سومین همایش ملی
زبان شناسی رایانشی
۲۸ و ۲۹ آبان ۱۳۹۳
دانشگاه صنعتی شریف

دانشگاه صنعتی شریف
انجمن زبان‌شناسی ایران

طراحی تحلیلگر ساختواژی برای زبان پارسی

با استفاده از ساختار تراگذر و روش دوسطحی

مجتبی روحانیان
دانشجوی کارشناسی ارشد
دانشگاه صنعتی شریف
rohanian@mehr.sharif.edu

کامران معافی غفاری
دانشجوی کارشناسی ارشد
دانشگاه صنعتی شریف
ghaffari@mehr.sharif.edu

بهرام وزیرنژاد
استادیار
دانشگاه صنعتی شریف
bahram@sharif.edu

چکیده

در این پروژه، با استفاده از ساختار ریاضی تراگذر[1] و بر اساس مدل ساختواژی دوسطحی (two-level morphology) مدلی به منظور ساخت یک تحلیلگر ساختواژی برای زبان پارسی طراحی گردید. در پیاده سازی این نرم افزار، از زبان های برنامه نویسی لکس (lexc)، پایتون و جعبه ابزار اسکریپت نویسی فوما (foma) استفاده شد. حاصل کار به صورت یک نرم افزار متن-باز مجزا و هم به عنوان بخشی از یک جعبه ابزار پردازش زبان پارسی قابل استفاده و به صورت رایگان در دسترس کلیه پژوهشگران است. در این نرم افزار نه تنها قواعد ساختواژی، بلکه واژگان نیز در قالب تراگذر عمل می کند. ویژگی دیگر، دوسویه بودن نرم افزار است، به این ترتیب امکان هر دو عمل تجزیه و ترکیب وجود دارد. نکته ی حائز اهمیت این پروژه، ایجاد pipe میان اسکریپت های فوما و پایتون است. لذا از نتایج فوما می توان به عنوان ورودی در پایتون استفاده کرد و با پردازش های بعدی نتایج نهایی را بهبود بخشید.

کلیدواژه‌ها: تحلیلگر ساختواژی، پارادایم دوسطحی، تراگذر، فوما، زبان لکس

۱. مقدمه

ساختواژه یکی از بخش های علم زبان شناسی است که به بررسی و مطالعه ساختار درونی کلمات و ارتباط اجزای آنها با یکدیگر و معنا می‌پردازد. ساختواژه به دو بخش اصلی تقسیم می شود: تصریف و واژه سازی. تصریف به بررسی صورت کلمات می‌پردازد و واژه سازی ناظر بر فرایند تولید واژه‌های جدید در یک زبان است. وجود تحلیلگرهای ساختواژی خودکار که بتوانند اجزای کلمات را شناسایی و نقش آنها را تحلیل کنند همواره یکی از نیازهای اساسی بسیاری از ابزارهای پردازش زبان می باشد. از این رو تلاش هایی در این راستا صورت گرفته است که به برخی از آنها اشاره می شود.

[1] transducer

یکی از تحلیلگرهای ساختواژی تولید شده مربوط به پروژه پارس‌مورف می باشد (Mavaji, V., Eslami, M., Vazirnezhad, B., 2012). این تحلیلگر توانایی تحلیل ساختار تصریفی کلمات و هم چنین تحلیل کلمات مشتق و مرکب را دارد و طبق ادعای نگارندگان دارای دقت ۹۵ درصد می باشد. البته معیار سنجش و نحوه دستیابی به این دقت از سوی نگارندگان ارائه نشده است. پارس مورف به صورت متن-باز نمی‌باشد ولی امکان استفاده از دموی آن فراهم است.

پروژه STeP1 (Shamsfard M., Jafari, HS., Ilbeygi M., 2010) شامل چند ابزار مختلف پردازش زبان فارسی است که تحلیلگر ساختواژی یکی از ابزارهای آن می باشد. این تحلیلگر بر اساس اتوماتون حالت محدود[۲] طراحی شده است و می‌تواند ستاک و وندهای تصریفی و برخی از وندهای اشتقاقی را شناسایی و تحلیل کند. در این سیستم از یک برچسب‌زن مقوله نحوی نیز کمک گرفته شده است. این سیستم دو سویه نیست و صرفا برای امر تجزیه کاربرد دارد و هدف اصلی آن از تحلیل ساختواژی یافتن ستاک است. این پروژه نیز به صورت متن-باز ارائه نشده است.
تحلیلگر ساختواژی قابل ذکر دیگر پروژه Perstem است که به زبان برنامه نویسی پرل[۳] نوشته شده و دارای استمر، تحلیلگر ساختواژی و برچسب‌زن مقوله نحوی است (Dehdari, Lonsdale, 2008).

