - 1]) * 30 +
        buf[plats];
    if (!(fyrtabell[l >> 4] & (1 << ((int) l & 017)))) return(plats);
  }
  return(0);
}

/* LagraFyrgrafer ser till att ett ords alla fyrgrafer r tilltna */
INLINE void LagraFyrgrafer(char *ord)
{ static char buf[LANGD+4];
  char *s;
  long l;
  int plats;
  sprintf(buf, "-%s-", ord);
  for (s = buf; *s; s++)
    if (*s >= 'a' && *s <= '}') *s -= 'a'; else
    if (*s >= 'A' && *s <= ']') *s -= 'A'; else
      *s = 29;
  for (plats = 3; buf[plats]; plats++) {
    l = ((buf[plats - 3] * 30 + buf[plats - 2]) * 30 + buf[plats - 1]) * 30 +
        buf[plats];
    fyrtabell[l >> 4] |= (1 << ((int) l & 017));
  }
}

/* FyrKoll kollar om ett ords fyrgrafer r tilltna, plats anger index i
ord fr en ndring. Om plats r negativt kollas hela ordet.
extra r antalet extra positioner som r ndrade */
int FyrKoll(char *ord, int plats, int extra)
{ char buf[LANGD+4];
  char *s;
  long l;
  sprintf(buf, "-%s-", ord);
  for (s = buf; *s; s++)
    if (*s >= 'a' && *s <= '}') *s -= 'a'; else
    if (*s >= 'A' && *s <= ']') *s -= 'A'; else
      *s = 29;
  *s = '/';
  if (plats >= 2) {
    plats++;
    buf[plats + 4 + extra] = '/';
  } else {
    if (plats >= 0) buf[plats + 5 + extra] = '/';
    plats = 3;
  }
  while (buf[plats] != '/') {
    l = ((buf[plats - 3] * 30 + buf[plats - 2]) * 30 + buf[plats - 1]) * 30 +
        buf[plats];
    if (!(fyrtabell[l >> 4] & (1 << ((int) l & 017)))) return(0);
    plats++;
  }
  return(1);
}
```

```
void FyrAdderaOrd(char *ord)
{ int i;
  if (strlen(ord) <= 1) return; /* Strunta i enbokstavsord */
  if (fyrMaxAntalOrd == 0) {
    fyrMaxAntalOrd = 20;
    fyrOrd = (char **) malloc(sizeof(char *) * fyrMaxAntalOrd);
  } else {
    for (i = 0; i < fyrAntalOrd; i++) {
      if (!strcmp(ord, fyrOrd[i])) return;
      if (!strcmp(ord + 1, fyrOrd[i] + 1))
        if (abs(*ord - *fyrOrd[i]) == 'a' - 'A') return;
    }
    if (fyrAntalOrd >= fyrMaxAntalOrd) {
      fyrMaxAntalOrd += 20;
      fyrOrd = (char **) realloc(fyrOrd, sizeof(char *) * fyrMaxAntalOrd);
    }
  }
  fyrOrd[fyrAntalOrd] = (char *) malloc(strlen(ord)+1);
  strcpy(fyrOrd[fyrAntalOrd], ord);
  fyrAntalOrd++;
}

void FyrSudda(void)
{ int i;
  for (i = 0; i < fyrAntalOrd; i++) free(fyrOrd[i]);
  fyrAntalOrd = 0;
}

/* S_byt byter plats p tv bokstver i taget, dr den frsta bokstaven har
   index mellan fran och till i ord */
void S_byt(char *ord, int len, int fran, int till)
{ int i;
  char t;
  if (till >= len - 1) till = len - 2;
  if (fran < 0) fran = 0;
  for (i = fran; i <= till; i++) {
    t = ord[i]; ord[i] = ord[i+1]; ord[i+1] = t;
    if (FyrKoll(ord, i, 1)) if (ILexikon(ord)) FyrAdderaOrd(ord);
    t = ord[i]; ord[i] = ord[i+1]; ord[i+1] = t;
  }
}

/* S_in skjuter in ett nytt tecken i en position i taget mellan fran och till
   i ord */
void S_in(char *ord, int len, int fran, int till)
{ int i;
  char t, buf[LANGD+1];
  if (till > len) till = len;
  if (fran < 0) fran = 0;
  if (fran == 0) {
    strcpy(buf + 1, ord);
    for (t = 'A'; t <= ']'; t++) {
      buf[0] = t;
      if (FyrKoll(buf, 0, 0)) if (ILexikon(buf)) FyrAdderaOrd(buf);
    }
  } else strcpy(buf, ord);
  for (i = fran; i <= till; i++) {
    strcpy(buf + i + 1, ord + i);
    for (t = 'a'; t <= '}'; t++) {
      buf[i] = t;
      if (FyrKoll(buf, i, 0)) if (ILexikon(buf)) FyrAdderaOrd(buf);
    }
    buf[i] = ord[i];
  }
}

