"Corect" — un "spell-check"-er pentru limba română

Mihai Budiu — mihaib@cs.cmu.edu, `http://www.cs.cmu.edu/~mihaib`

16 mai 1997

Subiect: un corector de texte pentru limba română

Cunoştinţe necesare: familiaritate cu sisteme de prelucrare a textelor

Cuvinte cheie: spell-check, emacs, ispell, standard, inernaţionalizare

Cuprins

Folosind programul `ispell` am construit un pachet de utilitare care uşurează scrierea şi corectarea textelor în limba română. Pachetul este, sper, relativ uşor de instalat şi folosit pe platforme Unix, şi este disponibil oricui fără restricţii.

Poate fi obţinut prin web de la `http://www.cs.cmu.edu/~mihaib/ftp/rom-spell.taz`. În acest articol discut în oarece detaliu despre una dintre sculele pe care pachetul se bazează (excelentul program `ispell`) şi justific unele dintre deciziile pe care le-am luat cînd am construit pachetul. Pe alocuri divaghez despre importanţa unui design clar separat, şi despre funcţia standardelor în ziua de azi.

1 Programul `ispell`

`ispell` este software "free", deci accesibil oricui. Rolul lui este destul de simplu: analizează texte şi indică cuvintele a căror grafie nu o cunoaşte, permiţînd în felul acesta corectarea unora dintre erorile de dactilografie. `ispell` vine de la "fabricanţi" cu o întreagă suită de programe care permit construirea, manipularea şi întreţinerea de dicţionare.

`ispell` este prin natura lui un program nu prea inteligent: are la îndemînă un dicţionar cu *toate* cuvintele corecte, iar ceea ce face este să caute fiecare cuvînt din textul de corectat în dicţionar. Dacă găseşte cuvîntul îl socoteşte bun, dacă nu, se uită în dicţionar după cuvinte care sunt "apropiate" ca scriere de cel absent, pe care apoi le propune spre înlocuire, presupunînd ca acel cuvînt a fost greşit scris.

Ce înseamnă că "două cuvinte sunt aproape" pentru `ispell`? Un cuvînt este aproape de altul dacă se poate obţine printr-una din următoarele operaţii:

- Adăugarea unei litere;
- Ştergerea unei litere;
- Transpunerea a două litere consecutive.

De asemenea, `ispell` poate identifica lipsa unui spaţiu dintre două cuvinte.

Deşi un astfel de program nu poate oferi nici o garanţie în producerea de texte corecte, este extrem de eficace în a elimina o mare parte dintre erorile tipografice.

Chiar faptul că `ispell` nu are nici o cunoştinţă despre limbă îl face foarte util, pentru că permite folosirea lui cu texte scrise în orice limbă, cu condiţia posedării unui dicţionar potrivit.

2 Un principiu de Design

Programe de genul `ispell` există de multă vreme în lumea Unix (de ex. programul `spell`; to spell = a scrie literă cu literă). Ceea ce face `ispell` atrăgător este universalitatea lui. `ispell` este un exemplu tipic de program care încarnează un foarte important principiu de proiectare în calculatoare. Acest principiu se manifestă în separarea dintre *mecanism* şi *politică* (policy/mechanism). Principiul indică faptul că un aparat trebuie să fie cît mai universal, ca să nu oblige pe cel care îl foloseşte la anumite feluri de manipulare, care să îi reducă utilitatea.

Tehnica este extrem de importantă în proiectarea sistemelor de operare, şi în general a programelor care servesc drept unelte[1] (de exemplu bibliotecile). Un sistem de operare trebuie de exemplu să-ţi ofere toate metodele prin care să poţi controla *cînd* se vor executa programele tale, dar să nu impună o anumită ordine de execuţie a lor. Sistemul de operare este cel care execută programele, dar utilizatorul este cel care indică ordinea şi priorităţile.

Un alt exemplu interesant este în reţelele de calculatoare: protocoalele de transmisiune a datelor ştiu să mute biţi de la un calculator la altul, chiar foarte îndepărtat. În acest caz mecanismul este cel

[1]Prin antiteză cu programele care fac ceva util ele însele: aplicaţiile.

care transportă datele. Politica de alegere a traseului pe care le urmează datele este însă complet separată de mecanismul de transport; ea este decisă de un set complet separat de protocoale, numite protocoale de rutare. Această flexibilitate permite construirea de protocoale de rutare noi şi înlocuirea celor vechi "din mers", fără a întrerupe funcţionarea reţelei în vreun fel.