معماری نرم افزار تحلیلگر ساختواژی که در این پروژه ساخته شده بر مبنای الگویی در زبان شناسی رایانشی است که با نام مدل دو سطحی شناخته می شود. این روش در سال ۱۹۸۳ به وسیله ی زبان شناس فنلاندی کیمو کاسکنیمی ابداع شده است (Karlsson,2007:1). در این پروژه قواعد ساختواژی برگرفته از *ساختار تصریفی کلمه در زبان فارسی* (اسلامی، محرم و علیزاده، صدیقه. ۱۳۸۸) می باشند و از *واژگان زایای زبان فارسی* (اسلامی و همکاران. ۱۳۸۳) به منظور انتخاب صورت واژگانی کلمات استفاده شده است.

ایده ی این روش، ملهم از کارهایی است که در اواخر دهه ۶۰ میلادی در آواشناسی زایشی انجام شد. در آن زمان برای اولین بار، به منظور نمایش تبدیلات واجی، از *قواعد بازنویسی وابسته به بافت*[۴] استفاده شد. (Chomsky, Halle,1968) مثلا:

$$x \Longrightarrow y \mid z \_ w$$

یعنی x به y تبدیل می شود، در حالتی که x بین z و w قرار داشته باشد. همانند نحو، نظام آوایی زبان را می‌توان در دو سطح روساخت و زیرساخت نشان داد. روساخت آن چیزی است که به وقوع پیوسته و ما شاهد آن هستیم. یعنی تجسد زبان به صورت بالفعل است. اما یک ژرف ساخت نیز وجود دارد که نشان دهنده دانش زبانی

[2] Finite State Automaton
[3] Perl
[4] context-sensitive rewrite rules

گوینده است و نمایشگر تغییراتی است که گویشور زبان پیش از بیان یک رشته از آواها روی زبان اعمال می کند. به نظر چامسکی و هالی، این تغییرات با قواعد وابسته به بافت که شرحش رفت قابل توضیح هستند. (Karttunen, Beesley, 2001: 1)

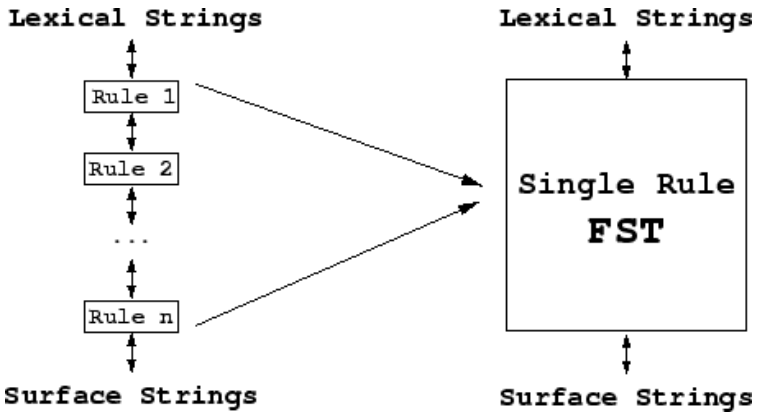
از سوی دیگر در طبقه بندی که چامسکی از زبان ها ارائه می دهد (Chomsky, 1956: 113–124)، زبان های وابسته به متن یکی از قوی ترین انواع زبان هاست (شکل ۱).

شکل ۱. رده بندی زبان ها از دیدگاه چامسکی

در اوایل دهه ی هفتاد میلادی، ثابت گردید که با آنکه قواعد بازنویسی با زبان های وابسته به متن قابل توضیح هستند، با این حال می توان آنها را با ابزار FST و زبان های رگولار که از مرتبه ی پایین تر هستند نیز مدل سازی کرد (Douglas, 1972).

از سوی دیگر می دانیم که هر مجموعه ای از آتاماتا را می توان با هم ترکیب کرده و به یک آتاماتون بزرگ تر رسید(Schützenberger, 1961) (شکل ۲) این خاصیت، زبان شناسان را به این فکر انداخت که آواشناسی و ساختواژه ی زبان ها را با ابزار آتاماتا مدل سازی کنند(Kaplan, Kay, 1981)

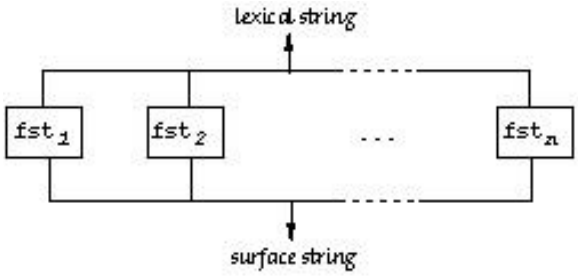
شکل ۲. قواعد بازنویسی را می توان ترکیب کرده و در یک fst واحد به کار برد (Karttunen, Beesley, 2001: 1)