/* S_bort tar bort ett tecken i en position i taget mellan fran och till
i ord. Drefter provas alla mjliga tecken som utbyte. */
void S_bort(char *ord, int len, int fran, int till)
{ int i;
  char buf[LANGD];
```

```
  if (fran < 0) fran = 0;
  if (till >= len) till = len - 1;
  if (fran > 0) strncpy(buf, ord, fran);
  for (i = fran; i <= till; i++) {
    strcpy(buf + i, ord + i + 1);
    if (FyrKoll(buf, i, -1)) if (ILexikon(buf)) FyrAdderaOrd(buf);
    S_in(buf, len-1, i, i);
    buf[i] = ord[i];
  }
}

void GenereraAlternativaOrd(char *ord)
{ int felplats, len = strlen(ord);
  int fyrresultat[LANGD+1], vanstrastefel, hograstefel, diff;
  char buf[LANGD+2], *s;
  long l;

  sprintf(buf, "-%s-", ord);
  for (s = buf; *s; s++)
    if (*s >= 'a' && *s <= '}') *s -= 'a'; else
    if (*s >= 'A' && *s <= ']') *s -= 'A'; else
      *s = 29;
  *s = '/';
  hograstefel = -1;
  for (felplats = len - 2; felplats >= 0; felplats--) {
    l = ((buf[felplats] * 30 + buf[felplats + 1]) * 30 + buf[felplats + 2]) *
        30 + buf[felplats + 3];
    fyrresultat[felplats] = !(fyrtabell[l >> 4] & (1 << ((int) l & 017)));
    if (hograstefel < 0) if (fyrresultat[felplats]) hograstefel = felplats;
  }
  if (hograstefel >= 0) {
    for (felplats = 0; felplats <= hograstefel; felplats++)
      if (fyrresultat[felplats]) {
        vanstrastefel = felplats;
        break;
      }
    diff = hograstefel - vanstrastefel;
    if (diff <= 4) {
      S_byt(ord, len, hograstefel - 2, vanstrastefel + 2);
      if (diff <= 3) {
        S_bort(ord, len, hograstefel - 1, vanstrastefel + 2);
        if (diff <= 2) S_in(ord, len, hograstefel, vanstrastefel + 2);
      }
    }
  } else {
    S_byt(ord, len, 0, len - 2);
    S_bort(ord, len, 0, len - 1);
    S_in(ord, len, 0, len);
  }
}

void InitieraBokstavsTabeller(void)
{
  if (x8bitar) {
    bokstavsTabell = ISO_ASCII;
    if (xTex) bokstavsTabell['\\'] = '/';
  } else {
    bokstavsTabell = ASCII_ASCII;
    if (xTex) bokstavsTabell['/'] = '/';
  }
}

INLINE void SkrivOrd(char *s)
{
  if (x8bitar) {
    while (*s) putchar(ASCII_ISO[*s++]);
  } else printf("%s", s);
}

INLINE int SkippaRestenAvOrdet(void)
{ int t;
  while ((t = getc(ordf)) != EOF) {
```

```
    if (bokstavsTabell[t]) continue;
    if (t == '-') continue;
    if (xTex && t == '/') {
      t = getc(ordf);
      if (t == '\'' || t == '`' || t == '^' || t == '"' || t == '=' ||
          t == '.' || t == '~') continue;
    }
    return(1);
  }
  return(0);
}

int TagOrd(char *s, char *urord)
{ int t, u, i, nyrad;
 start:
  do {
    i = 0;
    bindestreck = 0;
    do {
      u = getc(ordf);
      if (u == EOF) return(0);
    } while (!(t = bokstavsTabell[u]) || t == '-');

    while (1) {
      if (t == '-' || (xTex && t == '/')) {
        /* I Tex behandlas /- precis som bara - */
        if (t == '/') if ((u = getc(ordf)) != '-') {
          if (u == '\'' || u == '`' || u == '^' || u == '~') {
            if ((u = getc(ordf)) != EOF)
              if ((t = bokstavsTabell[u])) continue;
          }
          ungetc(u, ordf);
          goto ordSlut;
        }
        u = getc(ordf);
        if (u == ' ' || u == '\t' || u == '\n') {
          nyrad = (u == '\n');
          while (1) {
            if ((u = getc(ordf)) == EOF) goto ordSlut;
            if (u == '\n') nyrad = 1;
            else if (u != ' ' && u != '\t') break;
          }
          if (!nyrad) {
            ungetc(u, ordf);
            goto ordSlut;
          }
        }
        if ((t = bokstavsTabell[u])) {
          s[i] = urord[i] = '-';
          if (++i >= LANGD) {
            if (!SkippaRestenAvOrdet()) return(0);
            goto start;
          }
          bindestreck++;
        } else goto ordSlut;
      }
      s[i] = t;
      urord[i] = u;
      if (++i >= LANGD) {
        if (!SkippaRestenAvOrdet()) return(0);
        goto start;
      }
      if ((u = getc(ordf)) == EOF) break;
      if (!(t = bokstavsTabell[u])) break;
    }
   ordSlut: ;
  } while (i < ORDMIN);
  s[i] = urord[i] = '\0';
  return(1);
}
```

