emToken: Unicode-képes tokenizáló magyar nyelvre

Mittelholcz Iván

MTA Nyelvtudományi Intézet,
1068 Budapest, Benczúr u. 33., e mail: mittelholcz.ivan@nytud.mta.hu

Kivonat Cikkünkben az emToken tokenizáló programot mutatjuk be. Ennek főbb tulajdonságai között említhető, a széleskörű UTF 8 támogatás, a konfigurálhatóság, az automatikus tesztkörnyezet és a programkönytár által nyújtott API. Az előállított – XML vagy JSON formátumú – kimenet detokenizálható. A program forráskódja szabadon elérhető GPLv3 licenc alatt. Az emToken az `e magyar` eszközlánc tokenizálásért felelős modulja.

Kulcsszavak: tokenizálás, mondatra bontás, természetesnyelv feldolgozás, kutatási infrastruktúra

1. Bevezetés

Cikkünkben az `e-magyar` szövegfeldolgozó eszközlánc részeként kifejlesztett új magyar tokenizáló és mondatra bontó eszközt mutatjuk be. Az `e-magyar` rendszerről átfogó ismertetést ad [5].

Magyar nyelvű szövegek tokenizálására széles körben használják a Huntoken[1] programot. Ez lényegében unixos parancssori szűrőknek egy bash szkript által összekötött sorozata. Maguk a szűrők a Flex[2] lexergenerátorral előállított, majd lefordított bináris fájlok. Ezek a standard inputról olvassák a bemenetüket, és a standard outputra írják a kimenetüket. A bash szkript a Unixban általánosan alkalmazott pipeline mechanizmust használja ezek összekötésére. A Flex-fájlokban definiált szűrők végzik a szöveg előzetes tisztítását (pl. a HTML-karakterentitások kezelését), a mondatra bontást, a rövidítések kezelését, magát a tokenizálást és a poszttokenizálást.

A tokenizálás feladatával kapcsolatban megfogalmazható néhány fontos igény, amit a Huntoken számos erénye mellett sem tud kielégíteni. Ezek a következők:

A Flexben nincs Unicode-támogatás, csak az egy bájtos karakterkódolásokat tudja helyesen kezelni. Ennélfogva a Flexre támaszkodó Huntoken sem képes a manapság leginkább elterjedt UTF-8-as karakterkódolású szövegek feldolgozására.

[1] http://mokk.bme.hu/resources/huntoken/
[2] http://flex.sourceforge.net/

A Huntoken készítésének idejében a tokenizálókkal még sok olyan feladatot is elvégeztettek, ami nem tartozik szorosan véve a tokenizáláshoz, de reguláris kifejezésekkel hatékonyan megoldhatónak gondoltak. Így a Huntoken is tartalmaz olyan elemeket, amelyek ma már egyértelműen a tulajdonnév-felismerők vagy éppen a morfológiai elemzők feladata. Ide tartozik például az ISBN számok vagy a képletek felismerése, és bizonyos ragozási alakok felismerése és címkézése.
Sok tokenizáló, köztük a Huntoken is, a szóközjellegű karaktereket egyszerűen kiszűri a szövegből, és csak a tartalmas tokeneket és punktuációkat adja vissza. Ugyanakkor a későbbi feldolgozási lépésekben hasznos lehet megőrzésük (például van olyan eset, amikor nem mindegy, hogy egy írásjel tapadt az őt megelőző szóra, vagy szóköz volt köztük). Ezt az igényt a *detokenizálhatóság* címszóval szokás jelölni: egy tokenizáló kimenete detokenizálható, ha az eredményül kapott szövegből az eredeti rekonstruálható.

Az alábbiakban bemutatásra kerülő emToken[3] a fentebb megfogalmazott igényeket kívánja teljesíteni. Ehhez támaszkodik a már meglévő Huntokenre, de jelentősen el is tér attól, ahogy azt az alábbi felsorolás mutatja.

A Huntoken meglévő szabályai részben újra lettek implementálva, részben ki lettek hagyva (ez utóbbiak mind a tulajdonnév-felismerés vagy a morfológiai elemzés tárgykörébe sorolhatók), és készültek új szabályok is.
Az emToken a Flex helyett lexergenerátorként a Quexet[4] használja. A Quex mellett szól, hogy kifejezetten gyors véges állapotú automatát képes generálni, és hogy támogatja az UTF-8-as kódolású bemenetek feldolgozását is, többek közt a Unicode-karakterjellemzők használatával.[5]
Választható kimeneti formátum. Jelenleg az XML és a JSON támogatott, és tervbe van véve a TSV formátum implementálása.
A kimeneti címkekészletbe bekerült egy, a szóközjellegű karaktereknek megfelelő címke, az új tokenizáló ugyanis a detokenizálhatóság céljából megőrzi a szóközjellegű karaktereket is.

2. A rendszer áttekintése

2.1. Módszer

A tokenizálás a programozási nyelvekhez készült *fordítóprogram*okban, valamint *linterek*ben és *prettyprinter*ekben is a szöveges bemenet feldolgozásának szükséges fázisa. Az erre használt szoftvert *lexikai elemzőnek* vagy röviden *lexer*nek nevezik.[6] Bár a természetes nyelvek elemzése általában eltérő elvekre épül, mint a mesterségeseké, azért a természetes nyelvek tokenizálásánál sem szokatlan a lexerek használata.

[3] https://github.com/dlt rilmta/quntoken.
[4] http://quex.sourceforge.net/
[5] A Unicode karakterjellemzőkről bővebben l. [6].
[6] A lexikai elemzésről bővebben l. [3, 13 24. o.].

A lexereket nem szokás kézzel megírni, általában egy erre a célra szolgáló programmal generálják őket. Ezeknek *minta akció* párokat lehet megadni, ahol a *minta* egy reguláris kifejezés, az *akció* pedig annak a leírása, hogy mit csináljon a generált lexikai elemző, ha a minta illeszkedik. Az akciót vagy a generáló program saját kódkészletével kell megadni, vagy a generált kód nyelvén. A generáló program ezek alapján egy véges állapotú automata kódját állítja elő, amely reguláris nyelvek hatékony elemzésére képes (l. [1, 38-43. o.]).

A természetes nyelvek tokenizálását nem lehet könnyen egyetlen lexikai elemzésen belül elvégezni, ezért több lexikai elemző egymás után kötése lehet a megfelelő megoldás. Az emToken a Quex nevű lexergenerátorral készített lexikai elemzők egymás után kötött soraként gondolható el, ahol az egyik elemző kimenete a következő bemeneteként szolgál. Maga a Quex C++ nyelvű forráskódot generál, és az emToken túlnyomó részt szintén C++ nyelven készült keretrendszere az egyes elemzők összekötését és a köztük történő kommunikációt biztosítja.

2.2. Bemenet

A program bemenetéül teljesen annotálatlan, UTF-8 kódolású szöveg szolgál. A emToken nincs tekintettel a különböző (HTML vagy XML) címkékre, sem a HTML-karakterentitásként elkódolt karakterekre. Ilyen fájlok feldolgozása esetén ajánlott azokat előzőleg konvertálni sima szöveges állományokká.