۲. جزئیات مدل دو سطحی

تا پیش از مدل دوسطحی، روش هایی که برای مدل سازی ساختواژه ی زبان از fst استفاده می کردند همگی اشکالی عمده داشتند. مشکل اینجا بود که وقتی قواعد ساختواژی را به صورت دنباله وار[۵] به هم اثر دهیم و ترکیب کنیم، تراگذر از یک طرف درست عمل می کند اما اگر سوی تراگذر را عوض کنیم تا مثلا به جای عمل ترکیب عمل تجزیه را انجام دهد، در تحلیل خود دچار خطا می شود و کلماتی را که در واژه نامه نیستند نیز به عنوان جوابهای ممکن به خروجی می فرستد و در برخی حالات که قواعد به صورت بازگشتی روی هم اثر می‌کنند امکان بروز حلقه بی نهایت[۶] نیز وجود دارد.

کاسکنیمی برای حل نسبی این مشکل، دست به طراحی یک فرمول بندی ریاضی جدید از قواعد بازنویسی زد. از نظر او قواعد بازنویسی که توسط چامسکی و هالی ابداع شده اند در زمینه ی ساختواژه کارآمد نیستند، چرا که آنها به صورت دنباله وار اعمال می شوند. او قواعد را طوری فرمول بندی کرد که نه به صورت دنباله وار بلکه به صورت دوسویه[۷] و موازی[۸] روی هم اثر کنند. (Koskenniemi, 1983) (شکل ۳)



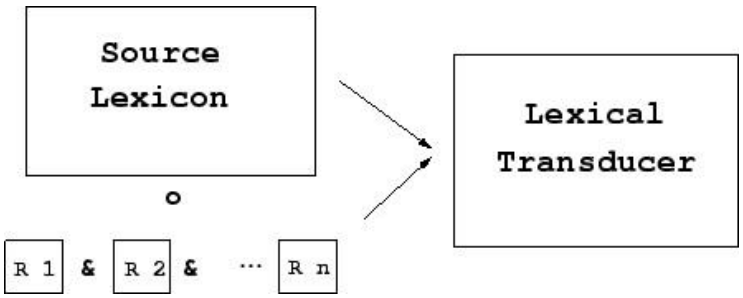
[5] sequential
[6] infinite loop
[7] bidirectional
[8] parallel

شکل ۳. fstها در پارادایم دوسطحی به صورت موازی عمل می کنند (Karttunen, 1991)

مثلا قاعده ای که طبق قواعد بازنویسی چامسکی و هالی در ابتدای مقاله ذکر شد، طبق فرمول بندی جدید به صورت زیر نوشته می شود:

$$\begin{matrix} x \\ y \end{matrix} \Longleftrightarrow z \_ w$$

یعنی x وy زمانی که میان z و w واقع شوند با همدیگر معادلند. دوسویه بودن این فرمول بندی، به زبان شناس اجازه می دهد تا قواعدی تنظیم کند که هر دو روال تجزیه و ترکیب را کنترل کرده و بیشتر کلماتی که ممکن است در واژگان نبوده و به اشتباه در ترکیب تولید شده اند را غربال کند. به معنای دیگر، در پارادایم دوسویه، نه تنها قواعد به صورت تراگذر هستند، بلکه خود واژگان نیز به شکل تراگذر طراحی می شود و در انتها تراگذر ترکیبی قواعد و تراگذری که از ساخت واژگان ساخته شده است را با هم ترکیب می کنیم. لذا کل نرم افزار به صورت یک تراگذر واحد قابل نشان دادن خواهد بود.(شکل 4)



شکل 4. تراگذر واحد در پارادایم دوسطحی(Karttunen, Beesley, 2001)

به عنوان مثال استفاده از این خاصیت در نرم افزاری که برای پارسی طراحی شد به این معنی است که کاربر امکان این را دارد تا نه تنها از نرم افزار در دو سوی تجزیه و ترکیب بهره ببرد بلکه تمام حالت های مختلف تحلیل یک کلمه را نیز مشاهده کند. مثلا ورودی Xaridam (خریدم) به دو صورت مختلف تحلیل می شود که نرم افزار به درستی هر دو حالت را با استفاده از قواعد ساختواژی تحلیل می کند:

Xaridam
+Pos**xarid**+1p+Sg
+Pos**xar**+3p+Sg+ObjC+1p+Sg

با همکاری کاسکنیمی و محققان دانشگاه استنفورد خصوصا لاری کارتونن که با بخش تحقیقاتی شرکت Xerox در ارتباط بود، مدل سازی رایانه ای این مدل صورت گرفت و در قالب یک زبان برنامه نویسی (lexc) به

