# 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 mondathatárt, 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.).