درس نظریه زبان ها و ماشین ها

دکتر فرزانه غیور باغبانی

پروژه

طراح ..... امیرمحمد درپوش

تاریخ انتشار 1402/12/28

تاریخ تحویل 1403/2/14

آداب نامه پروژه

- پروژه در قالب گروه های دو نفره انجام می شود.
- زبان های برنامه نویسی مجاز python, C#, C++ می باشند.
- هدف از پروژه طراحی شده، ارائه مثالی کاربردی از درس نظریه زبانها و ماشینها میباشد. برای راحتی کار، پیاده سازی در 5 بخش انجام میشود. با توجه به احتمال سنگین شدن کار در بخش پنجم، 4 بخش اول اصلی و بخش پنجم اختیاری خواهد بود.
- ارزیابی عملکرد شما در توسعه پروژه، در ارائه توسط تیم تی ای انجام می شود. همچنین برای اطمینان از صحت کد، در کوئرا تست کیس قرار داده شده است.
- در صورت انجام تقلب یا کپی، نمره پروژه صفر منظور خواهد شد.
- در صورت وجود هرگونه سوال یا ابهام از آقای درپوش (@dorman8288) سوال بپرسید.

در این پروژه قصد داریم یکی از کاربرد های اصلی درس نظریه زبان ها در دنیای واقعی را بررسی کنیم.

- **Lexical Scanner**

در دنیای برنامه نویسی Lexical Scanner ها نقش مهمی در پردازش زبان های برنامه نویسی و زبان های مختلف دارند و در فیلد های متنوعی مثل ساخت کامپایلر و پردازش زبان طبیعی (NLP) استفاده می‌شوند. وظیفه این تجزیه کننده ها تجزیه و تشخیص واژه های یک متن و توکن بندی آن می‌باشد. به طور مثال این کد ساده در زبان C را در نظر بگیرید.

```
int main (){
    Printf(“Hello World”);
}
```

کامپایلر در مراحل ابتدایی پردازش احتیاج دارد که هر کلمه و معنی آن در زبان مشخص شده را بداند. به طور مثال در این کد کلمه int از نوع KEYWORD و کلمه main از نوع IDENTIFIER است. کاری که Lexical Scanner ها برای ما انجام می‌دهند این است که کلمات ورودی را جدا و دسته بندی کنند تا پردازش آن ها در مراحل بعدی آسان تر باشد. به طور مثال اسکنر زبان C برای ورودی بالا توکن های زیر را به ترتیب خروجی می‌دهد.

```
int - KEYWORD
main – IDENTIFIER
( - SEPERATOR
) – SEPERATOR
{ - SEPERATOR
Printf – IDENTIFIER
( - SEPERATOR
```

“Hello World”- STRING

) – SEPERATOR

; - SEPERATOR

} – SEPERATOR

هدف این پروژه درست کردن یک Lexical Scanner Generator است. یعنی برنامه ای که با گرفتن قوانین و دسته های مختلف توکن ها بتواند یک اسکنر مخصوص آن قوانین بسازد و متن را بر آن اساس آن توکن بندی کند. برای اینکار ابتدا باید یک نگاه کلی به روش ساخت یک اسکنر بندازیم.

- **Overview of The Algorithm**

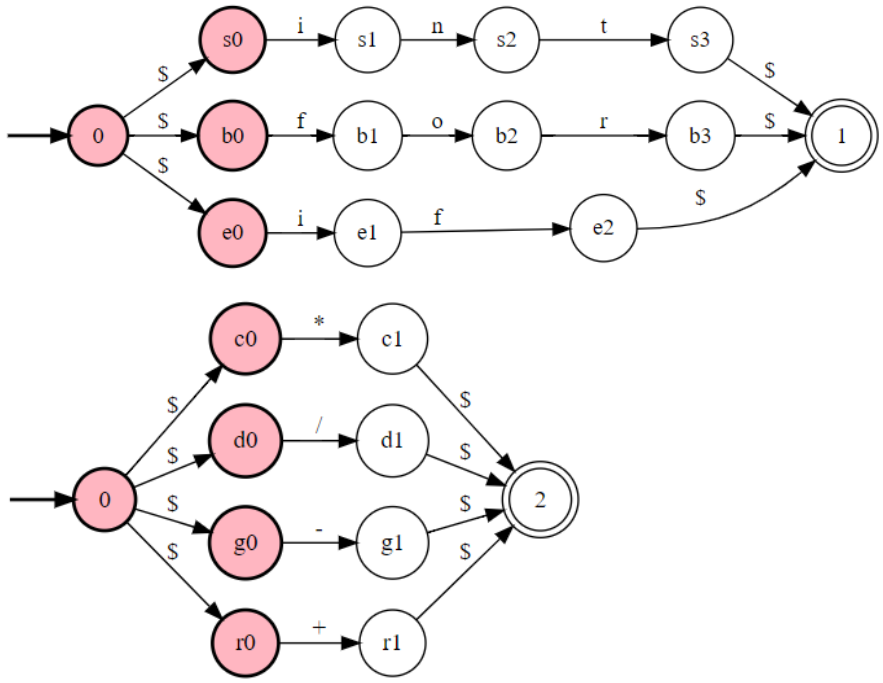
برای ساخت یک اسکنر ابتدا احتیاج داریم که قوانین گرامری مخصوص هر دسته را به صورت رسمی مشخص کنیم. در درس نظریه با یکی از این نماد گزاری ها یعنی Regular Expression ها آشنا شدید. همان طور که می‌دانید Regex ها نمایانگر زبان های منظم هستند در نتیجه می‌توان آن ها را تبدیل به ماشین های متناهی کرد که مناسب پردازش متن می‌باشند. یکی از راه های تبدیل Regex به NFA الگوریتم تامپسون است که با استفاده از آن می‌توان هر Regex ای را به یک NFA متناظر با همان زبان تبدیل کرد.