همراه یک کامپایلر و تفسیرگر به نام xfst[9] عرضه شد (Karttunen et al, 1987). foma یک نسخه ی متن-باز بر مبنای همین نرم افزار است.[10]

از اوایل دهه ی ۹۰، و با ساخت نرم افزار PC-KIMMO استفاده از این روش در ساخت تحلیلگرهای ساختواژی رونق بیشتری گرفت (Antworth, 1990) و تا به حال زبان های زیادی، از جمله عربی (Beesley, 1989) با این روش تحلیل شده اند. برای زبان پارسی، تنها یک پروژه به صورت متن-بسته وجود دارد که در آن از فرمول-بندی lexc استفاده شده است (Megerdomian, 2000).

۳. اسکریپت نویسی و پیاده سازی تحلیلگر ساختواژی برای زبان پارسی

در پروژه حاضر، شیوه ی کار به این گونه بود که برای هرکدام از مقوله های اصلی ساختواژی (فعل، اسم، صفت، قید،...) یک فایل اسکریپت جداگانه با پسوند lexc. نوشته شد و سپس در فایل نهایی foma، این ترانسدیوسرها فراخوانی و ترکیب شد و درمرحله آخر ویرایش‌های واجی نیز بر روی آنها اعمال گردید.

در اسکریپت‌های فوما، انتهای رشته با # نشان داده می شود. ۰ نشاندهنده ی رشته‌ی تهی است. نیم فاصله با ~ و فاصله نیز با ^ نمایش داده شده است.
مثلا برای ساخت واژگان اسم به صورت تراگذر، ابتدا با دستور Root باید ساخت‌های پایه‌ای واژه را مشخص کنیم. در اینجا پیشوندها و ریشه‌ها برای زبان پارسی کافیست تا همه‌ی تغییرات را از روی آنها بسازیم:

```
LEXICON Root
N;
NounPrefix;
```

پس از آن، بر اساس برچسب‌های از پیش تعریف شده، عناصر Root را تعریف می‌کنیم. به طور مثال، پیشوند منفی ساز بی در اسامی پارسی به این شکل تعریف شد:

```
LEXICON NounPrefix
+Neg:by^ N;
```

به این معنی که با داشتن تگ Neg+ در مقوله‌ی اسم، یکی از ساختارهای ممکن به صورت: بی+فاصله+اسم می باشد.

برای تعریف N نیز کافیست تا ریشه های مربوطه را نوشته و پسوندهای تصریفی پایانی اسامی را مشخص کنیم:

```
LEXICON     N
ktaab       NInf;
prndh       NInf;
```

[9] Xerox Finite State Tool
[10] http://foma.googlecode.com

پایانه های اسمی بنا به تگ مربوطه مقدارشان تغییر می کند. مثلا برای حالت اسم مفرد این پایانه مقدار تهی دارد اما در حالتی که تگ جمع داریم، پایانه ی اسمی "ها" یا "ان" (یا جمع عربی ات) است و پس از آن به انتهای رشته می رسیم:

```
LEXICON NInf
+N+Sg:0 #;
+N+Pl:^haa #;
+N+Pl:^aan #;
+N+Arb+F+Pl:^aat #;
+N+Arb+Pl:^yn #;
+N+Pl+Indf:^haayy #;
+N+Pl+Indf:^aany #;
```

تمامی واژه بست ها هم در همین قسمت با تگ صحیح تعریف شده است. مثلا برای واژه بست ـم:

```
+N+Poss+1P+Sg:^m #;
```

جمع های مکسر را نیز همانند پروژه ی دیگر پارسی (Megerdoomian, 2004) به صورت استثنا و جداگانه تعریف می کنیم. مثلا برای جمع عربی کتب داریم:

```
LEXICON N
ktaab+N+Pl:ktb #;
```

برای حالاتی که پس از اعمال پایانه، تغییرات آوایی داریم (مثلا: پرنده+ ان = پرندگان) در مرحله ی نهایی که تراگذرها را با هم ترکیب می کنیم، قواعدی تعریف کرده و این تصحیحات را انجام می دهیم. ضمن آنکه علائم مرزهای کلمه را با تعریف قواعد جدید حذف می کنیم:

```
define Gaf h "^" -> g || _ [{aan}|{aany}|{y}];
define CaretDeletion "^" -> 0;
```