A program bemenetével kapcsolatos még egy megszorítás. Megfelelő pontossággal működő tokenizáló nem készíthető el nyelvfüggetlen módon. Ennek oka, hogy a természetes nyelvek írásrendszerei jelentősen eltérnek abból a szempontból, hogy mi tekinthető bennük szó- vagy mondathatárnak. A magyar nyelv tokenizálását céljául tűző szoftvertől a nagyon eltérő írásmódok elemzése nem várható el. Az emToken a Unicode-karakterek közül csak a latin, a görög és a cirill ábécék betűit kívánja kezelni. Az ezen ábécéken kívüli betűket egy helyettesítő karakterre (U+FFFD REPLACEMENT CHARACTER,) cseréli le.

2.3. Kimenet

A program kimenete szintén UTF-8 kódolású szöveg. Jelenleg két formátum választható, az XML és a JSON.

Az emToken a következő címkekészletet használja a szöveg részeinek jelölésére:

s: Mondatok jelölése. Egy címke mindig csak egy mondathoz tartozik. (Két mondat között mindig vannak szóközjellegű karakterek.)

ws: Szóközök címkéje. A szóköz lehet mondatszintű címke is (mondatok között lévő szóközök), de lehet mondaton belüli is. A mondatokkal ellentétben a szóközcímkék bármennyi, egymás után következő szóközjellegű karaktert körülfoghatnak. Az egymást követő szóközjellegű karakterek akkor is egybe fognak tartozni, ha amúgy különböző karakterek (pl. sima szóköz és új sor karakter).

w: Szavak címkézésére szolgál. Mindig mondatokon belül található. Követheti szóköz vagy punktuáció.

c: Punktuációk jelölése. Az emToken igyekszik punktuációk bizonyos csoportjait egységként kezelni. Ez hasznos funkció a *smiley*-k esetében vagy a . . . -nál is. Más esetekben az írásjelek sorozata egyenként kerül feldolgozásra, azaz az egyes írásjelek külön-külön címkét kapnak, még ha folytatólagosan következnek is.

A emToken címkekészlete nagyban támaszkodik a Huntoken címkekészletére. Az egyetlen lényeges eltérés a `ws` címke, mivel a Huntoken sehogy nem jelöli (nem is őrzi meg) a szóközöket. Ennek az új címkének a bevezetése teszi lehetővé a detokenizálhatóságot.

2.4. API

A tokenizáló nem csak egy bináris állományként használható. A fordításkor létrejövő statikus könyvtáron és a megfelelő header állományon keresztül tetszőlegesen felhasználható más programok készítése során. Használatkor megadhatók a használandó elemzőmodulok, a bemenő szövegek forrása, a kimenet pedig vagy a standard kimenetre, vagy egy megadott `stringstream` objektumba íródik.

3. Elemzési rétegek

Az emTokenben több elemzési szint követi egymást. Az egyes rétegek közti kommunikáció céljára egy saját belső reprezentációt használtunk. Ebben az egyes címkéknek megfelelő jelek magában a szövegben, *inline* módon helyezkednek el. A hatékony feldolgozáshoz fontos volt, hogy az egyes címkéket olyan karakterekkel jelöljük, amelyek a szövegben nem fordulhatnak elő.

A Quexben írt minták esetén is a lexergenerátoroknál megszokott két elv érvényesül:

1. A leghosszabban illeszkedő mintához tartozó akció fog lefutni.
2. Két, egyforma hosszan illeszkedő minta közül a forrásban előbb szereplő fog lefutni.

A Quex-modulokban használt reguláris kifejezések értelmezésénél ezért sokszor nem elég maguknak a reguláris kifejezéseknek az értelmezése, de a többi mintát és ezek sorrendjét is érdemes figyelembe venni. A Quex nyújtotta lehetőségeket részletesen bemutatja [4].

A fejezet további részeiben az egyes Quex-modulokat ismertetjük, a feldolgozás sorrendjében.

3.1. Előfeldolgozás

Az előfeldolgozó modul feladata a bemenet ellenőrzése és szükség esetén tisztítása. Ez az érvénytelen karakterek cseréjét jelenti, illetve minden cserénél egy hibaüzenet küldését a standard hibakimenetre (*stderr*), ami tartalmazza az érvénytelen karaktert és annak helyét a bemenetben (sor- és oszlopszám).

3.2. Elválasztáskezelő

Ennek a rétegnek a használata opcionális, a `-d` parancssori kapcsoló megadásával lehet aktívvá tenni. Ekkor minden sor végi elválasztójel és az azt követő sorvége karakter törlődik, az elválasztott szavak így egyben fognak megjelenni. Ennek a modulnak a hatása detokenizálással nem fordítható vissza, illetve a modul nem biztosítja a kettős betűk megfelelő visszaállítását az elválasztás törlése után (pl. *sz-sz* $\rightarrow$*ssz*).

3.3. Mondatszegmentálás

A mondatra bontás technikailag két Quex-modulra lett felosztva. Az első felel egy nagy és összetett szabály alkalmazásával az alap mondatra bontásért, ami a legtöbb esetben elegendő. A második feladata a kivételek kezelése.

Az alap mondatra bontás során a következő szabályok érvényesülnek:

Mondat kezdete: A kérdőjel, a felkiáltójel és a szóközjellegű karakterek kivételével bármilyen karakter állhat mondatkezdő pozícióban.
Egy mondat tetszőlegesen sok szóközt tartalmazhat akár folytatólagosan is.
Új sor karakterből szintén tetszőleges számút tartalmazhat, de nem folytatólagosan (két egymást követő új sor karakter után mindig új mondat kezdődik).
Mondat vége: A mondat végén opcionálisan mondatzáró írásjel (pont, felkiáltójel, kérdőjel) lehet. A mondatzáró írásjeleknek nem kell tapadniuk, és az őket követő zárójelek és idézőjelek még az aktuális mondathoz tartoznak.
Nincs mondathatár mondatzáró írásjelek után az alább leírt pozíciókban:

- Ha a mondatzáró karakter után következő szó kisbetűvel kezdődik.
- Ha a mondatzáró karakter után vessző, pontosvessző, kettőspont vagy kötőjel következik.
- Ha a pontot közvetlenül egy szóalkotó karakter követi (például URL-ekben).
- Ha a mondaton belüli zárójelpár a záró tagja előtt mondatzáró írásjel is van (pl. *Péter születésnapjára (idén volt a 10.) Mari nem ment el.*).

A felismert mondatokat és mondatközi szóközöket a modul felcímkézve adja tovább a mondatrabontást korrigáló modulnak.