```
INLINE void SuddaBindestreck(char *ordin, char *ordut)
{
  if (xSammansatta) if (ordin[1] == '-')
    ordin += 2; /* Gr om t ex c-uppgiften till uppgiften */
  while (*ordin)
    if (*ordin == '-') ordin++;
    else *ordut++ = *ordin++;
  *ordut = '\0';
}

void Initiera(void) {
  int i,j;
  long h;
  for (i = 0; i < PTAL; i++) {
    h = pow2[i][0] = ETT14;
    for (j = 1; j < LANGD; j++) {
      h = (h << VSHIFT)%p[i];
      pow2[i][j] = h;
    }
  }
  InitieraBokstavsTabeller();
  if (xAndelser) InitSuf();
}

/* Prefix kollar om argumentet r ett tilltet frled i sammansttningar */
INLINE int Prefix(char *ord)
{
  return(!strcmp(ord, "auto")); /* Fyll p med flera prefix hr */
}

INLINE int Finns(char *ord)
{ int i;
  if (!xIntePetig) {
    i = FyrKollaHela(ord);
    if (i > 0) {
      if (xSammansatta) return(Finns2Petig(ord, i));
      if (xAndelser) {
        char ord2[LANGD];
        if (Suffix(ord,ord2)) {
          if (strlen(ord2) < ORDMIN) return(0);
          if (xSammansatta) if (Finns2Petig(ord2, i)) return(1);
        }
      }
      return(0);
    }
  }
  if (ILexikon(ord)) return(1);
  if (xAndelser) {
    char ord2[LANGD];
    if (Suffix(ord,ord2)) {
      if (strlen(ord2) < ORDMIN) return(0);
      if (ILexikon(ord2)) return(1);
      if (xSammansatta) if (Finns2(ord2)) return(1);
    }
  }
  if (xSammansatta) if (Finns2(ord)) return(1);
  return(0);
}

/* Finns2 kollar om ett ord r sammansatt */
int Finns2(char *ord)
{ char ord1[LANGD], ord2[LANGD];
  int i, j, l = strlen(ord);
  i = l / 2;
  j = l - i;
  if (i < DELORDMIN) return(0); /* Ordet mste vara minst 2*DELORDMIN */
                        /* bokstver fr att kunna vara en sammansttning */
  strncpy(ord1, ord, i);
  if (i == j) { /* jmnt antal bokstver - kolla delning i mitten */
    ord1[i] = '\0';
    if (ILexikon(ord1)) {
```

```
      /* Hr kan man kolla suffix */
      KOLLAORD(ord + i)
      if (ord[i] == 's' && i > DELORDMIN && bindebokstav[ord[i-1]] == 's')
        KOLLAORD(ord + i + 1)
    } else if (Prefix(ord1)) { KOLLAORD(ord + i) }
    i--;
    j++;
  }
  strncpy(ord2, ord, j);
  for (; i >= DELORDMIN; i--, ord2[j] = ord[j], j++) {
    ord1[i] = '\0';
    ord2[j] = '\0';
    if (ILexikon(ord1)) {
      /* Hr kan man kolla suffix */
      KOLLAORD(ord + i)
      if (ord[i] == 's' && bindebokstav[ord[i-1]] == 's')
        KOLLAORD(ord + i + 1)
    } else if (Prefix(ord1)) { KOLLAORD(ord + i) }
    if (ILexikon(ord2)) {
      /* Hr kan man kolla suffix */
      KOLLAORD(ord + j)
      if (ord[j] == 's' && i > DELORDMIN && bindebokstav[ord[j-1]] == 's')
        KOLLAORD(ord + j + 1)
    } else if (Prefix(ord2)) { KOLLAORD(ord + j) }
  }
  return(0);
}

/* Finns2Petig kollar om ett ord r sammansatt och har tilltna fyrgrafer */
int Finns2Petig(char *ord, int fyrKoll)
{ char ord1[LANGD];
  int i, l = strlen(ord);
  if (fyrKoll > 0) i = fyrKoll; else i = FyrKollaHela(ord);
  if (i == 0) return(Finns2(ord)); /* Bara tilltna fyrgrafer */
  i--;
  if (i > l - DELORDMIN) i = l - DELORDMIN;
  strncpy(ord1, ord, i);
  for (; i >= DELORDMIN; i--) {
    ord1[i] = '\0';
    if (ILexikon(ord1)) {
      /* Hr kan man kolla suffix */
      KOLLAORDPETIG(ord + i)
      if (ord[i] == 's' && bindebokstav[ord[i-1]] == 's')
        KOLLAORDPETIG(ord + i + 1)
    } else if (Prefix(ord1)) { KOLLAORD(ord + i) }
  }
  return(0);
}

int OppnaNastaInfil(void)
{
  while (aktuelltFilnr < antalInfiler) {
    strcpy(ordfNamn, infiler[aktuelltFilnr++]);
    if (!(ordf = fopen(ordfNamn,"r"))) {
      fprintf(stderr,"Kan inte ppna filen %s\n", ordfNamn);
      continue;
    }
    return(1);
  }
  return(0);
}

void OppnaFiler(void)
{
  if (antalInfiler == 0) {
    filter = 1;
    ordf = stdin;
    strcpy(ordfNamn, "stdio");
  } else {
    if (antalInfiler > 1) fleraInfiler = 1;
```