قواعد بالا، هاء را در حالتی که قبل از بعضی رشته های خاص بیاید به گاف تبدیل می کند و مرزها را نیز حذف می نماید. با این توضیح، مثلا اگر ورودی *مردها* و یا *پرندگان* را برای تجزیه داشته باشیم، خواهیم داشت:

```
mrdhaa
mrd+N+Pl

prndgaan
prndh+N+Pl
```

برای بررسی افعال فارسی ابتدا در قسمت بالایی اسکریپت فوما، تگهای اصلی را به شرح زیر معرفی می کنیم:

+Pos	حالت مثبت
+Neg	حالت منفی

+Inf	حالت مصدری
+Cn	استمراری
+Subjcv	التزامی
+Imp	امری
+Sim	ساده
+PPart	ماضی نقلی
+PPerf	ماضی بعید
+F	آینده
+Pass	مجهول
+1p	اول شخص
+2p	دوم شخص
+3p	سوم شخص
+Sg	مفرد
+Pl	جمع
+ObjC	واژه بست مفعولی

حال برای ساختن لکسیکون از ریشه (Root) شروع می‌کنیم و نخست به بررسی زمان های حال استمراری و التزامی می پردازیم. در این حالت، پیشوندی که تگ آن مشخص است به فعل V1 ختم می شود:

LEXICON Root

+Neg+Cn:nemi V1;

+Pos+Cn:mi V1;

+Neg+Subjcv:n V1;

+Pos+Subjcv:b V1;

افعال نوع V1 در انتهای اسکریپ با فرمت زیر تعریف شده اند:

LEXICON V1

afkan V1inf;

afraaz V1inf;

با این معنی که بن فعل مضارع به متغیر V1inf که در واقع inflection فعل می باشد منتهی شده است. سپس مشخص می کنیم که V1inf چه مقادیری را دریافت می کند:

```
LEXICON V1inf
+1p+Sg:am    #;
+2p+Sg:i     #;
+3p+Sg:ad    #;
+1p+Pl:im    #;
+2p+Pl:id    #;
+3p+Pl:and   #;
+1p+Sg+ObjC:am  ObjClitics;
+2p+Sg+ObjC:i   ObjClitics;
+3p+Sg+ObjC:ad  ObjClitics;
+1p+Pl+ObjC:im  ObjClitics;
+2p+Pl+ObjC:id  ObjClitics;
+3p+Pl+ObjC:and ObjClitics;
```

مثلا "نمی افکنم" را می توان به این شکل ساخت:

+Neg+Cn**afkan**+1p+Sg

برای حالاتی که مانند مثال بالا، پس از جایگذاری به انتهای رشته ی حرفی می رسیم، آن را با علامت # نشان می دهیم. ممکن است در انتهای فعل، واژه بست مفعولی وجود داشته باشد که بنا به بافت مقدار آن تغییر کند (مثال: نمی افکنمش) برای نشان دادن این حالات، رشته به متغیر objClitics ختم شده است و برای تعیین مقادیر مختلف آن نیاز به تعریف دسته مجزایی داریم:

```
LEXICON ObjClitics
+1p+Sg:am       #;
+2p+Sg:at       #;
+3p+Sg:ash      #;
+1p+Pl:eman     #;
+2p+Pl:etan     #;
+3p+Pl:eshan    #;
```

پس از این به سراغ افعال امری مثبت و منفی می رویم. برای این کار به ذیل Root بازگشته و تگهای مناسب و مجموعه ی فعلی جدید را معرفی می کنیم:

+Neg+Imp:n V2;

+Pos+Imp:b V2;

گروه فعلی V2 نیز همانند گروه قبلی شامل یک سری بن مضارع است که به متغیر مرتبط با گروه خود ختم شده است:

LEXICON V2

aab V2inf;

aafarin V2inf;

afkan V2inf;

afraaz V2inf;

تعریف V2inf ساده است. در حالت مفرد به رشته ی تهی می رسیم و در حالت جمع به رشته ی id بر می‌خوریم. در حالتی که واژه بست مفعولی داشته باشیم نیز متغیر مربوطه در انتهای رشته عمل می کند:

```
LEXICON V2inf
+Sg:0   #;
+Pl:id  #;
+Sg+ObjC:0   ObjClitics;
+Pl+ObjC:id  ObjClitics;
```

حالا می توانیم به سراغ گذشته ی ساده و استمراری برویم. نکته ی اصلی اینجاست که برای این کار به گروه فعلی دیگری (V3) نیاز داریم که بر مبنای بن ماضی ساخته بشوند. پس فرمت کلی افعال این گروه به صورت زیر است:

LEXICON V3

goft V3inf;

به قسمت Root بازگشته و تگهای مربوطه را اضافه می کنیم:

```
+Neg+Cn:nemi V3;
+Pos+Cn:mi V3;
+Neg+Sim:n V3;
+Pos+Sim: V3;
```