3.4. Mondatszegmentálás-korrekció

Ez az elemzési réteg végzi a mondathatárok korrigálását. Az előző modul a fenti szabályok által megengedett helyek mindegyikére beilleszti a mondatok nyitó és záró címkéit. Mivel olyan szabályszerű eset nincs, amiben a mondatszegmentáló modul nem tesz mondathatárt oda, ahova kellene, ezért az itt megfogalmazott mintáknak és eljárásoknak csak annyi a dolga, hogy a következő helyekről töröljék a mondathatárok címkéit:

mondatkezdő sorszámok után[7],
dátumok közben és dátumok után,
sorszámot követő paragrafusjel előtt,
pont és csillag között,
római számok után (kivéve *CD*),
monogram után,
pontra végződő rövidítések után.

A rövidítések azonosításához az emToken jelenleg a Huntoken eredeti rövidítéseit tartalmazó fájlnak az UTF-8-ra konvertált és kiegészített változatát használja. Tetszőleges, alternatív rövidítéslista is használható, ezt fordításkor kell a *Makefile*nak átadni. A rövidítéslistából a Quex-kódot egy python szkript hozza létre. Új rövidítéslista készítéséhez figyelembe kell venni, milyen formátumot vár a generáló szkript:

Egész sort kommentelni a # karakterrel lehet. Sor közbeni kommentelésre nincs lehetőség.
Minden rövidítés új sorba kell kerüljön.
Nem kell pont a rövidítések végére (ha van, nem baj, de nem fogja használni a program).
Minden kisbetűvel kezdődő rövidítés három változatban fog megjelenni a generált `abbrev.qx` állományban: az eredeti alakban, nagy kezdőbetűvel és csupa nagybetűvel. Ha egy rövidítés eredetileg is nagy kezdőbetűvel van megadva, akkor csak a csupa nagybetűs alakja fog pluszban létrejönni, és ha csak ez utóbbi volt eredetileg is megadva, akkor az `abbrev.qx` is csak ezt fogja tartalmazni.

Megjegyzendő, hogy a teljes feldolgozási láncban a mondatszegmentálás-korrigáló modul egymás után kétszer szerpel. Ennek az az oka, hogy az egymással átfedő szabályok közül egy „menetben" csak az egyik szabály fut le. Példának legyen két szabály: 1) ami illeszkedik az `a.b` sztringre, amiből `ab`-t csinál, és 2) ami illeszkedik a `b.c`-re, amiből `bc`-t csinál. Ekkor az `a.b.c` sztring feldolgozása során először lefut az 1)-es szabály, ami illeszkedik az `a.b`-re és az `ab.c`-t adja eredményül. Ezután az elemzés a `.c`-től folytatódik tovább, így a 2)-es szabály már nem tud illeszkedni, hiába van `b.c` részsztringje a bemenetnek. Ha másodszor is lefut az elemző, akkor az első szabály már nem fog illeszkedni, de a második igen, és előállítja a kívánt kimenetet, azaz `abc`-t. A mondatszegmentálás korrekciójakor a fenti példához hasonló szabályokról van szó, azaz egy karakterlánc egyes részláncait a felesleges mondathatárokat kell bizonyos pozíciókból törölni.[8]

[7] Sorszámok után már az alap mondatszegmentálást végző modul sem tesz mondat határt, ha utána kisbetűvel vagy írásjellel folytatódik a mondat. A mondatkezdő sorszámok esetében viszont nagybetűs folytatásnál sincs mondathatár (pl. *1. Fejezet*).

[8] A Huntoken is a kétszer futtatást választotta a probléma kiküszöbölésére.

3.5. Tokenizáló

Ez a modul a bemeneti szöveg további, a megállapított mondathatárokon belüli szegmentálását végzi. A modul a felcímkézett mondatközi szóközöket változtatás nélkül adja vissza, míg a címkézetlen, mondaton belüli szóközsorozatokat felcímkézi. Hasonlóan könnyű a punktuációk kezelése is. A csoportosítható írásjelek csoportosan lesznek felcímkézve, az egyedülállók ha nem részei egy hosszabban illeszkedő mintának pedig egyenként.

A szavak felismeréséért felelős alapszabály a következőkre van tekintettel:

> Zárójelek kezelése. Egy zárójelpár a szó részének tekinthető, ha a pár legalább egyik tagja a szó belsejében található. Máskülönben a zárójelek különálló írásjelnek számítanak. Példák XML-annotációval: `<w>záró(jel)</w>`, `<w>(záró)jel</w>` és természetesen `<w>zár(ó)jel</w>`, de `<c>(</c><w>zárójel</w><c>)</c>`.
>
> Szó belsejében lehet pont, ez nem okozhat szótörést, pl. `<w>R.I.P.</w>`, de alapesetben a közvetlenül a szó előtt vagy után található pont is a szóhoz tartozik.
>
> Szóvégi pont nem tartozik a szóhoz, ha mondatzáró címke előtt található, kivéve, ha rövidítésről van szó. Pl. `...<w>vége</w><c>.</c></s>`, de `...<w>stb.</w></s>`
>
> A szóhoz kötőjellel kapcsolódó *-e* partikula mindig külön tokenként lesz címkézve, pl. `<w>van</w><w>-e</w>`.

Ezen alapszabályt egészíti ki még pár, kivételekre vonatkozó szabály. Ezek lényege, hogy bizonyos körülmények között akkor is egyben tartanak egy szót, ha az alapszabály értelmében azt több tokenre kéne bontani. A kivételek kezelésének elvi alapja hasonló a mondathatárok korrekciójához. Olyan alapszabályt kell készíteni, ami mindenhol töri a karaktersorozatot, ahol kell, de törheti ott is, ahol nem kell. Ha ez a feltétel teljesül, akkor csak olyan kivételek lesznek, ahol nem kell több sorozatra bontani a bemenetet. Az ilyen eseteket leíró minták biztos, hogy hosszabban fognak illeszkedni, mint az alapszabályban megfogalmazott minta. A lexergenerátorok és köztük a Quex garantálják, hogy a konkurens szabályok közül a hosszabban illeszkedő fog lefutni és ez épp az, amire a kivételek kezelésekor szükség van.

Az alábbi kivételes eseteket kezeli szavakként (vagyis látja el `w` címkével, és tartja egyben) az emToken:

> informatikai kifejezések: e-mail cím, URL, fájlnevek helyettesítő karakterrel, elérési utak;
>
> szavak *et* jellel (pl. *AT&T*);
>
> szavak aposztróffal;
>
> számok ponttal, vesszővel vagy spáciummal tagolva;
>
> tizedes törtek;
>
> zárójeles felsorolások, pl. *b*).

3.6. Konverterek

Az emToken két, választható kimeneti formátumát is két Quex-modul hozza létre. Ezek a tokenizáló által használt belső reprezentációt konvertálják a XML vagy JSON formátumra.

4. Tesztelés

Egy természetes nyelvi tokenizálónak nagyon sok követelményt kell egyszerre teljesítenie, sok szabályt tartamaz, melyeknek a kivételeit is helyesen kell kezelnie. Ekkora szabálytömeget nem nagyon lehet egyidejűleg fejben tartani, a tesztelés itt a szokottnál is fontosabb. Éppen ezért az emToken automatikus tesztelést biztosít a fejlesztéshez. Amikor új fordítás készül, a *Makefile* automatikusan lefuttatja az emToken teljeskörű tesztelését is. Ez azonnali visszajelzést ad a fejlesztőnek, hogy egy új szabály implementálása vagy egy régebbi módosítása milyen hatással volt a rendszerre, okoz-e hibát valahol, vagy sem.