```
    if (!OppnaNastaInfil()) exit(1);
  }
  /* ppna lexikonfilen: */
  if (!(lexf = fopen(lexfNamn,"r"))) {
    fprintf(stderr,"Kan inte ppna filen %s\n", lexfNamn);
    exit(1);
  }
  if (xRattstavningsforslag || !xIntePetig)
    if (!(fyrf = fopen(XFYRGRAFER, "r"))) {
      fprintf(stderr,"Kan inte ppna filen %s\n", XFYRGRAFER);
      exit(1);
    }
}

void OppnaUtfil(void)
{
  if (!laserOrdlista) {
    strcpy(xlexfNamn, ordfNamn);
    strcat(xlexfNamn, DOKUMENTORDLISTEEFTERNAMN);
  }
  if ((xlexf = fopen(xlexfNamn,"a"))) inforOrdlista = 1;
  else {
    fprintf(stderr, "Kan inte skriva p filen %s\n", xlexfNamn);
    inforOrdlista = 0;
  }
}

void StangFiler(void)
{
  if (antalInfiler > 0) fclose(ordf);
  if (inforOrdlista) fclose(xlexf);
}

long TagExtraOrdlista(FILE *xlexf)
{ long antal = 0;
  char *s, buf[LANGD];
  while (fgets(buf, LANGD-1, xlexf)) {
    s = buf;
    while ((*s = dubbelBokstavsTabell[(unsigned char) (*s)])) s++;
    if (*buf) {
      LagraOrd(buf);
      if (xRattstavningsforslag || !xIntePetig) LagraFyrgrafer(buf);
      antal++;
    }
  }
  fclose(xlexf);
  return(antal);
}

void TagStandardOrdlista(char *namn)
{
  if ((xlexf = fopen(namn, "r"))) TagExtraOrdlista(xlexf);
}

void TagLexikon(void)
{ int i;
  long antal = 0;
  char slask;

  if (fread(tabell, sizeof(unsigned short), STORLEK, lexf) != STORLEK
      || fread(&slask, sizeof(char), 1, lexf) == 1) {
    fprintf(stderr, "%s har fel format fr att vara en lexikonfil\n",
            lexfNamn);
    fclose(lexf);
    exit(1);
  }
  fclose(lexf);
  if (xRattstavningsforslag || !xIntePetig) {
    if (fread(fyrtabell, sizeof(unsigned short), FYRSTORLEK, fyrf) !=
        FYRSTORLEK || fread(&slask, sizeof(char), 1, fyrf) == 1) {
      fprintf(stderr, "%s har fel format fr att vara en fyrgraffil\n",
              XFYRGRAFER);
```

```
      fclose(fyrf);
      exit(1);
    }
    fclose(fyrf);
  }
  for (i = 0; i < antalOrdlistor; i++) {
    if (!(xlexf = fopen(ordlistor[i], "r")))
      fprintf(stderr, "Kan inte lsa ordlistan %s\n", ordlistor[i]);
    else antal += TagExtraOrdlista(xlexf);
  }
  if (!xKort) { /* KLISTFIL innehller redan fljande tre ordlistor */
    if (xForkortningar) TagStandardOrdlista(XFORKORTNINGAR);
    if (xNamn) TagStandardOrdlista(XNAMN);
    if (xDatatermer) TagStandardOrdlista(XDATATERMER);
  }
  if (xTex) TagStandardOrdlista(XTEX);
}

void TagNastaExtraOrdlista(void)
{
  strcpy(xlexfNamn, ordfNamn);
  strcat(xlexfNamn, DOKUMENTORDLISTEEFTERNAMN);
  if ((xlexf = fopen(xlexfNamn, "r"))) {
    laserOrdlista = 1;
    TagExtraOrdlista(xlexf);
  } else laserOrdlista = 0;
}

void SkrivLexikon(void)
{
  if (!(lexf = fopen(utlexfNamn,"w"))) {
    fprintf(stderr,"Kan inte ppna filen %s\n", utlexfNamn);
    exit(1);
  }
  fwrite(tabell, sizeof(unsigned short), STORLEK, lexf);
  fclose(lexf);
}