Pe scurt: dacă faci un aparat complicat, nu restrînge modul în care poate fi folosit, ci construieşte-i suficiente mînere cît să poată fi controlat în cît mai multe feluri.

3 Pachetul pentru limba română

Unde trebuie puse mînerele nu este întotdeauna evident. Proiectantul variantei internaţionale a lui `ispell`, Geoff Kuenning, a studiat cu atenţie locurile în care politica era îngropată în interiorul programului, şi a încercat să o extragă în afară. Acest lucru este vizibil utilizatorului prin faptul că pentru a folosi `ispell` trebuie să scrii o grămadă de fişiere auxiliare, care controlează modul în care lucrează. Ceea ce am şi făcut, pentru a putea face `ispell` utilizat cu texte româneşti.

În cele ce urmează voi descrie pe scurt conţinutul pachetului; această descriere îmi va prilejui identificarea porţiunilor din `ispell` care sunt supuse controlului utilizatorului.

3.1 Fişierele de prefixe/sufixe

Principala piesă din configurarea `ispell` este fişierul de afixe. Numele nu este foarte fericit ales, pentru că funcţiunea acestuia este, pe lîngă de a indica metode de derivare cu prefixe şi sufixe (care pot fi folosite pentru a comprima dicţionarele), şi descrierea parametrilor funcţionării lui `ispell`.

O primă alegere cu care m-am confruntat a fost dacă dicţionarele vor conţine semne diacritice sau nu. Pentru că nu m-am putut hotărî, pînă la urmă am construit două configuraţii şi două dicţionare, pentru ambele cazuri.

Pentru cazul semnelor diacritice a trebuit să fac faţă următoarelor două decizii:

- Dacă folosesc sau nu noile reguli de scriere ale Academiei cu â; rezultatul îl ghiciţi din textul de faţă, care este corectat cu `ispell`;
- Cum voi reprezenta semnele diacritice. Această problemă este extrem de spinoasă, şi îmi va da prilejul să fac o lungă digresiune:

3.2 Problema internaţionalizării

Calculatoarele depind într-o sumedenie de moduri subtile de faptul că au fost concepute de oameni care foloseau mai ales limba engleză (orice istorie a calculatoarelor va recunoaşte rolul americanilor şi, ceva mai puţin, al englezilor). Modul în care sunt prelucrate textele, în care sunt afişate şi citite este adesea strîns legat de caracteristicile speciale ale limbii engleze. De acest lucru şi-au dat seama cei care au încercat să universalizeze folosirea calculatoarelor, pentru a le adapta şi altor naţii.

Fără a avea pretenţia de a trata exhaustiv această problemă, să aruncăm o privire asupra influenţei limbii asupra programelor.

3.2.1 Traseul informaţiei

Pentru a prelucra texte avem trei categorii mari de operaţii:

- Citirea textelor de la tastatură;
- Prelucrarea textelor prin feluriţi algoritmi;
- Afişarea (vizualizarea) textelor.

Pentru fiecare din aceste activităţi este responsabil un program distinct. De citirea tastaturii se ocupă "driver"-ul de terminal. Acesta are de obicei mai multe funcţiuni:

- Deduce combinaţia de taste apăsată; o reprezintă printr-un şir de coduri;
- Traduce codurile la o reprezentare "standard"; de exemplu cînd apeşi tasta pe care scrie "A" returnează codul ASCII al lui "a", care e 97.
- Cîteodată face o minimă editare a textului: de exemplu dacă apeşi pe DELETE şterge un caracter.

În funcţie de sistemul de operare, utilizatorul are mai multe opţiuni la-ndemînă pentru a influenţa driverul: poate să citească direct codurile, să influenţeze modul în care se face traducerea, să evite editarea automată textului.