A teszteléshez a Google Teszt C++-os keretrendszert használtuk.[9] A fordítás során a Makefile-nak megadható a teszteseteket tartalmazó fájlok halmaza, ezáltal könnyen lehet alternatív szabályokhoz alternatív teszteseteket rendelni.

5. Kiértékelés

A tokenizáló teljesítményének kiértékeléséhez a Szeged Korpusz 2.0-t [2] vettük alapul. Ami a mondatszegmentálást illeti, itt 81.648 mondatból 2.131 esetben hibázott az emToken, ami 97,39%-os pontosságot jelent. A hibák jelentős része abból ered, hogy az emToken (akárcsak a Huntoken), nem kezd új mondatot, ha a mondatzáró írásjel után kisbetűvel folytatódik a karakterlánc, szemben a Szeged Korpusszal, ahol ez gyakran előfordul.

A tokenizálásnál *word accuracy*-t mértünk, amire 99,27%-ot kaptunk (1.478.300 tokenre összesítve 10.903 hibás token jutott). Ezen hibák egy része a tokenizálási sémák különbségéből fakad (pl. az emToken a szóközökkel tagolt számokat igyekszik egyben tartani), másik része tényleges hiba.

6. Tervek

Az emToken jelenleg egy szálon fut. Ennek velejárója, hogy bármely elemzőmodul futásának feltétele, hogy az előtte lévő modulok már befejezzék a munkájukat. A tokenizáló sebességét a program többszálúsításával tervezzük továbbfejleszteni.

[9] https://github.com/google/googletest

7. Köszönetnyilvánítás

A tokenizáló elkészítésében Miháltz Márton nyújtott folyamatos szakmai segítséget. Köszönet érte. Továbbá köszönet illeti Simon Esztert a kiértékelésben és Nemeskey Dávidot a tesztelésben nyújtott segítségéért.

Az `e-magyar` eszközlánc és benne az emToken tokenizáló az MTA 2015. évi Infrastruktúra-fejlesztési Pályázat 2. kategóriájában elnyert támogatás segítségével valósult meg.

Hivatkozások

1. Bach, I.: Formális nyelvek. Typotex (2005)
2. Csendes, D., Csirik, J., Gyimóthy, T., Kocsor, A.: The Szeged Treebank. In: Lecture Notes in Computer Science: Text, Speech and Dialogue. pp. 123–131. Springer (2005)
3. Csörnyei, Z.: Fordítóprogramok. Typotex (2006)
4. Schäfer, F.R.: The Quex Manual (0.65.6) (2015), elérhető: http://quex.sourceforge.net/doc/html/main.html (letöltés: 2017. január 2.).
5. Váradi, T., Simon, E., Sass, B., Gerőcs, M., Mittelholcz, I., Novák, A., Indig, B., Prószéky, G., Farkas, R., Vincze, V.: `e magyar`: digitális nyelvfeldolgozó rendszer. In: XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017). p. (jelen kötetben). Szeged (2017)
6. Whistler, K., Freytag, A.: The unicode character property model. Tech. rep., The Unicode Consortium (2015), elérhető: http://www.unicode.org/reports/tr23/tr23 11.html (letöltés: 2017. január 2.).

Az emMorph morfológiai elemző annotációs formalizmusa

Novák Attila[1,2], Rebrus Péter[3], Ludányi Zsófia[3]

[1] MTA-PPKE Magyar Nyelvtechnológiai Kutatócsoport,
[2] Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai Kar
1083 Budapest, Práter utca 50/a, e-mail:novak.attila@itk.ppke.hu
[3] MTA Nyelvtudományi Intézet
1068 Budapest, Benczúr utca 33
e-mail:{rebrus.peter, ludanyi.zsofia}@nytud.mta.hu

Kivonat A morfológiai elemző – lévén minden nyelvfeldolgozási lánc kezdeti lépése – a nyelvtechnológiai alkalmazásokban kiemelkedő szerepű. A kimenet értelmezése szempontjából rendkívül fontos a morfológiai elemzés kimenetének egységesítése. Cikkünkben az *emMorph* morfológiai elemzőrendszer és az *emLem* lemmatizáló implementációjának ismertetése után bemutatjuk ezen eszközök kimeneti formalizmusát, különös tekintettel a morfológiai címkékre.

1. Bevezetés

A *Nyílt, integrált magyar nyelvtechnológiai kutatási infrastruktúra fejlesztése* projekt (`e-magyar`) az MTA Nyelvtudományi Intézete vezetésével, az MTA SZTAKI, a SZTE, a PPKE és az AITIA International Zrt. közreműködésével valósult meg[4]. Célja egy olyan nyílt forrású, szabadon hozzáférhető nyelvtechnológiai infrastruktúra kiépítése volt, melynek elemei a magyar nyelv gépi elemzésének alapvető eszközeit tartalmazzák integrált, szabványos keretben [11]. A rendszer részét képezi egy új magyar morfológiai elemző, amelynek implementációja a nyílt forráskódú véges állapotú transzducertechnológiát alkalmazó hfst rendszer felhasználásával valósult meg. Jelen cikk célja a megvalósult *emMorph* morfológiai elemző[5] és az arra épülő *emLem* lemmatizáló[6] implementációjának ismertetése és az elemző, illetve a lemmatizáló kimeneti formalizmusának bemutatása, különös tekintettel a morfoszintaktikai címkékre.

2. A morfológiai elemző implementációja

A morfológiai elemző adatbázisa elsősorban az eredetileg a Humor morfológiai elemző motorhoz [8] készült magyar morfológiai adatbázison alapul [5], amelyet

[4] `http://e-magyar.hu`
[5] `https://github.com/dlt-rilmta/emMorph`
[6] `https://github.com/dlt-rilmta/hunlp-GATE/tree/master/Lang_Hungarian/resources/hfst/hfst-wrapper`

kiegészítettünk olyan szavakkal, amelyek az eredeti Humor leírásban nem, a *morphdb.hu* [10] adatbázisban viszont szerepeltek, miután az utóbbi listából kiszűrtük a hibás, illetve elhanyagolhatóan ritka szavakat. A morfológiai leírást kezelő keretrendszer egy procedurális szabályrendszer felhasználásával magas szintű és redundanciamentes morfémaleírásokból állítja elő az egyes morfémák lehetséges allomorfjait, azok tulajdonságait (jegyeit) és azokat a jegyalapú megszorításokat, amelyeknek az egymással szomszédos morfok között teljesülnie kell. Emellett a helyes szószerkezetek leírását egy kiterjesztett véges állapotú szónyelvtan-automata ábrázolja.