همانطور که مشاهده می شود، مورد آخر که نشاندهنده ی گذشته ی ساده است، دارای پیشوند نیست (مثال: رفت)

در ادامه، مطابق روال قبلی V3inf را تعریف می کنیم:

```
LEXICON V3inf
+1p+Sg-ObjC:am  #;
+2p+Sg-ObjC:i   #;
+3p+Sg-ObjC:0   #;
+1p+Pl-ObjC:im  #;
+2p+Pl-ObjC:id  #;
+3p+Pl-ObjC:and #;
+1p+Sg+ObjC:am  ObjClitics;
+2p+Sg+ObjC:i   ObjClitics;
+3p+Sg+ObjC:0   ObjClitics;
+1p+Pl+ObjC:im  ObjClitics;
+2p+Pl+ObjC:id  ObjClitics;
+3p+Pl+ObjC:and ObjClitics;
```

گروه چهارم فعلی، حالات مثبت و منفی گذشته ی نقلی است (گفته ام، نگفته ام) که در Root منظور می شود:

+Neg+PPart:n V4;

+Pos+PPart: V4;

و فرمت کلی V4 چنین است:

LEXICON V4

anduxt V4inf;

V4inf را هم همانند موارد قبل با استفاده از تگهای مناسب و شناسه های مربوطه تعریف می کنیم و جایی را هم برای حالات منتهی به واژه بست مفعولی در نظر می گیریم:

```
LEXICON V4inf
+1p+Sg-ObjC:eam      #;
+2p+Sg-ObjC:ei       #;
+3p+Sg-ObjC:east     #;
+1p+Pl-ObjC:eim      #;
+2p+Pl-ObjC:eid      #;
+3p+Pl-ObjC:eand     #;
+1p+Sg+ObjC:eam      ObjClitics;
+2p+Sg+ObjC:ei       ObjClitics;
+3p+Sg+ObjC:east     ObjClitics;
+1p+Pl+ObjC:eim      ObjClitics;
+2p+Pl+ObjC:eid      ObjClitics;
+3p+Pl+ObjC:eand     ObjClitics;
```

گذشته ی بعید را هم دقیقا با همین ساختار تعریف می کنیم:

+Neg+PPerf:n V5;

+Pos+PPerf: V5;

با این تفاوت که در تعریف متغیرهای این گروه، شناسه های ماضی بعید استفاده می گردد:

```
LEXICON V5inf
+1p+Sg-ObjC:ebudam  #;
+2p+Sg-ObjC:ebudi   #;
+3p+Sg-ObjC:ebud    #;
+1p+Pl-ObjC:ebudim  #;
+2p+Pl-ObjC:ebudid  #;
+3p+Pl-ObjC:ebudand #;
+1p+Sg+ObjC:ebudam  ObjClitics;
+2p+Sg+ObjC:ebudi   ObjClitics;
+3p+Sg+ObjC:ebud    ObjClitics;
+1p+Pl+ObjC:ebudim  ObjClitics;
+2p+Pl+ObjC:ebudid  ObjClitics;
+3p+Pl+ObjC:ebudand ObjClitics;
```

برای گروه فعلی ششم، سراغ فعل های مجهول می رویم که از بن ماضی ساخته شده اند (خورده شده است):

```
+Pos+Pass: V6;
+Neg+Pass:n V6;
```

فرمت کلی فعل های گروه۶:

xord V۶inf;

نحوه ی تعریف V6inf:

```
LEXICON V6inf
+1p+Sg:shodeam      #;
+2p+Sg:shodei       #;
+3p+Sg:shodeast #;
+1p+Pl:shodeim      #;
+2p+Pl:shodeid      #;
+3p+Pl:shodeand #;
+1p+Sg+ObjC:shodeam     ObjClitics;
+2p+Sg+ObjC:shodei      ObjClitics;
+3p+Sg+ObjC:shodeast    ObjClitics;
+1p+Pl+ObjC:shodeim     ObjClitics;
+2p+Pl+ObjC:shodeid     ObjClitics;
+3p+Pl+ObjC:shodeand    ObjClitics;
```

گروه فعلی هفتم، حالات التزامی را در بر می گیرد (خورده باشم) که بنیان آن همانند گروه قبل است با این تفاوت:

```
LEXICON V7inf
+1p+Sg:ebasham  #;
+2p+Sg:ebashi   #;
+3p+Sg:ebashad  #;
+1p+Pl:ebashim  #;
+2p+Pl:ebashid  #;
+3p+Pl:ebashand #;
+1p+Sg+ObjC:ebasham     ObjClitics;
+2p+Sg+ObjC:ebashi      ObjClitics;
+3p+Sg+ObjC:ebashad     ObjClitics;
+1p+Pl+ObjC:ebashim     ObjClitics;
+2p+Pl+ObjC:ebashid     ObjClitics;
+3p+Pl+ObjC:ebashand    ObjClitics;
```