Dacă lucrezi pe un calculator la distanţă, de exemplu conectat prin `telnet` prin reţea, lucrurile se complică, pentru că între driver-ul care citeşte tastatura (aflat pe calculatorul local) şi aplicaţia ta (aflată la distanţă) se mai interpun încă două programe care codifică fiecare tastă apăsată:

- Programul local de `telnet`, citeşte tastele de la driver şi le traduce într-un mod standardizat pentru comunicaţia prin reţea;
- Programul de la distanţă, demonul `telnetd`, care primeşte de la `telnet` codurile standard ale tastelor şi le traduce pentru maşina de acolo.

Afişarea textelor este iarăşi de obicei sarcina unui "driver". Dacă avem de-a face cu un terminal alfa-numeric (sau o fereastră care emulează un astfel de terminal prin software), atunci de obicei driver-ului i se dau secvenţe de caractere ASCII, pe care ştie el cum să le afişeze; din cînd în cînd i se dau şi coduri numite "caractere de control", care de obicei nu lasă urme pe ecran, ci îi spun terminalului *cum* să manipuleze caracterele ce urmează. De exemplu putem indica, folosind caractere de control, în ce loc pe ecran să se scrie următorul caracter ASCII. Care este secvenţa de caractere de control care spune acest lucru depinde de terminal; fiecare fabricant a inventat propriile convenţii.

De cînd firma Xerox a inventat ecranul grafic ("bitmapped screen"), a deveni un lucru comun ca interfaţa dintre aplicaţie şi utilizator să nu fie un simplu şir de caractere pe care terminalul le afişează, ci o matrice mare de punctuleţe care pot fi controlate individual. Aplicaţiilor li se pune la îndemînă posibilitatea de a desena orice formă doresc; foarte frecvent însă sistemul de operare pune la dispoziţie funcţii de bilbiotecă pentru a desena caractere cu felurite forme ("fonturi").

În fine, ultimul aspect al prelucrării de texte este procesarea lor cu feluriţi algoritmi: căutări, sortări, tehnoredactare, corectură, editare, transmitere de poştă electronică, etc. Vom reveni pe scurt asupra fazei de prelucrare după ce discutăm despre standardizare.

3.2.2 Standardizarea

Un element cheie în interacţiunea între aceste părţi (drivere, aplicaţii, biblioteci, sistemul de operare) este faptul că *toate folosesc aceeaşi metodă de codificare*. Cînd apăsaţi tasta "A", driverul de tastatură întoarce 97, iar aplicaţia ştie că a primit un "A", pe care îl poate da bibliotecii de afişare, care ştie că trebuie să deseneze o astfel de literă.

Oricine a lucrat pe o tastatură prost configurată, sau a încercat să acceseze de la distanţa o aplicaţie care decodifică ciudat tastatura, sau a manipulat tabele de translatare ale codurilor, sau a primit mail cu caractere internaţionale va înţelege importanţa acestei convenţii. Literei "A" i se asociază codul 97 de către codul ASCII (American Standard Code for Information Interchange), care este standardizat de ANSI (American National Standard Institute); e un fel de STAS local. Din fericire, datorită dominaţiei SUA în arena calculatoarelor, ASCII este practic implementat de toată lumea. Şi organizaţia mondială a standardelor ISO (International Standards Organization) a acceptat ASCII sub numele ISO 646. Codificarea ASCII însă recomandă numai 128 de simboluri (din care 32 nici măcar nu sunt vizibile, ci sunt "caractere de control"). De îndată ce ieşim din perimetrul acestora, domneşte haosul.

Pentru caractere pe 8 biţi există standardul ISO-8859, care oferă o duzină de interpretări pentru caracterele cu coduri peste 128. Pentru limba română cea mai interesantă interpretare este cea dată de ISO-8859-2, numită şi Latin-2, care conţine toate semnele speciale cu diacritice româneşti. Problema acum este însă: înainte de a cădea de acord ce semn este cel cu codul 200, trebuie să cădem de acord ce cod folosim din duzina de variante ISO sau alte variante. Plus că fiecare aplicaţie trebuie să fie capabilă să interpreteze toate variantele, etc.