Az eredeti Humor elemzőprogram ezeket az allomorflexikonokat, az allomorfok közötti szomszédossági megszorításokat és a véges állapotú szónyelvtan-automatát közvetlenül használja a szóalakok elemzése közben. Az új hfst-alapú implementációban [3] mindezek az adatszerkezetek egyetlen véges állapotú transzducerben jelennek meg.

A véges állapotú transzduceren alapuló morfológiai rendszerek létrehozásánál általában az a szokásos eljárás, hogy a *lexc* lexikondefiníciós nyelv [1] segítségével létrehoznak egy alap-morfémalexikont, amelyben a morfémák valamiféle mögöttes reprezentációban szerepelnek, és a leírás e mellett tartalmaz egy az *xfst* újraírószabály-formalizmusa [1] segítségével megadott vagy a Kimmo-féle kétszintű megszorításokon alapuló szabálykomponenst, amelyet a mögöttes alakokat tartalmazó lexikonnal komponálva előáll a morfémák mögöttes és felszíni alakjai közötti, az adott kontextusban megfelelő leképezés. A hagyományos megközelítésben tehát a *lexc* lexikon és az *xfst* szabályrendszer kompozíciója hozza létre a morfológiai elemző transzducert.

Az általunk készített véges állapotú magyar morfológiai leírás ezzel szemben nem tartalmaz külön sem *xfst* újraíró szabályokat, sem Kimmo-féle kétszintű megszorításokat tartalmazó szabálykomponenst, hanem a morfémák allomorfjait és a hozzájuk tartozó szomszédossági megszorításokat folytatási osztályok formájában tartalmazó adatbázist közvetlenül egy a lexc formalizmus segítségével leírt lexikonná konvertáljuk, amely a mögöttes alakok (lemmák) és a felszíni alakok közötti helyes leképezést már tartalmazza, így további szabályokra nincs szükség. Az eredeti Humor formalizmus szónyelvtan-automatáját a véges állapotú leírásban a *flag diacritcs* konstrukció [1] alkalmazásával ábrázoltuk. Ez a leírás tartalmazza a morfémák közötti nem lokális megszorításokat is (pl. hogy a melléknevek felsőfokát jelölő prefixumot a szón belül valahol vagy egy középfokjelnek vagy valamilyen más felsőfokjelet engedélyező morfémának követnie kell). A Humor formalizmusban leírt adatbázis véges állapotú leírássá konvertálására alkalmazott algoritmusok részletes leírását l. [7] 6. fejezetében, illetve itt: [6].

3. Lemmatizálás

A morfológia az összetett és képzett szavak esetében az összetételi tagokat, illetve a képzőket is azonosítja. Amennyiben az összetett vagy képzett szó a lexikonba egyben is fel van véve, több elemzés is kijöhet, amelyek különböző részletességű elemzését adják az adott szónak. A *fejetlenség* főnév elemzésekor például

az elemző ezt egyben is megtalálja, ugyanakkor visszavezeti a *fejetlen* melléknévre, a *fej* főnévre és a *fej* igére is. Bár ezek az elemzések részben különböző szemantikai tartalmakat tükrözhetnek *(káosz, átgondolatlanság, fejnélküliség, a fejés elmaradása)*, ezek közül a jelentések közül némelyik szinte egyáltalán nem jelenik meg ténylegesen előforduló szövegekben, ráadásul a morfológiai elemzésre épülő és a nyelvi elemzés egyéb szintjeit végző eszközöknek általában nincs is szükségük ilyen részletességű elemzésre. Amire viszont szükségük van, az az adott szó lemmája (szótári töve), valamint (elsősorban a ragok, illetve bizonyos nagyon produktív képzők, pl. az igenévképzők által megtestesített) morfoszintaktikai jegyei. A lemma magában foglalja a szóban levő töveket és képzőket, mindazt, amit nem morfoszintaktikai jegyek formájában szeretnénk a további nyelvi elemzést végző eszközök számára továbbadni.

A hfst rendszer [3] morfológiai elemzést végző eszközei (a *hfst-lookup*, illetve a *hfst-optimized-lookup*) alapesetben nem olyan elemzést állítanak elő, amely közvetlenül alkalmas lenne a lemma előállítására, ugyanis kizárólag az adott elemzést alkotó morfémák mögöttes alakját és a morfoszintaktikai címkéket adják vissza, az ezeknek megfelelő felszíni alakot nem, így a képzőt tartalmazó tövek teljes szótári alakja nem mindig számítható ki. A hfst-lookup fejlesztője kérésünkre kiegészítette az eszközt egy olyan funkcióval, amely az elemzett szót alkotó morfémák felszíni és mögöttes alakját egyszerre adja vissza (illetve ténylegesen működőképessé tette ezt a korábban nem működő funkciót). Ugyan ez a kimenet emberi fogyasztásra nem igazán alkalmas[7], de lehetővé tette, hogy ennek felhasználásával létrehozzuk a morfológiai elemző kimenetére épülő Java nyelven implementált, ezért platformfüggetlen lemmatizáló eszközt (emLem), amely a tőalkotó elemek (tövek, képzők) összevonásával kiszámolja az adott elemzéshez tartozó lemmát (ehhez az utolsó tőalkotó elem kivételével a felszíni alakra van szükség), annak eredő szófaját, és ehhez hozzáadja a nem tőalkotó morfémák által hordozott morfoszintaktikai jegyek címkéit.

Az azonos lemmát, szófajt és egyéb morfoszintaktikaicímke-sorozatot eredményező különböző részletességű elemzések (pl. a *fejetlenség* főnév elemzései) a lemmatizáló kimenetén egyetlen elemzésként jelenhetnek meg, hiszen ezek a magasabb nyelvi szinteket feldolgozó elemzők számára (szófaji egyértelműsítő, szintaktikai elemző stb.) ekvivalensek. Ugyanakkor a lemmatizáló képes a részletes elemzések visszaadására is úgy, hogy az az elemzést alkotó morfok felszíni alakját is tartalmazza olvasható és jól kereshető formában[8]. A lemmatizáló viszonylag bonyolult algoritmust valósít meg, amely nem triviális morfológiai konstrukciók (pl. ikerszavak) és különleges beállítások (pl. ha az igenévképzőket nem tekintjük tőalkotónak) esetén is helyes lemmát ad.[9] Az alkalmazott lemmatizáló algoritmussal kapcsolatos további részletek [7] 4.3 fejeztében olvashatók.

[7] t:t e:e h:h e:é n:n :[/N] e:e c:c s:s k:k é:e :[_Dim:cskA/N] j:j é:e :[Poss.3Sg] t:t :[Acc]

[8] tehén[/N]=tehen+ecske[_Dim:cskA/N]=ecské+je[Poss.3Sg]=jé+t[Acc]=t

[9] Léteznek igenévképzőt tartalmazó alaktani konstrukciók, amelyekre hibás tövet kapunk, ha az igenévképző(vel azonos alakú képző)t nem tekintjük a tő részének: pl. *húsdarál(ó)*.

