بسم الله الرحمان الرحیم

پروژه تحلیلگر لغوی

مصطفی وحدانی دهکلانی

مستندات پروژه:

NFA و DFA های پروژه برای بررسی راحت تر به همراه محتویات پروژه در فایل های جداگانه به نام های DFA.pdf و NFA.pdf قرار گرفته اند و می توانید برای بررسی جهت فهمیدن راحت تر روند اجرای کد نوشته شده به آنها مراجعه کنید.

هدف از نوشتن این پروژه این بوده است که با استفاده هر زبان برنامه نویسی که علاقمند هستیم یک تحلیلگر لغوی را با استفاده از تبدیلات RE به DFA و سپس ایجاد کردن جدول انتقال حالت و در نهایت تبدیل آن به کد طراحی کنیم.

نکات مهم:

نکته اول) در فایل های تحویل داده شده به دلیل اینکه جدول انتقال حالت فضای نوشتاری زیادی را اشغال می کند در این سطح فقط به ارائه DFA و NFA ها بسنده کرده ام و از طریق دیاگرام های تحویل داده شده می توان متوجه شد که الگوی هر توکن به چه شکل در کد نوشته شده تشخیص داده می شود.

نکته دوم)

توجه شود که در جهت جلوگیری از پیچیده شدن DFA های نهایی، برای آنها حالت اررو/تله/لوب رسم نکرده ام اما درون کدی که نوشته ام در صورت برخورد با حالت تله، انقدر کاراکتر بعدی را میخوانیم تا به یک کاراکتر جداکننده خط/تمام کننده فایل برسیم به محض اینکه چنین کاراکتری را در کد پیدا کردیم یک خطا در فایل outlexerror.txt گزارش می کنیم و تشخیص الگوی توکن بعدی را دوباره از سر می گیریم.

نکته سوم)

سعی شده که کد ها به صورت ماژولار و تابعی نوشته شوند و نام توابع و متغیر ها به گونه ایی انتخاب شوند تا خواندن کد برای مخاطب راحت تر باشد. در نتیجه با رعایت این موارد، توضیحاتی که در ادامه می دهم و دانستن DFA های مربوطه که پایه و اساس کارکرد تحلیلگر بر اساس آنها می باشد با مراجعه به کد بنده نباید با چالش زیادی جهت فهم کد نوشته شده رو به رو شوید.

ماژول lexdriver.py: این ماژول وظیفه پیدا کردن توکن را از طریق خواندن کاراکتر های درون لیست پایتونی مد نظر دارد و در صورت پیدا شدن یک توکن صحیح آنرا در فایل outlextoken ذخیره می کند و در صورتی که توکن خطا تولید شود آنرا در فایل outlexerror.txt ذخیره می کند.

ماژول globals.py: در این ماژول متغیر هایی را قرار داده ام که در طول اجرای برنامه مورد استفاده قرار می گیرند که از مهمترین آنها می توان متغیر های CHARACTERS_LIST، LEN_CHARACTER_LIST، RESERVED_IDS و pointer_digit را نام برد.

ماژول __init__.py: از این فایل جهت ایجاد دسترسی در سطح پروژه برای متغیر های تعریف شده درون فایل globals.py و setting.py استفاده می شود.

ماژول setting.py: از این فایل جهت یافتن مسیر فعلی کد اجرایی در سیستم شما استفاده می شود تا بتواند فایل های نهایی را در مسیر فعلی شما ذخیره کند و همچنین بتواند فایل منبع را بخواند.

ماژول utils.py: در این ماژول توابعی پایه ایی را تعریف کرده ام جهت در شناسایی الگویی توکن به ما کمک می کنند.

ماژول tokenization.py: این ماژول به بررسی الگوی هر توکن می پردازد و از توابعی که درون utils.py تعریف شده بهره می برد.

نکته چهارم)

کد نوشته شده بنده از ساختمان داده LIST زبان پایتون برای خواندن کاراکتر های درون فایل استفاده می کند به این صورت که ابتدا کل کاراکتر درون فایل را میخوانیم و درون یک LIST قرار می دهیم سپس برای اینکه بدانیم چه زمانی به آخر لیست رسیده ایم و کاراکتر های فایل ما واقعا به پایان رسیده اند یک کاراکتر اضافی (“\n” – newline) را به لیست خود اضافه میکنیم (البته که این کاراکتر فقط آخر فایل را به ما نشان می دهد و آنرا واقعا نمیخوانیم و مورد بررسی قرار نمی دهیم) و در نهایت با استفاده از از متغیری به نام pointer_digit که در سطح global پروژه تعریف کرده ایم به این متغیر در همه ی ماژول ها دسترسی داریم. هدف از این کار هم این بوده که بدانیم باید کدام کاراکتر که کاراکتر بعدی ما هست را بررسی کنیم.

نکته پنجم)

در مطالب بالا اشاره کرده ام که جهت جلوگیری از پیچیدگی DFA ها حالت اررو در خود DFA لحاظ نکرده ام اما در کدی که نوشته شده وظیفه این امر بر عهده تابع error_state می باشد که هر بار فراخوانی شد یعنی در جایی از کد حالت تله وجود دارد و باید توسط این تابع شناسایی شود

این تابع درون خود از تابع دیگری به نام error_state_location استفاده می کند که خروجی این تابع جایی از فایل است که خطای لغوی وجود دارد. خروجی این تابع به صورت زیر است:

به طور مثال فرض کنید داریم:

[(1, 2), (1, 10)]

این بدان معنی است که در خط اول فایل از کاراکتر شماره دوم خط که توکن خطا را ایجاد کرده تا کاراکتر شماره دهم که توکن معتبر پیدا شده حالت تله اتفاق افتاده.

یا برای حالتی دیگر فرض کنید داریم:

[(10, 5), (14, 20)]

این بدان معنی است که در خط دهم فایل از کاراکتر شماره پنجم خط که توکن خطا را ایجاد کرده تا خط چهاردهم کاراکتر شماره بیستم که توکن معتبر پیدا شده حالت تله اتفاق افتاده.

توجه شود که شمارش برای پیدا کردن مکان خطای مد نظر از یک شروع می شود نه صفر.

در مقابل هرگاه توکن صحیحی را پیدا کرده تابع final_state فراخوانی می شود که کارکری تقریبا مشابه به error_state دارد با این تفاوت که به دنبال کاراکتر معتبر نمی گردد و مستقیم بعد از پیدا کردن مکان توکن آنرا گزارش می کند.

خروجی تابع token_location که مکان توکن صحیح را در فایل نمایش می دهد به صورت زیر است:

به طور مثال فرض کنید:

(10, (1, 12))

این بدان معنی است که در خط دهم فایل از کاراکتر یکم تا کاراکتر دوازدهم توکن صحیحی پیدا شده است.

نکته هفتم)

در این پروژه از کتابخانه هایی نظیر:

کتابخانه os : جهت پیدا کردن مسیر فعلی برنامه در حال اجرا

کتابخانه platform : جهت شناسایی سیستم کاربر برای تشخیص path separator

کتابتخانه enum: برای ایجاد مقادیر ثابت در فایل globals.py

چگونگی اجرا کد؟

برای پیدا کردن هر توکن باید تابع next_token که درون فایل lexdriver.py ما باشد اجرا شود و تا زمانی که به آخر فایل نرسیده ایم این تابع فراخوانی می شود تا توکن بعدی شناسایی شود.