گروه هشتم برای نشان دادن حالات مصدری استفاده می شود و ساختار ساده ای دارد:

+Inf: V8;

و

LEXICON V۸

afkand #;

و

```
LEXICON V8inf
0:an #;
```

گروه نهم و آخر برای زمان آینده است و آن نیز به این شکل تعریف می شود:

```
+Pos+F: V9inf;
+Neg+F:n V9inf ;
```

فرمت گروه ۹ دقیقا مانند گروه ۸ است اما متغیر پیشوندی آن به شکل زیر است:

```
LEXICON V9inf
+1p+Sg-ObjC:xaham    V9;
+2p+Sg-ObjC:xahi     V9;
+3p+Sg-ObjC:xahad    V9;
+1p+Pl-ObjC:xahim    V9;
+2p+Pl-ObjC:xahid    V9;
+3p+Pl-ObjC:xahand   V9;
```

اکنون اسکریپت را کامپایل کرده و آن را با چند مثال تست می کنیم:

read lexc Verbs.lexc

خواهم افکند:

```
foma[0]: read lexc Verbs.lexc
Root...21, V1inf...12, V2inf...4, V3inf...12, V4inf...12, V5inf...12, V6inf...12
, V7inf...12, V8inf...1, V9inf...6, ObjClitics...6, V1...12, V2...12, V3...13, V
4...13, V5...13, V6...13, V7...13, V8...13, V9...13
Building lexicon...
Determinizing...
Minimizing...
Done!
12.7 kB. 526 states, 763 arcs, 9073 paths.
foma[1]: up
apply up> xaaham^afkand
+Pos+F+1p+Sgafkand
apply up>
```

+Pos+F+1p+Sg**afkand**

گفته بودمتان:

```
apply up> goft~ebudam~etaan
+Pos+PPerfgoft+1p+Sg+ObjC+2p+Pl
apply up>
```

+Pos+PPerf**goft**+1p+Sg+ObjC+2p+Pl

رسیده باشد:

```
apply up> resid~ebaashad
+Pos+Subjcvresid+3p+Sg
apply up>
```

+Pos+Subjcv**resid**+3p+Sg

بکش:

```
apply up> b~kesh
+Pos+Impkesh+Sg
apply up>
```

+Pos+Imp**kesh**+Sg

بریده شده است:

```
apply up> borid~eshodeast
+Pos+Passborid+3p+Sg
apply up>
```

+Pos+Pass**borid**+3p+Sg

۴. ارزیابی و نتیجه گیری

نرم افزار حاضر اولین تحلیلگر ساختواژی متن-باز پارسی است که مطلقا با مدل دوسطحی طراحی شده و صرفا از ساختار ریاضی تراگذر استفاده کرده است. کل محتوای نرم افزار در قالب یک تراگذر واحد و با فرمت DOT موجود است و تصویر این تراگذر نیز با استفاده از نرم افزار gvedit تولید شده و قابل مشاهده است. به وسیله ی اسکریپتی که در پایتون نوشته شد، خروجی های فوما با پایتون pipe گردید و می توان گرفتن ورودی، ذخیره‌ی خروجی در فایل مجزا و مشاهده ی نتایج را در دل یک GUI که در پایتون طراحی شده است انجام داد. همین طور می توان لیستی از کلمات را در فایلی به عنوان ورودی به برنامه داد و خروجی تحلیل شده را با همان فرمت در فایلی جدا مشاهده کرد.

به دلیل دو سویه بودن تراگذرها و ساختمان کلی واژگان، میزان دقت این تحلیلگر ۹۵ درصد است. میزان اندک خطای سیستم مربوط به معدود حالاتی است که نیاز به بررسی بیشتر دارد. مثلا در هنگام ترکیب اگر ورودی را mrd+N+Pl وارد کنیم دو حالت زیر را دریافت می کنیم:

mrdhaa
mrdaan

اگر کدهای فوما بخشی از یک جعبه ابزار بزرگتر باشد بهتر است بعد از pipe شدن فوما با نرم افزار مربوطه، با وزن دهی به نتایج در محیط جدید، آنها را یا غربال کرده و تنها بهترین را انتخاب کنیم یا صرفا به رده بندی آنها اکتفا کنیم. با این حال بدون در نظر داشتن بافت متنی انجام این کار چندان منطقی نیست.