4. Kiértékelés

A morfológia elemző fedésével kapcsolatban Kornai András és kollégái készítettek független kiértékelést az elemző 2016 augusztusi verziójával. Bár ezen cikk célja elsősorban az elemző által generált annotáció ismertetése, itt röviden bemutatjuk ennek a kiértékelésnek az eredményét. A kiértékeléshez két nagyméretű magyar nyelvű korpuszt, az MNSZ2-t (Magyar Nemzeti Szövegtár V2.0[10]) és a WebKorpusz 2.0-t (WK2[11]) használták. A korpuszokból azokat a szavakat választották ki, amelyek legalább három MNSZ2-részkorpuszban szerepeltek, és a WebKorpuszban is legalább háromszor előfordultak. A kiválasztott 1363692 szóalak az MNSZ2 95,65%-át és a WK2 94,66%-át fedi le. A kiválasztás során a két korpusz tokenjeinek 5,12%-a esett ki. A tesztanyagból az elemző által felismert szóalakok korpusztokenekre visszavetített aránya 92,63%, a nem elemzetteké 2,25%. Kornaiék ezt az fedést „kiemelkedően jó"-nak minősítették.[12]

5. A morfológiai elemző által generált annotáció

5.1. Motiváció

A morfológiai elemzés kimenetének egységesítése rendkívül fontos a kimenet értelmezése szempontjából, legyen az elemzés automatikus vagy nyelvészeti alapú, és a kimenet feldolgozása automatizált vagy emberi erővel történő. Az ilyen kimeneti annotációs rendszerekben a morfológiai elemzők tipikusan kétfajta információt jeleníthetnek meg: morfológiait és morfoszintaktikait. A morfoszintaktikai információ megadja, hogy az adott szóalak milyen szintaktikai környezetben és funkcióban fordulhat elő, előre megadott morfoszintaktikai tulajdonságokhoz rendelt értékek használatával. A morfológiai információ megmutatja, hogy mely morfémaváltozatokból (morfokból) áll össze a szó, és ezekhez a morfokhoz mely morfoszintaktikai jegyek rendelhetők. E két információtípust tipikusan egyszerre szokták az annotációs rendszerek megjeleníteni, de különböző rendszerek különböző arányban. A két szélsőség egyikét a nyelvészeti morfo(fono)lógiai elemzés képviseli, ahol az explicite nem megjelenő morfoszintaktikai információk nem lényegesek (hiányozhatnak), viszont a morfokra való szegmentálás általában központi jelentőségű. Ezekkel szemben állnak azok a formális annotációs rendszerek, amelyekben csak morfoszintaktikai jegyek vannak, és az annotáció nem tartalmaz a morfszegmentálásra vonatkozó információt (ez utóbbira példa az ún. Universal Dependencies [4], az MSD-kódolás vagy a hunmorph rendszerben működő ún. KR-kódolás [9]). Több rendszerben a kétféle információt az annotáció egyszerre tartalmazza (pl. ilyen a Humor [5,8] vagy a Xerox magyar morfológiai elemzője), de ezek megjelenítése sokszor némileg ad hoc módon történik.

[10] `http://mnsz.nytud.hu`

[11] `http://mokk.bme.hu/en/resources/webcorpus`

[12] A jelenlegi verzió az itt ismertetettnél jobb fedést mutat, mert egy jelentős hibaosztály (Kornaiék a kötőjeles szavak egy nagy osztályára nem kaptak elemzést) megszűnt.

Ennek praktikus okai vannak: az írott szóalakok szegmentálása bizonyos esetekben szükségszerűen önkényes: pl. a *hússzal* szóalak morfokra bontásakor a *húsz* tő és a *szal* eszközhatározó-rag közötti határ meghúzása a helyesírás sajátosságai miatt sehogy sem lesz igazán jó. A Humor rendszerben használt *hússz+al* tagolás mellett praktikus (a lexikonmérettel és a jegyrendszer komplexitásával kapcsolatos) szempontok szólnak: a kétjegyű betűre végződő szavakhoz mindenképp elő kell állítani egy-egy plusz allomorfot, ugyanakkor az ezekhez kapcsolódó eszközhatározórag-allomorfból ebben az esetben elég, ha egy van a lexikonban.

Az emMorph elemző kimeneti formalizmusa kialakításakor abból indultunk ki, hogy az egyszerre kell szolgálja a számítógépes nyelvfeldolgozást és a nyelvészeti elemző munkát. Ennek megfelelően igyekeztünk arra törekedni, hogy az annotáció mind a releváns morfológiai szegmentálást, mind a szükséges morfoszintaktikai jegyeket tükrözze, és belőle ezek külön-külön is kinyerhetők legyenek. Ugyanakkor mivel az elemző alapvetően a Humor rendszer számára implementált szabályrendszeren alapszik, a szegmentálás tekintetében megmaradt néhány a Humor leírásból örökölt kompromisszum. Egy másik megszorítás az volt, hogy szerettük volna a korábban használt annotációs sémák és az új rendszer közötti konverziót lehetőleg minél teljesebb mértékben lehetővé tenni. Ezért azokat a komplex toldalékokat, amelyekhez tartozó címke a korábbi rendszerek valamelyikében nem tagolódott világosan elkülöníthető elemekre (pl. az *-i* „birtoktöbbesítő jel”-et tartalmazó birtokos végződések), nem szegmentáltuk szét különálló elemekre az új annotációs sémában sem, hanem azokat a fúziós morfémáknak megfelelő módon ábrázoltuk (l. a 5.5 részt).

Az annotációs rendszer egyben szabványosítási javaslat a magyar nyelvű automatikus morfológiai elemzők kimeneti formátumára, és a magyar alaktan nyelvészeti glosszáinak formátumára. A korábbi magyar morfológiai elemzők egyedi és mind egymástól, mind az esetleges nemzetközi szabványoktól eltérő címkéket használtak. A projekt keretében megvalósult elemző címkekészletét ezzel szemben igyekeztünk nemzetközi szabványhoz igazítani: amennyire lehetséges volt, a nyelvészeti annotációra széles körben egyfajta szabványként használt Leipzig Glossing Rules (LGR) [2] javaslatait követtük. A címkék meghatározásakor emellett az ott leírtakat kiegészítő lényegesen kibővített listára (List of glossing abbreviations = LOGA)[13] támaszkodtunk, amelyet az ezekben a dokumentumokban leírtak szellemében kiegészítettünk a hiányzó (elsősorban képzőkkel kapcsolatos) címkékkel.