Tentative de standardizare şi mai îndrăzneţe s-au făcut: un consorţiu gigant, din care fac parte Microsoft, IBM, Xerox, Sun şi alte (zeci) de firme din lumea calculatoarelor, a fost înfiinţat, pe nume Unicode. Acesta, cu ajutorul a zeci de experţi lingvişti a propus o standardizare uniformă a tuturor caracterelor, din toate limbile lumii, folosind pentru aceasta 16 biţi pe caracter. Fiecare caracter ar avea astfel o reprezentare unică şi ne-ambiguă. Dificultăţile de înfrînt sunt enorme; de exemplu trebuie cumva luate în considerare standardele naţionale (cum ar fi cel elaborat de Japonezi) şi reconciliate.

Din păcate concordia nu este încă realizată, pentru că ISO a propus în acelaşi scop propriul ei standard, diferit de Unicode, pe 32 de biţi, numit ISO DIS 10646.

Dificultatea la ora asta este deci nu că nu există standarde, ci că există prea multe. O aplicaţie poate folosi propriile ei convenţii interne pentru citirea tastaturii/modificare/afişare, dar de îndată ce se pune problema *schimbului* de informaţii, absenţa unui standard unic (sau a unui număr mic de standarde) face problema intratabilă.

În general, viaţa noastră de toate zilele depinde enorm de standarde, într-un fel pe care nici nu-l realizăm. Dacă nu am avea standarde trenurile nu ar pute circula pentru că ecartamentul ar fi diferit în ţări diferite (de altfel, la ruşi chiar este mai mare), aparatele electrice nu ar funcţiona decît unde au fost proiectate (şi aşa americanii merg la 110V), maşinile ar trebui să care benzina de acasă ca să se potrivească cifra octanică, am vedea la televizor numai ce ar transmite bunica, cu care folosim aceeaşi codificare, etc, etc.

3.2.3 Prelucrările pe texte

Înainte de a ne îndepărta de zona standardelor internaţionale, să mai aruncăm o privire asupra unor întrebări care sunt legate de reprezentarea caracterelor, şi al căror răspuns influenţează modul în care algoritmii vor procesa textele:

- Nu toate limbile se scriu în aceeaşi direcţie; ebraica se scrie de la dreapta la stînga, iar japoneza pe verticală. Cînd am un text englezesc cu citate din vechiul testament în ebraică, cum reprezint textul în calculator, în ce ordine?
- Literele se pot scrie adesea în felurite moduri; chiar lăsînd la o parte informaţia legată de font, dimensiune şi orientare. În arabă unele litere se scriu diferit în funcţie de vecinii lor;
- Voi reprezenta ă ca două semne (a şi căciula) — din care căciula este un semn care "nu ocupă spaţiu", deci se scrie peste cel anterior — sau ca o singură literă?
- Reprezint "A" şi "a" ca pe o aceeaşi literă, într-un caz modificată (mărită), sau ca pe două litere deosebite?
- În anumite limbi un semn se comportă ca mai multe litere (în germană ß se sortează ca şi cum s-ar scrie "ss"), dar se scrie ca una singură. În alte limbi e invers: în Spaniolă "ll" este considerată o singură literă la sortare. Algoritmii de sortare devin deci mult mai complicaţi, pentru că nu mai pot face simple comparaţii literă cu literă (o simplă sortare lexicografică).
- În limba chineză literele nu sunt ordonate alfabetic, ci după numărul de liniuţe cu care se desenează.
- Informaţii ca: sfîrşitul de paragraf, de pagină, indentarea, numărul de spaţii albe sunt sau nu reprezentabile prin caractere (cum sunt de exemplu în setul ASCII "New-Line" sau "Tab")?
- Cînd scriu programe în C, cum reprezint caractere internaţionale pentru mesaje date utilizatorului?
- Cum pot crea un fişier cu nume în Swahili?
- Cum se scrie o expresie regulată în Kanji?

Oricare din aceste probleme poate fi rezolvată, dar pînă cînd nu toată lumea le va rezolva pe toate în acelaşi fel (prin intermediul unui standard), programele riscă să manipuleze obiecte pe care nu le mai înţelege nimeni în afară de autori.

Probabil că învingător în lupta standardelor va ieşi pînă la urmă Unicode, datorită suportului unor firme extrem de mari, şi datorită calităţilor sale tehnice. Standardul este descris în 8 cm[2] de hîrtie la ora actuală, şi mai creşte.