INLINE void VersalerGemena(register char *ordin, register char *ord,
                           register char *Ord)
{ register int i;
  int baraGemena, baraVersaler;
  baraGemena = baraVersaler = 1;
  for (i = 0; ordin[i] != '\0'; i++) {
    if (stor(ordin[i])) {
      baraGemena = 0;
      ord[i] = Liten(ordin[i]);
    } else {
      baraVersaler = 0;
      ord[i] = ordin[i];
    }
  }
  if (baraGemena) ord[0] = Ord[0] = '\0';
  else {
    ord[i] = '\0';
    if (baraVersaler) {
      Ord[0] = ordin[0];
      strcpy(Ord + 1, ord + 1);
    } else Ord[0] = '\0';
  }
}

/* EnvArg hmtar en environmentvariabel och bryter upp den i argv och argc */
int EnvArg(char *namn, int *argc, char ***argv)
{ char *s, *start, *env, **v;
  int antal = 0, len, i;
  extern char *getenv();

  if (!(env = getenv(namn))) return(-1);
  while (*env == ' ' || *env == '\t') env++;
  s = env;
  while (*s) {
```

```
    antal++;
    while (*s && *s != ' ' && *s != '\t') s++;
    while (*s == ' ' || *s == '\t') s++;
  }
  *argc = antal + 1;
  *argv = v = (char **) calloc(antal + 2, sizeof(char *));
  for (i = 1, s = env; i <= antal; i++) {
    start = s;
    len = 0;
    while (*s && *s != ' ' && *s != '\t') { len++; s++; }
    v[i] = (char *) calloc(len + 1, sizeof(char));
    strncpy(v[i], start, len);
    while (*s == ' ' || *s == '\t') s++;
  }
  return(antal);
}

char LasFilnamn(char valjare, char *filnamn, int *i, int argc, char **argv)
{
  if (*i + 1 < argc)
    if (argv[*i + 1][0] != '-') {
      strcpy(filnamn, argv[++(*i)]);
      return(0);
    }
  fprintf(stderr, "Filnamn mste ges efter -%c.\n", valjare);
  return(1);
}

void KollaArg(int filnamnTillatna, int argc, char **argv)
{ int i, fel = 0;
  char *s;

  if (ordlistor)
    ordlistor = (char **) realloc(ordlistor,
                                  sizeof(char *) * (antalOrdlistor + argc/2));
  else ordlistor = (char **) malloc(sizeof(char *) * (argc/2));
  if (filnamnTillatna) infiler = (char **) malloc(sizeof(char *) * argc);
  for (i = 1; i < argc; i++) {
    s = argv[i];
    if (*s == '-') {
      s++;
      while (*s) {
        switch (*(s++)) {
          case 'q': /* Ls inte filens ordlista */
          case 'Q': xLasOrdlista = 0; break;

          case 'e': /* Endast ett felmeddelande fr varje felstavat ord */
          case 'E': xEndastEtt = 1; break;

          case 'k': /* Kort lista med fel - acceptera s mycket som mjligt */
          case 'K':
           xKort = xEndastEtt = 1;
           xSammansatta = xAndelser = xForkortningar = xNamn = xDatatermer = 1;
           break;

          case 'r': /* Ge rttstavningsfrslag */
          case 'R': xRattstavningsforslag = 1; break;

          case 'p': /* Var petig, dvs kolla rimliga bokstavskombinationer */
          case 'P': xIntePetig = 0; break;

          case 's': /* Godknn alla sammansttningar */
          case 'S': xSammansatta = 1; break;

          case 'a': /* Godknn alla ndelser */
          case 'A': xAndelser = 1; break;

          case 'f': /* Ls in tillggsordlistan med frkortningar */
          case 'F': xForkortningar = 1; break;

          case 'n': /* Ls in tillggsordlistan med namn */
          case 'N': xNamn = 1; break;

          case 'd': /* Ls in tillggsordlistan med datatermer */
```

```
        case 'D': xDatatermer = 1; break;

        case 't': /* Ls in tillggsordlistan med TeX-termer */
        case 'T': xTex = 1; break;

        case '8': x8bitar = 1; break; /* Anvnd ISO 8859-1 */
        case '7': x8bitar = 0; break; /* Anvnd normala 7-bitstecken */

        case 'o': /* Ls tillggsordlista */
        case 'O':
         if (i + 1 < argc)
           ordlistor[antalOrdlistor] = (char *) malloc(strlen(argv[i+1]) + 1);
         if (LasFilnamn('o', ordlistor[antalOrdlistor], &i, argc, argv))
           fel++;
         else antalOrdlistor++;
         break;

        case 'i': /* Infr ord i dokumentordlistan */
        case 'I': xInforOrdlista = 1; break;

        case 'l': /* Ls frn annat lexikon */
        case 'L':
         if (LasFilnamn('l', lexfNamn, &i, argc, argv)) fel++;
         break;

        case 'u': /* Utlexikonfilnamn */
        case 'U':
         if (LasFilnamn('u', utlexfNamn, &i, argc, argv)) fel++;
         else xSkrivLexikon = xEndastEtt = 1;
         break;