5.2. Az annotáció felépítése

Míg a Leipzig Glossing Rulesban javasolt annotációs séma szerint az annotáció külön sorokban tartalmazza a morfokra szegmentált elemzett alakot és a morfokhoz tartozó morfoszintaktikai jegyeket (amely csak a tövek esetén tartalmaz alaki információt: a lemmát), a véges állapotú morfológiai elemző kimenetén ezek az elemek szekvenciálisan jelennek meg: az egyes morfok mögöttes és felszíni alakja, illetve a hozzá tartozó morfoszintaktikai címke együtt jelenik meg

[13] `https://en.wikipedia.org/wiki/List_of_glossing_abbreviations`

a kimeneten. A szegmentálás jelölésére a Leipzig Glossing Rulesban a kötőjel használatát javasolják. Ennek használata – tekintettel arra, hogy a sztenderd helyesírásban ez igen gyakran eleve a szóalak része – nem lett volna praktikus.[14] Ehelyett az elemző kimenetén szögletes zárójelbe tett morfoszintaktikai címkék jelölik implicit módon a szegmentálási határokat. A Leipzig Glossing Rulesban javasolt gyakorlattól még abban a fontos kérdésben tértünk el, hogy az LGR-t követő kiadványokban – némileg meglepő módon – gyakran egyáltalán nem használnak szófajcímkéket: a tövek szófaját semmilyen módon nem jelölik. Hogy ennek a gyakorlatnak mi az oka, azt nem érdemes találgatni, mi mindenesetre nem követtük.

Az emMorphban használt annotációban a címkék egyes alaki tulajdonságai egyértelmű összefüggésben vannak az adott morféma típusával. A tőmorfémák címkéje /-lel kezdődik (`fej[/N]` főnév), a képzőké _-sal, és a képző címkéjét követő / után a képző eredő szófaja áll (`etlen[_Abe/Adj]` névszói fosztóképző „abesszívusz"), az inflexiók címkéje pedig nem tartalmaz speciális karaktert (`t[Acc]` tárgyesetrag). A szófajcímkék elé helyezett / a morphdb.hu-ban használt KR-kódrendszerből származik, a képzők _-sal való megjelölése pedig a Humor-kódkészlet sajátossága volt.

További eltérés az LGR-hez képest, hogy az emMorph kimenete a toldalékmorfok lexikai alakjait is tartalmazza. Ez nem valamiféle absztrakt fonológiai alak, hanem azzal az allomorffal azonos, amelyet az adott toldalékmorféma akkor vesz fel, amikor a szó végén áll. Ennek elsősorban a képzők esetében van jelentősége és a lemmatizáláshoz szükséges. Az emMorphra épülő emLem lemmatizáló az adott elemzéshez tartozó lemma kiszámolásakor azt a tőalkotó morfokból állítja össze. Az utolsó tőalkotó elem a lexikai, a többi a felszíni alakjában szerepel a lemmában (1. táblázat).

surface form	**butá**	**cská**	**bb**	**já**	**tól**	**nadrág**	**ocská**	**tól**
lexical form (lemma)	*buta*	*cska*	*bb*	*ja*	*tól*	*nadrág*	*ocska*	*tól*
`abstract lex. form`	`buta`	`LVcskA`	`LAObb`	`LjA`	`LtŐl`	`nadrág`	`LVcskA`	`LtŐl`
tag	/Adj	_Dim/Adj	_Comp/Adj	Poss.3Sg	Abl	/N	_Dim/N	Abl
lemma 1	**butá**	**cská**	*bb*					
lemma 2	**butá**	*cska*				**nadrág**	*ocska*	
lemma 3	*buta*					*nadrág*		

1. táblázat. Képzett és ragozott szavak lemmatizálása

5.3. Szegmentálás és alternációk

A kötőhangzót általában az utána álló toldalékhoz kapcsoljuk:
`nap[/N]ok[Pl]at[Acc]`. Az epentetikus mássalhangzókat ezzel szemben (pl. *bőv+en, ven+ne*) általában a tőhöz számítjuk.

A morfsorozat az aktuális alakban szereplő tőallomorf részsztringjeit tartalmazza. A lemma neve viszont általában a paradigma alapalakja, mely az izoláltan

[14] Az LGR formalizmusát eleinte elsősorban a helyesírási normával nem rendelkező „bennszülött" nyelvekkel kapcsolatos terepmunkagyűjtések eredményének lejegyzésére használták.

megjelenő alakkal azonos (ha ez létezik). Váltakozó tő esetén a tőallomorf nem mindig egyezik meg a lemma nevével: pl. *fá-* ∼ *fa, bokr-* ∼ *bokor, tav-* ∼ *tó, nyar-* ∼ *nyár, ve-* ∼ *vesz, vol-* ∼ *van.* Az ikes igék esetén az alapalak (és így a lemma neve) az ikes alak, függetlenül attól, milyen tőváltozat jelenik meg a szóban forgó alakban: *laktok:* `lakik[/V]tok[Prs.NDef.2Pl]`.

Ha az alapalak is több alakban jelenhet meg (mint az sz∼d váltakozást mutató igéknél), akkor a gyakoribb alakot vesszük lemmának – az, hogy ez melyik, az egységes lemmaazonosíthatóság miatt előre rögzíteni kell minden egyes ilyen lemmánál: *növekednek:* `növekszik[/V]nek[Prs.NDef.3Pl]`.

5.4. Hiányos és helyettesítő paradigmák

Ha egy morfológiailag hiányos paradigmájú elem alapalakja hiányzik, akkor a lemma neve a morfológiailag legjelöletlenebb alak. Plurale tantum (pl. *üzelmek, bélbolyhok, légutak*) esetén ez a nem birtokos nominativusi többes számú alak. Possessivum tantum (pl. *eleje, alja, hóna, öccse*) esetén a lemma neve az egyes számú E.3 birtokos nominativusi alak. Egyes esetekben a kétféle defektivitás egyszerre érvényesül (pl. *eleik, feleink*), ekkor a lemma a többes számú E.3 birtokos alak: *eleiknek* `elei[/N]ik[Pl.Poss.3Pl]nek[Dat]`.

Az igei defektivitás azon eseteinél, ahol nem áll rendelkezésre a jelen idő kijelentő mód indefinit E.3 alak (pl. *sínyli, kétli*), akkor a definit E.3 kijelentő mód jelen idejű alak lesz a lemma neve: *sínylitek:* `sínyli[/V]itek[Prs.Def.2Pl]`.

5.5. Fúziós morfémák

Ha egy morfhoz több jegyet kell rendelni (fúziós morféma), akkor a szóban forgó jegyek egy []-en belül jelennek meg, és ponttal választjuk el őket. Például egyes birtokosjelölős alakokban a toldalék egyszerre utal a birtoklásra (Poss) és a birtok számára/személyére (pl. 1Sg): *nadrágomat* `nadrág[/N]om[Poss.1Sg]at[Acc]`. Az elemzések Humor-elemzésekre és címkékre való leképezhetősége érdekében így jártunk el néhány olyan toldalék esetében is, amelyek esetében a szegmentálás egyébként nem lenne lehetetlen (bár bizonyos dilemmák felmerülnének): (`jaim[Pl.Poss.1Sg]`, `nátok[Cond.Def.2Pl]`, `nátok[Cond.NDef.2Pl]`, `tatok[Pst.NDef.2Pl]`, `tátok[Pst.Def.2Pl]`). A zérusmorfok jelölése nem különleges, egyszerűen üres a felszíni alakjuk (és általában a lexikai is).