Sistemul de operare Windows NT de la Microsoft este construit integral cu Unicode în interior; totul, de la nume de fişiere la mesaje de eroare este construit pe 16 biţi. Biblioteci speciale transformă datele manipulate de programele "vechi" (care lucrează cu caractere pe 8 biţi) în Unicode înainte de a ruga driverele din nucleu să le afişeze. Chiar faptul ca Windows NT domină piaţa sistemelor de operare la ora actuală este un motiv major pentru ca Unicode să învingă.

[2] grosime.

3.2.4 Reprezentarea diacriticelor

Închei aici digresiunea mea despre standarde şi reprezentarea caracterelor internaţionale, şi revin la pachetul meu mult mai modest, care încearcă să rezolve unele dintre problemele limbii române.

Hotărîrea mea finală a fost să reprezint absolut tot textul folosind semne ASCII, lucru care o să mă scutească de o sumedenie de probleme de portabilitate, pentru că virtual toate terminalele din lume recunosc setul de caractere ASCII. În schimb mi-am propus să folosesc o convenţie care să aibă următoarele proprietăţi:

- Să folosească semnele ASCII într-un mod ne-ambiguu (de exemplu dacă aş fi folosit semnul `a+` pentru ă, aş fi avut mari dificultăţi în a distinge formulele matematice de texte cu diacritice; convenţia pe care am aleas-o în final nu e perfect ne-ambiguă, dar e tolerabil de neambiguă, cred;

- Ca o consecinţă a ne-ambiguităţii, ar trebui să fie foarte uşor de scris un program care să convertească un text scris cu diacritice folosind convenţia mea, în orice altă convenţie de reprezentare. Practic dacă fac un *search-and-replace* pe un text pot obţine orice altă convenţie. Par-examplu, dacă dvs. folosiţi pentru ă semnele /a, şi eu folosesc {a}, atunci înlocuind uniform {a} cu /a convertesc un text din reprezentarea mea în a dumneavoastră.

- Semnele pe care le folosesc să fie uşor de tastat. În limba română există o sumedenie de cuvinte cu semne diacritice, deci mîna mea va trebui să nu se deplaseze prea mult pe tastatură pentru a scrie semnele însoţitoare; de preferabil să nu folosesc SHIFT sau ceva înrudit.

- Metoda să se împace bine cu alte programe care manipulează texte. De exemplu eu scriu texte folosind editorul Emacs (vedeţi şi articolul meu din PC-Report pe Mai 1997, aflat şi la `http://www.cs.cmu.edu/~mihaib/articles`), iar în Emacs tasta `M-f` se deplasează peste un cuvînt. Aş fi vrut ca semnele diacritice pe care le folosesc să nu fie considerate de Emacs ca fiind înafara cuvintelor.

Pînă la urmă alegerea s-a oprit asupra semnului apostrof: '. De aceea voi scrie acum literele astfel: `'a 'i 's 't` pentru ă, î, ş, ţ.

Probabil că alegerea pentru â a fost mai puţin fericită: am ales `"a`.

Dar să revenim la fişierele de configurare pentru `ispell`[3].

Odată ce am hotărît ce litere voi folosi, am purces la a scrie un fişier de configurare pentru `ispell`. Formatul fişierului este documentat în paginile de manual ale programului `ispell`. Acesta este conţinutul fişierului `rom.signs.aff`: fişierul de afixe pentru limba română cu semne diacritice (cel pentru configuraţia fără diacritice este mai simplu).

```
nroffchars    ().\\*
texchars      ()\[]{}<\>\\$*.%
stringchar    "'a" "'A"
stringchar    "'i" "'I"
```

[3] Ca dovadă că principiile enumerate mai sus au fost bune, iniţial alesem semnul \ pentru diacritic; schimbarea la apostrof a fost însă foarte simplă, şi a fost cauzată de dorinţa de a inter-opera cu LaTeXpentru care semnul \ are o semnificaţie specială.

```
stringchar      "'s" "'S"
stringchar      "'t" "'T"
stringchar      "\"a" "\"A"
wordchars       [a-z] [A-Z]
```

Folosind un limbaj foarte simplu îi descriu lui `ispell` care caractere au semne speciale în fişierele scrise pentru `nroff` şi TEX (acestea sunt două programe foarte răspîndite în lumea Unix pentru tehnoredactarea de texte, care au folosesc nişte limbaje speciale pentru a descrie aranjarea în pagină a textului. Practic eu trebuie să îi spun lui `ispell` că atunci cînd manipulez un text scris pentru `nroff`, respectiv TEX, trebuie să ignore cuvintele care descriu aranjarea în pagină, şi să corecteze numai restul textului.)