        default:
         fprintf(stderr, "Oknd vljare: %c\n", *(s-1));
         fel++;
         break;
       }
      }
    } else {
      if (filnamnTillatna) infiler[antalInfiler++] = argv[i];
      else fprintf(stderr, "Oknd vljare: %s\n", argv[i]);
    }
  }
  if (fel) {
    fprintf(stderr,
            "Syntax:\n");
    fprintf(stderr,
            " stava [-o ordlista] [-afnsdqiektr8] infiler\n");
/* [-o ordlista] [-l lexikonfil] [-u utlexikonfil] [-afnsdqiektr8] infiler */
    fprintf(stderr,
" a (ndelser), f(rkortningar), n(amn), s(ammansttningar), d(atatermer),\n");
    fprintf(stderr,
" q (ls inte %s-fil), i(nfr i %s-fil), e(n felrapport per ord),\n",
    DOKUMENTORDLISTEEFTERNAMN, DOKUMENTORDLISTEEFTERNAMN);
    fprintf(stderr,
" k (kort lista med fel, samma som -eafnsd), t(exfil), r(ttstavningsfrslag)\n");
    fprintf(stderr,
" p (extra petig kontroll), 8 (8-bitarskoder anvnds fr svenska bokstver)\n");
    exit(1);
  }
}

/* KollaOrd kollar ordet som det r, med bara sm bokstver och, om det
   bara innehller versaler, med stor begynnelsebokstav */
INLINE int KollaOrd(char *ordin)
{ static char ord[LANGD], Ord[LANGD];
  if (Finns(ordin)) return(1);
  VersalerGemena(ordin,ord,Ord);
  if (*ord) if (Finns(ord)) return(1);
  if (*Ord) if (Finns(Ord)) return(1);
  return(0);
}

/* KollaDelar kollar alla ordets delar som r avskiljda med bindestreck
```

```
   var fr sej och returnerar sant d alla finns */
INLINE int KollaDelar(char *ordin, char *ord2)
{ char *t;
  do {
    t = ord2;
    while (*ordin && *ordin != '-') *t++ = *ordin++;
    *t = '\0';
    if (t - ord2 >= ORDMIN) if (!KollaOrd(ord2)) return(0);
  } while (*ordin++);
  return(1);
}

int main(int argc, char **argv)
{ char ord2[LANGD], ordin[LANGD], Ord[LANGD], urord[LANGD], **envv;
  int envc, i;

  *lexfNamn = '\0';
  *xlexfNamn = '\0';
  if (EnvArg("STAVA", &envc, &envv) > 0) KollaArg(0, envc, envv);
  KollaArg(1, argc, argv);
  if (!*lexfNamn) strcpy(lexfNamn, xKort ? KLISTFIL : LISTFIL);
  else xKort = 0; /* s inlsning av diverse lexikon inte hindras */
  Initiera();
  OppnaFiler();
  TagLexikon();
  do {
    if (xLasOrdlista) TagNastaExtraOrdlista();
    if (xInforOrdlista) OppnaUtfil();
    while (TagOrd(ordin, urord)) {
      if (KollaOrd(ordin)) continue;
      if (bindestreck) {
        if (xSammansatta) if (KollaDelar(ordin, ord2)) continue;
        SuddaBindestreck(ordin, ord2);
        if (KollaOrd(ord2)) continue;
      }
      if (xRattstavningsforslag) {
        FyrSudda();
        if (fleraInfiler) printf("%s:\t%s: ", ordfNamn, urord);
        else printf("%s: ", urord);
        GenereraAlternativaOrd(ordin);
        if (fyrAntalOrd == 0) {
          VersalerGemena(ordin, ord2, Ord);
          if (*ord2) GenereraAlternativaOrd(ord2);
        }
        if (fyrAntalOrd > 0) {
          for (i = 0; i < fyrAntalOrd; i++) {
            SkrivOrd(fyrOrd[i]);
            putchar(' ');
          }
          putchar('\n');
        } else printf("? hittar inga liknande ord\n");
      } else
      if (fleraInfiler) printf("%s:\t%s\n", ordfNamn, urord);
      else printf("%s\n", urord);
      if (inforOrdlista) fprintf(xlexf,"%s\n", urord);
      if (xEndastEtt) {
        LagraOrd(ordin);
        if (xRattstavningsforslag || !xIntePetig) LagraFyrgrafer(ordin);
      }
    }
    StangFiler();
  } while (OppnaNastaInfil());
  if (xSkrivLexikon) SkrivLexikon();
  return(0);
}
```

3 suffix.c