از طرفی در ترکیب، ممکن است حالت هایی تولید شود که چندان مصطلح نیست یا از لحاظ آوایی ناخوشایند است، مثلا *ناخوب* یا *دفتران*. غیر از راه حل ثانوی که در بالا گفته شد، می توان در اسکریپت های فوما نیز برای اسامی خاصی که برخی از پایانه ها را نمی پذیرند برچسب گذاشت و از ترکیب شدنان با آن پایانه ها اجتناب کرد. انجام این کار، به تحلیل زبان شناختی مجزایی نیاز دارد و در این نسخه از نرم افزار، تا جایی که ممکن بود این قواعد اعمال گردید و برای نسخه های بعدی اقدام به تکمیل آن خواهد شد.

منابع

Antworth, Evan L. (1990), *PC-KIMMO: a two-level processor for morphological analysis.* Occasional Publications in Academic Computing No. 16. Dallas, TX: Summer Institute of Linguistics.

Beesley, K. R. (1989), *Computer analysis of Arabic morphology: A two-level approach with detours.* In *Third Annual Symposium on Arabic Linguistics*, Salt Lake City. University of Utah.

Beesley, K. R, Karttunen. Lauri. (2003), *Finite-State Morphology: Xerox Tools and Techniques*. CSLI Publications, Palo Alto. California

Chomsky, Noam (1956). "Three models for the description of language". IRE Transactions on Information Theory (2)

Chomsky, Noam and Halle, Morris. (۱۹۶۸), *The Sound Pattern of English*. New York: Harper & Row

Dehdari, J., Lonsdale, D. (2008).A link grammar parser for Persian. In Simin Karimi,Vida Samiian, and Don Stilo, editors, *Aspects of Iranian Linguistics*, volume 1. Cambridge Scholars Press.

Johnson, C. Douglas (1972), *Formal Aspects of Phonological Description*. Mouton. The Hague.

Karlsson, Fred.(2007) *Kimmo Koskenniemi's first 60 years*. Stanford: CSLI Publications. Retrieved 2007-10-07 from:
http://www.ling.helsinki.fi/~fkarlsso/Kimmo_60_years.pdf

Kaplan, Ronald M. and Martin Kay. 1981. *Phonological rules and finite-state transducers*. In Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting. New York. Abstract.

Karttunen. Lauri (1991), Finite-state constraints In Proceedings of the International Conference on Current Issues in Computational Linguistics, Penang, Malaysia. University Sains Malaysia Retrieved 2014-08-31 from: http://web.stanford.edu/~laurik/publications/fsc-91/fsc91.html

Karttunen, L., Beesley, K.,R. (2001), *A short History of Two-level Morphology*, Paper Presented at European Summer Schools in Logic, Language and Information 2001 Special event "Twenty Years of Two-level Morphology", Helsinki, Finland. Retrieved 2014-08-30 from: http://www.ling.helsinki.fi/~koskenni/esslli-2001-karttunen/helsinki.html

Karttunen. Lauri, Koskenniemi, Kimmo., M. Kaplan, Ronald (1987), *A Compiler for Two-level Phonological Rules*. In Dalrymple, M. et al. *Tools for Morphological Analysis*. Center for the Study of Language and Information. Stanford University. Palo Alto. California

Koskenniemi, Kimmo (1983), *Two-level morphology: a general computational model for word-form recognition and production.* Publication No. 11. Helsinki: University of Helsinki Department of General Linguistics

Megerdoomian, Karine (2000), Unification-based Persian morphology. In Proceedings of CICLing , Mexico.

Megerdoomian, Karine (2004), Finite-State Morphological Analysis of Persian. In Proceedings of COLING'04 Workshop on Enhancing and Using Electronic Dictionaries. Geneva, Switzerland.

Schützenberger, M.-P. (1961), *Two level formalisms*. In Utrecht Working Papers in NLP, volume 5.

Shamsfard, M., Kiani, S., Shahedi, Y. (2009). STeP-1: standard text preparation for Persian language. *CAASL-3 – Third Workshop on Computational Approaches to Arabic Script-based Languages* [at] *MT Summit XII*, Ottawa, Ontario, Canada.

اسلامی، محرم، شریفی آتشگاه، مسعود، علیزاده لمجیری، صدیقه و زندی، طاهره. ۱۳۸۳. واژگان زایای زبان فارسی، مجموعه مقالات اولین کارگاه پژوهشی زبان فارسی و رایانه، دانشگاه تهران، تهران.

اسلامی، محرم و علیزاده، صدیقه. ۱۳۸۸. ساخت تصریفی کلمه در زبان فارسی، زبان و ادب فارسی، نشریه دانشکده ادبیات و علوم انسانی دانشگاه تبریز، سال ۵۲، شماره مسلسل ۲۱۱، ایران.