Apoi îi indic lui `ispell` că voi interpreta perechile de semne apostrof-a, apostrof-A, etc. drept o singură literă în textele mele. Asta este foarte important pentru că îi spune lui `ispell` că "fîş" este obţinut din "fşî" prin schimbarea a două litere (deci este "aproape"), şi nu prin permutarea unui şir de 4 caractere (cuvintele se vor scrie cu convenţia mea astfel: `f'i's`   `f's'i`).

În fine, indic faptul că semnele ASCII obişnuite pentru litere sunt componente ale cuvintelor. `ispell` permite reguli şi mai rafinate, pentru a indica de pildă că anumite caractere sunt considerate litere numai dacă apar în interiorul unui cuvînt (cum ar fi de pildă liniuţa de unire –), sau pentru a indica faptul că `nroff` sau TEX denotă altfel un acelaşi caracter.

Observaţi cît de flexibil este `ispell`: comportarea programului este influenţată de o grămadă de parametri dinafară, din fişierul de configurare. Aceasta este separarea de politică şi mecanism de care vorbeam, care îi permite lui `ispell` să fie simultan folosit în condiţii atît de diferite.

3.3 Dicţionarele

A doua problemă spinoasă era: de unde dicţionare? La această întrebare am răspuns imediat.

3.3.1 Achiziţia de date

`ispell` este un program interactiv, care este suficient de înţelept să fie de acord că nu ştie totul. De fiecare dată cînd `ispell` găseşte un cuvînt necunoscut, pe lîngă faptul că îmi oferă o listă de posibile corecturi, îmi oferă posibilitatea de a introduce cuvîntul în dicţionar. Exact pe asta m-am bazat şi eu: voi construi dicţionarul incremental, rulînd `ispell` pe texte (cît se poate de) corecte. După o fază plictisitoare în care mă va întreba mai toate cuvintele, va învăţa vocabularul meu de bază şi va deveni eficace.

Construcţia dicţionarelor este în plină desfăşurare, deşi ambele (cel cu diacritice şi cel fără) au cîte 40 de mii de cuvinte la ora actuală. Principala mea sursă de vocabular este ediţia pe Internet a ziarelor "România Literară" şi "Dilema", plus, mai puţin, textele pe care le scriu eu însumi.

3.3.2 Analiza statistică

După ce am strîns cîteva mii de cuvinte, am folosit celelalte programe din pachetul cu care vine `ispell` pentru a face o analiză statistică a dicţionarelor. Analiza este utilă pentru că în fişierele de configurare îi pot descrie lui `ispell` simple reguli de derivare cu prefixe şi sufixe, pe care apoi el le foloseşte pentru a reduce mărimea dicţionarelor, memorînd o singură dată rădăcinile comune.

Pachetul care conţine programul `ispell` oferă pentru acest scop programele `findaffix`, care analizează exhaustiv un dicţionar, generînd informaţii despre potenţialele derivări, şi `tryaffix`, care evaluează economia care se poate obţine folosind un anumit affix.

3.3.3 Compresia prin prefixe şi sufixe

Cum se folosesc afixele pentru a reduce mărimea dicţionarului? Pentru fiecare afix se defineşte (manual) cîte o prescurtare de o literă. De exemplu, pentru prefixul "re-" am alocat litera E. Atunci, în loc ca în dicţionar să apară cuvintele "educare" şi "reeducare, va apărea doar o înregistrare de forma "educare/E". Practic am înlocuit un cuvînt întreg cu două litere (/ şi E).