```
/*
   Copyright (C) 1990,1992
   Joachim Hollman och Viggo Kann
   joachim@nada.kth.se viggo@nada.kth.se
 */
#include <stdio.h>
#define TABSTORL 200
#define SUFSTORL 79
static char BARAa[] = "a";
static char TOMSTRANG[] = "";
static char *suffixtab[SUFSTORL * 2] = {
        "ad",           BARAa,
        "ade",          BARAa,
        "ades",         BARAa,
        "aktig",        TOMSTRANG,
        "an",           BARAa,
        "ande",         BARAa,
        "anden",        BARAa,
        "andena",       BARAa,
        "andet",        BARAa,
        "ar",           BARAa,
        "are",          TOMSTRANG,
        "aren",         BARAa,
        "arens",        BARAa,
        "ares",         BARAa,
        "arna",         TOMSTRANG,
        "arnas",        TOMSTRANG,
        "as",           BARAa,
        "ast",          TOMSTRANG,
        "aste",         TOMSTRANG,
        "at",           BARAa,
        "ats",          BARAa,
        "bar",          BARAa,
        "bart",         BARAa,
        "sbar",         "sbar",
        "sbart",        "sbart",
        "de",           BARAa,
        "des",          BARAa,
        "else",         BARAa,
        "elsen",        BARAa,
        "elser",        BARAa,
        "elserna",      BARAa,
        "en",           TOMSTRANG,
        "ens",          TOMSTRANG,
        "ende",         TOMSTRANG,
        "endet",        TOMSTRANG,
        "er",           BARAa,
        "erat",         "era",
        "ers",          TOMSTRANG,
        "erna",         TOMSTRANG,
        "ernas",        TOMSTRANG,
        "et",           TOMSTRANG,
        "ets",          TOMSTRANG,
        "het",          TOMSTRANG,
        "heten",        TOMSTRANG,
        "hetens",       TOMSTRANG,
        "heter",        TOMSTRANG,
        "heterna",      TOMSTRANG,
        "heternas",     TOMSTRANG,
        "ig",           TOMSTRANG,
        "iga",          TOMSTRANG,
        "igare",        TOMSTRANG,
        "igast",        TOMSTRANG,
        "igt",          TOMSTRANG,
        "ing",          BARAa,
        "ingar",        BARAa,
        "ingen",        BARAa,
        "isering",      TOMSTRANG,
```

```
        "isk",          TOMSTRANG,
        "iska",         TOMSTRANG,
        "iskt",         TOMSTRANG,
        "lig",          TOMSTRANG,
        "liga",         TOMSTRANG,
        "ligare",       TOMSTRANG,
        "ligast",       TOMSTRANG,
        "ligaste",      TOMSTRANG,
        "ligheter",     TOMSTRANG,
        "ligheterna",   TOMSTRANG,
        "ligheternas",  TOMSTRANG,
        "ligt",         TOMSTRANG,
        "man",          TOMSTRANG,
        "na",           "en",
        "nad",          "na",
        "nades",        "na",
        "nde",          "nna",
        "ning",         BARAa,
        "or",           BARAa,
        "orna",         "a",
        "rer",          "r",
        "s",            TOMSTRANG  };
static char tecken[TABSTORL], *kod[TABSTORL];
static int missl[TABSTORL], lyckad[TABSTORL];
static int fri;

char *LetaOrd(ord, slutpos)
char *ord;
int *slutpos;
{ int rad = 1, j;
  char *slutkod = NULL;
  for (j = strlen(ord) - 1; j >= 0; j--) {
    while (ord[j] != tecken[rad] )
      if (missl[rad]) rad = missl[rad];
      else return(slutkod);
    if (kod[rad]) {
      slutkod = kod[rad];
      *slutpos = j;
    }
    if (lyckad[rad]) rad = lyckad[rad];
    else return(slutkod);
  }
  return(slutkod);
}

int Suffix(ord, ord2)
char *ord, *ord2;
{ int slutpos;
  char *sufbyte;
  sufbyte = LetaOrd(ord, &slutpos);
  if (!sufbyte) return(0);
  strncpy(ord2, ord, slutpos);
  ord2[slutpos] = '\0';
  strcat(ord2,sufbyte);
  return(1);
}

void PlaceraOrd(ord, slutkod)
char *ord;
char *slutkod;
{ int rad = 0, j, senasterad = 0;
  for (j = strlen(ord) - 1; j >= 0; j--) {
    while (ord[j] != tecken[rad])
      if (missl[rad]) rad = missl[rad];
      else {
        missl[rad] = fri;
        tecken[fri] = ord[j];
        rad = fri;
        if (++fri >= TABSTORL) {
          fprintf(stderr, "Fel vid initiering av ndelsetabellen\n");
          exit(1);
```

```
        }
        goto stoppaInResten;
      }
    if (lyckad[rad]) {
      senasterad = rad;
      rad = lyckad[rad];
    } else goto stoppaInResten;
  }
  kod[senasterad] = slutkod;
  return;
 stoppaInResten:
  j--;
  for (;j>=0; j--){
    lyckad[rad] = fri;
    tecken[fri] = ord[j];
    rad = fri;
    if (++fri >= TABSTORL) {
      fprintf(stderr, "Fel vid initiering av {ndelsetabellen\n");
      exit(1);
    }
  }
  kod[rad] = slutkod;
  return;
}

void InitSuf()
{ int i;
  for (i = 0; i < TABSTORL; i++) {
    tecken[i] = '?';
    missl[i] = lyckad[i] = 0;
    kod[i] = NULL;
  }
  fri = 1;
  for (i = 0; i < SUFSTORL; i++) {
    PlaceraOrd(suffixtab[i*2], suffixtab[i*2+1]);
  }
}
```

4 stava.el

```
;; Spelling correction interface for Emacs.