Az igeidőt és a módot egymással komplementáris viszonyban levőnek tekintettük, így külön kijelentő mód jegyet nem vettünk fel, hanem valamely időjegy (`Prs`, `Pst`) meglétéből következik a kijelentő mód.

5.6. Unáris jegyek

Vannak olyan morfoszintaktikai dimenziók, amelyeknek csak egy értéke jelenik meg – ezek az ún. unáris jegyek. Azt az információt, hogy ilyen értékkel az alak nem rendelkezik, az annotáció nem jelöli (pontosabban az adott jegy hiányával jelöli). A modális igei alakokban (pl. *adhatsz* `ad[/V]hat[_Mod/V]sz[Prs.NDef.2Sg]`)

unáris jegy áll, ahogyan az összes képzett alakban is. Ezzel szemben az inflexiós jegyek nagy része nem unáris, például az igeragozás definitsége tekintetében az `Def` jegy szemben ál az `NDef` jeggyel, az alanyesetet is megjelöljük a `Nom` jeggyel. A jelen implementációban sajátos kivételként a névszóragozás paradigmájának leírásában az egyes szám jelöletlenül maradt. Ennek oka az volt, hogy a morfokra szegmentálás szempontjából ennek a jegynek mind a tőhöz, mind a toldalékokhoz rendelése ellentmondáshoz vezetett volna.

5.7. Az alkalmazott címkék

Mint korábban említettük, az elemzőben igyekeztünk következetesen az LGR és a LOGA dokumentumokban felsorolt címkéket használni, illetve az ott megadott alternatív jelölések közül választani. Azon címkék ügyében szavazással döntöttünk, amelyekkel kapcsolatban az előkészítő fázisban nem jutottunk konszenzusra. Így született többek között az igekötők `/Prev` (preverb), a igenevek `Ptcp` a névelők `Det`, a melléknevek, illetve a számnevek `Adj`, illetve `Num` címkéje. Az alkategóriára utaló jegyek a címkén belül |-lal elválasztva jelennek meg, pl. `/Adj|Pro|Int`: melléknévi kérdő névmás (pl. *milyen*). Zárójelben szerepel a vonzatos névutók vonzatát jelölő esetrag kódja: `/Post|(Abl)`. A (szinte) azonos funkciót nem fonológiailag vagy lexikailag kondicionált módon, hanem lényegében szabadon választhatóan különböző formában kifejező toldalékok esetében a funkció mellett a formára is utal a használt címke (a formára utaló címkerész előtt mindig kettőspont áll): `EssFor:ként, EssFor:képp, EssFor:képpen`, illetve `_Adjz_Type:fajta/Adj, _Adjz_Type:forma/Adj, _Adjz_Type:féle/Adj, _Adjz_Type:szerű/Adj` (Adjz: adjectivizer 'melléknévképző'). A képzők esetében a formára sokszor egyébként is utalunk. Sőt, időnként – amikor a funkció viszonylag heterogén, illetve nem volt egyszerű egy rövid címkében egyértelműen megnevezni – csak a formára (és az eredő szófajra) utal a címke: `_Adjz:i/Adj, _Adjz:s/Adj, _Adjz:Ő/Adj, _Adjz:Ű/Adj`.

6. Konklúzió

A cikkben bemutattuk az `e-magyar` projekt keretében megvalósult új, nyílt forráskódú morfológiai elemzőeszközt. Kitértünk a lemmatizáló és a morfológiai elemző implementációjának főbb kérdéseire, majd részletesen ismertettük a nyílt forráskódú emMorph morfológiai elemző és emLem lemmatizáló kimeneti formalizmusát, az általuk generált annotációt. Az emMorph által generált annotáció formalizmusa sztenderdizált, automatikus és kézi feldolgozásra is alkalmas. A jegyek elnevezése (rövidítése) és sorrendje a nemzetközi nyelvészeti konvenciókhoz kötődik, így jól olvasható, és a nyelv ismerete nélkül is értelmezhető.

7. Köszönetnyilvánítás

Az `e-magyar` eszközlánc az MTA 2015. évi Infrastruktúra-fejlesztési Pályázat 2. kategóriájában elnyert támogatás segítségével valósult meg. Köszönetet mon-

dunk Kornai Andrásnak és kollégáinak az elemző fedésének a 4. részben ismertetett kiértékelésért.

Hivatkozások

1. Beesley, K., Karttunen, L.: Finite State Morphology. No. 1 in CSLI studies in computational linguistics: Center for the Study of Language and Information, CSLI Publications (2003)
2. Comrie, B., Haspelmath, M., Bickel, B.: The Leipzig glossing rules: Conventions for interlinear morpheme-by-morpheme glosses (2008), `https://www.eva.mpg.de/lingua/pdf/Glossing-Rules.pdf`
3. Lindén, K., Silfverberg, M., Pirinen, T.: HFST tools for morphology – an efficient open-source package for construction of morphological analyzers. In: Mahlow, C., Piotrowski, M. (eds.) State of the Art in Computational Morphology, Communications in Computer and Information Science, vol. 41, pp. 28–47. Springer Berlin Heidelberg (2009)
4. McDonald, R., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., Hall, K., Petrov, S., Zhang, H., Täckström, O., Bedini, C., Bertomeu Castelló, N., Lee, J.: Universal dependency annotation for multilingual parsing. In: Proceedings of ACL 2013. pp. 92–97. Association for Computational Linguistics, Sofia, Bulgaria (August 2013)
5. Novák, A.: Milyen a jó Humor? In: I. Magyar Számítógépes Nyelvészeti Konferencia. pp. 138–144. SZTE, Szeged (2003)
6. Novák, A.: A Humor új Fo(r)mája. In: X. Magyar Számítógépes Nyelvészeti Konferencia. pp. 303–308. SZTE, Szeged (2014)
7. Novák, A.: A model of computational morphology and its application to Uralic languages. Ph.D. thesis, Roska Tamás Doctoral School of Sciences and Technology Pázmány Péter Catholic University, Faculty of Information Technology and Bionics, Budapest (2015)
8. Prószéky, G., Kis, B.: A unification-based approach to morpho-syntactic parsing of agglutinative and other (highly) inflectional languages. In: Proceedings of ACL '99. pp. 261–268. Association for Computational Linguistics, Stroudsburg, PA, USA (1999)
9. Rebrus, P., Kornai, A., Varga, D.: Egy általános célú morfológiai annotáció. Általános Nyelvészeti Tanulmányok XXIV., 47–80 (2012)
10. Trón, V., Halácsy, P., Rebrus, P., Rung, A., Vajda, P., Simon, E.: Morphdb.hu: Hungarian lexical database and morphological grammar. In: Proceedings of LREC 2006. pp. 1670–1673 (2006)
11. Váradi, T., Simon, E., Novák, A., Indig, B., Farkas, R., Vincze, V., Sass, B., Gerőcs, M., Iván, M.: e-magyar.hu: digitális nyelvfeldolgozó rendszer. In: XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017) (2017)