Pachetul `ispell` vine împreună cu un alt program numit `munchlist`, care, dîndu-i-se un dicţionar şi o listă de afixe, comprimă dicţionarul la o reprezentare minimă. Rezultatul este foarte eficace; chiar cu fişierele relativ primitive de afixe pe care le-am făcut am reuşit să reduc mărimea dicţionarelor la aproximativ 50-55%.

Restul fişierelor de configurare a afixelor este ocupat cu regulile prin care afixele se ataşează cuvintelor. `ispell` ştie să opereze mici modificări asupra rădăcinilor; asta creşte enorm utilitatea afixelor. Iată un fragment tipic dintr-o descriere a unui "flag" din `rom.signs.aff`:

```
suffixes

flag *I:
    U           >       LUI           # bou  > boului
     [^UA]      >       ULUI          # porc > porcului
     [GC] A     >       -A,II         # vaca > vacii
     [^CG] A    >       -A,EI         # baba > babei
```

Traducere: indicatorul I va funcţiona în 4 feluri, depinzînd de forma rădăcinii cuvîntului la care se aplică:

1. Dacă rădăcina se termină în "U", atunci indicatorul I va cauza concatenarea şirului "lui". De exemplu, în dicţionar cuvîntul "bou/I" va genera simultan "bou" şi "boului";

2. Dacă rădăcina *nu* (negaţia este indicată de semnul ^) se termină în "U" sau "A", atunci adaug şirul "ului";

3. Dacă rădăcina se termină în "ga" sau "ca", atunci regula va şterge "A"-ul de la coadă şi va concatena un "ii";

4. În fine, dacă rădăcina se termină în "a", dar nu în "ca" sau "ga", atunci şterg "A"-ul şi adaug "ei".

Observaţi că aceste reguli nu sunt folosite pentru a *genera* noi cuvinte din cele existente, ci doar pentru a comprima un set de cuvinte dat, deci nu pot greşi dacă avem de-a face cu o excepţie (de exemplu cuvîntul "poarta", care după regulile de mai sus ar deveni "poartei"). Singura consecinţă este că ambele cuvinte vor apărea în dicţionar, pentru că nu se poate aplica compresia (voi avea şi "poarta" şi "porţii" în dicţionar).

3.4 Interfaţa cu emacs

`ispell` este prin construcţie un program interactiv: analizează textul, şi la prima eroare scrie pe ecran porţiunea de text suspicioasă, indicînd posibile corecturi, şi aşteptînd ca utilizatorul să ia măsuri.

Dl. Kuenning însă a fost suficient de înţelept pentru a înzestra `ispell` cu un mod de funcţionare “batch”: ne-interactiv. Dacă `ispell` este pornit cu modificatorul `-a` atunci nu mai scrie nimic pe ecran şi nu mai citeşte tastatura, ci scrie şi citeşte de la intrarea şi ieşirea sa standard. Practic de la intrare `ispell` citeşte un cuvînt, iar la ieşire trimite şiruri de caractere care descriu părerea lui despre acel cuvînt (dacă e corect, dacă nu ce variante sunt, etc). De exemplu, (de data asta în engleză):

```
$ echo helo | ispell -a
@(#) International Ispell Version 3.1.20 10/10/95
& helo 9 0: halo, held, hell, hello, helm, help, hero, he lo, he-lo
$
```

Convenţiile prin care `ispell` răspunde sunt uşor complicate; pe scurt, în linia de mai sus a spus că nu crede că acel cuvînt e corect, (prin semnul &), şi a indicat apoi 9 posibilităţi corecte, separate prin virgule.

Aceasta este o altă instanţă a faimoasei separaţii dintre politică şi mecanism: mecanismul de corectare nu implementează şi politica de interacţiune.

Asta permite interfaţarea altor programe cu `ispell`. De exemplu editorul de texte Emacs ştie să converseze cu `ispell`, bagîndu-i pe gît (intrarea standard) cuvintele dintr-un buffer unul cîte unul. În funcţie de răspuns discută el însuşi cu utilizatorul, pentru a înlocui cuvintele greşite.