(defvar stava-command "stava -k"
  "*Command to run the spell program.")

(defvar stava-filter nil
  "*Filter function to process text before passing it to spell program.
This function might remove text-processor commands.
nil means don't alter the text before checking it.")

(defun stava-buffer ()
  "Check spelling of every word in the buffer.
For each incorrect word, you are asked for the correct spelling
and then put into a query-replace to fix some or all occurrences.
If you do not want to change a word, just give the same word
as its \"correct\" spelling; then the query replace is skipped."
  (interactive)
  (stava-region (point-min) (point-max) "bufferten"))

(defun stava-ord ()
  "Check spelling of word at or before point.
If it is not correct, ask user for the correct spelling
and query-replace the entire buffer to substitute it."
  (interactive)
  (let (beg end spell-filter)
    (save-excursion
     (if (not (looking-at "\\<"))
         (backward-1-swedish-word))
     (setq beg (point))
     (forward-1-swedish-word)
     (setq end (point)))
    (stava-region beg end (buffer-substring beg end))))

(defun stava-region (start end &optional description)
  "Like stava-buffer but applies only to region.
Used in a program, applies from START to END.
DESCRIPTION is an optional string naming the unit being checked:
for example, \"word\"."
  (interactive "r")
  (let ((filter stava-filter)
        (buf (get-buffer-create " *temp*")))
    (save-excursion
     (set-buffer buf)
     (widen)
     (erase-buffer))
    (message "Stavningskontroll av `%s'..." (or description "regionen"))
    (if (and (null filter) (= ?\n (char-after (1- end))))
        (if (string= "stava" stava-command)
            (call-process-region start end "stava" nil buf)
          (call-process-region start end shell-file-name
                               nil buf nil "-c" stava-command))
      (let ((oldbuf (current-buffer)))
        (save-excursion
         (set-buffer buf)
         (insert-buffer-substring oldbuf start end)
         (or (bolp) (insert ?\n))
         (if filter (funcall filter))
         (if (string= "stava" stava-command)
             (call-process-region (point-min) (point-max) "stava" t buf)
           (call-process-region (point-min) (point-max) shell-file-name
                                t buf nil "-c" stava-command)))))
    (message "Stavningskontroll av `%s'...%s"
             (or description "region")
             (if (save-excursion
                  (set-buffer buf)
                  (> (buffer-size) 0))
                 "felaktigt"
               "korrekt"))
    (let (word newword
          (case-fold-search t)
```

```
          (case-replace t))
      (while (save-excursion
               (set-buffer buf)
               (> (buffer-size) 0))
        (save-excursion
         (set-buffer buf)
         (goto-char (point-min))
         (setq word (downcase
                     (buffer-substring (point)
                                       (progn (end-of-line) (point)))))
         (forward-char 1)
         (delete-region (point-min) (point))
         (setq newword
               (read-input (concat "`" word
                                   "' oknt ord; ge det ord du vill ha istllet: ")
                           word))
         (flush-lines (concat "^" (regexp-quote word) "$")))
        (if (not (equal word newword))
            (progn
             (goto-char (point-min))
             (query-replace-regexp (concat "\\b" (regexp-quote word) "\\b")
                                   newword)))))))

(defun stava-string (string)
  "Check spelling of string supplied as argument."
  (interactive "Ge strngen: ")
  (let ((buf (get-buffer-create " *temp*")))
    (save-excursion
     (set-buffer buf)
     (widen)
     (erase-buffer)
     (insert string "\n")
     (if (string= "stava" stava-command)
         (call-process-region (point-min) (point-max) "stava"
                              t t)
       (call-process-region (point-min) (point-max) shell-file-name
                            t t nil "-c" stava-command))
     (if (= 0 (buffer-size))
         (message "%s ; r korrekt" string)
       (goto-char (point-min))
       (while (search-forward "\n" nil t)
         (replace-match " "))
       (message "%sfelaktigt" (buffer-substring 1 (point-max)))))))

(defun forward-1-swedish-word ()
  (interactive)
  (forward-word 1)
  (let ((ch (char-after (point))))
    (if (or (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ?- ch))
        (progn
          (forward-char 1)
          (forward-1-swedish-word)))))


(defun backward-1-swedish-word ()
  (interactive)
  (forward-word -1)
  (let ((ch (char-after (1- (point)))))
    (if (or (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ? ch)
            (char-equal ?- ch))
```

```
(progn
  (forward-char -1)
  (backward-1-swedish-word)))))
```