3.5 Fişierul de stil pentru LaTeX

Pentru că îmi tehnoredactez fişierele în LaTeX, era natural să caut o metodă de a folosi convenţia de scriere cu apostroafe direct în LaTeX. Nu a fost prea dificil, pentru că am dat peste un fişier de stil construit de Alexandru şi Dan Corlan pe care mi l-am însuşit, şi în care am operat nişte minuscule modificări. Fişierul pune la-ndemînă un nou “environment”, numit `romana`, în interiorul căruia semnul apostrof şi respectiv ghilimelele produc diacritice. Practic pot scrie acum texte astfel:

```
\documentstyle[romana-]{article}
\begin{document}
'In aceast'a parte apostroful
nu are nici o influen't'a.
\begin{romana}
Dar 'in aceast'a parte se
transform'a 'in diacritice.
\end{romana}
\end{document}
```

şi voi genera următorul document:

'In aceast'a parte apostroful nu are nici o influen't'a. Dar în această parte se transformă în diacritice.

Fişierul de stil ar putea beneficia de mici îmbunătăţiri, pentru că anumite combinaţii se comportă ciudat (de exemplu nu se mai pot scrie acolade după ghilimele), dar este mai adesea util decît dăunător.

3.6 Interfaţa corectorului de limba română cu utilizatorul

3.6.1 Instalare

Pentru a putea folosi `corect`, trebuie întîi sa aveţi pachetul cu `ispell`. Necesare sunt fişierele `buildhash` şi `ispell`; de celelalte vă puteţi lipsi. Dacă aveţi un sistem Unix, le puteţi lua de la `ftp://prep.ai.mit.edu/pub/gnu`.

Pagina lui `ispell` este la `http://fmg-www.cs.ucla.edu/fmg-members/geoff/ispell.html`; acolo mai puteţi afla o grămadă de lucruri utile despre acest program.

Cel care instalează pachetul de fişiere are o misiune relativ simplă: trebuie să decidă locul unde fişierele se vor plasa şi cum ajung utilizatorii la ele. Există două variante de bază:

1. Instalarea este făcută pentru întregul sistem (de administrator, în aşa fel încît toţi utilizatorii să poată beneficia);
2. Un utilizator fără privilegii îşi instalează pachetul pentru uzul personal în contul lui.

Cel care instalează va trebui să citească `Makefile`-ul şi să modifice cîteva variabile. Valorile care ar fi interesante sunt `ROMSPELLBASE` şi `LATEXDIR`, şi indică unde se vor plasa dicţionarele, respectiv fişierul de stil pentru LaTeX.

Pentru ca utilizatorii să poată accesa fişierul de stil, el trebuie să fie într-un loc în care LaTeXsă-l poată căuta. Administratorul îl poate pune acolo unde sistemul local are toate celelalte fişiere de stil. Un utilizator obişnuit îl poate pune oriunde, şi apoi poate indica lui LaTeXacel loc folosind variabila `TEXINPUT`. De exemplu, eu mi-am pus fişierul `romana-.sty` în `$HOME/data/tex` şi am pus variabila astfel:

```
export TEXINPUTS=".:$HOME/data/tex:"
```

În plus, pentru ca Emacs să poată invoca `ispell` pentru a corecta texte româneşti, fişierul `for-emacs`, care este generat automat, ar trebui executat de Emacs-ul tuturor utilizatorilor care scriu în română. Iarăşi, există două variante: fiecare utilizator poate concatena acest fişier la propriul fişier de iniţializare `.emacs`, sau acest fişier poate fi pus de administrator într-un fişier global de iniţializare al lui Emacs (de exemplu `site-start.el`).

3.6.2 Utilizare

După ce administratorul a făcut treburile murdare de instalare, folosirea corectorului ar trebui să fie banală. Datorită unui mic script, tot ce trebuie făcut pentru a corecta un text este de a tasta:

```
corect fisier-de-corectat
```

Simplu ca bună ziua!

(Dacă fişierul este scris în română cu diacritice, tastaţi `corect -s fisier`.)

Pentru a corecta din Emacs folosiţi comenzile Emacs:

```
M-x ispell-change-dictionary
```

La întrebarea care urmează, răspundeţi `romana`, respectiv `rom-diacritice`, după cum doriţi. Apoi tastaţi:

```
M-x ispell-buffer
```