پس در قدم اول لازم است که الگو های زبان خود را با استفاده از Regex نوشته و آن ها را با استفاده از الگوریتم تامپسون به NFA های متناظر تبدیل کنیم. به طور مثال دو الگوی زیر را در نظر بگیرید.

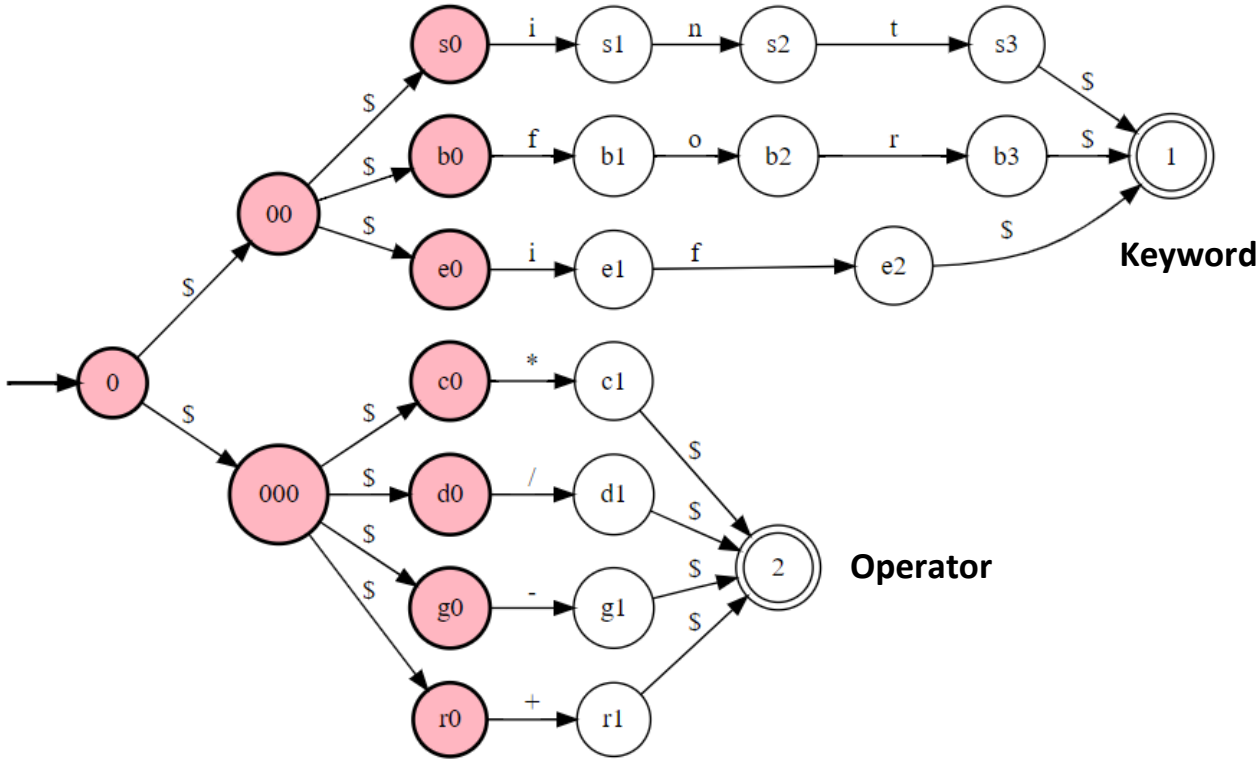
(int|for|if) → KEYWORD

(+|-|*|/) → OPERATOR

NFA های ساخته شده برای این دو الگو به این صورت می‌باشد.



حالا برای ساختن یک اسکنر که توانایی تشخیص این دو الگو را داشته باشد کافیست با اضافه کردن یک استیت این دو NFA را با هم دیگر ترکیب کنیم.



حالا برای تشخیص دسته یک توکن قبول شده توسط اسکنر کافیست به استیت نهایی نگاه کنیم. به طور مثال در صورتی که کلمه در استیت 1 قبول شده باشد از نوع Keyword و در صورتی که در استیت 2 قبول شده باشد از نوع Operator است.

به این صورت می‌توانیم برای هر تعداد الگو متفاوت یک اسکنر متناظر برای پردازش متن بسازیم. با اینکه این الگوریتم برای ساخت هر نوع اسکنری درست عمل می‌کند یک نکته منفی دارد که آن غیرقطعی بودن ماشین متناهی درست شده است. غیر قطعی بودن سرعت اسکن را چند برابر کند تر می‌کند پس برای حل این مشکل می توان از الگوریتم های تدریس شده برای تبدیل NFA به DFA و در نهایت ساده سازی DFA تولید شده استفاده کرد.

پس به صورت کلی ساخت یک Lexical Scanner متشکل از مراحل زیر است:

این مراحل به پنج بخش تقسیم شده اند که هر کدام از آن ها را در یک فاز پروژه پیاده سازی و تست می‌کنیم.

1. پذیرش رشته در Finite Automaton (25 نمره)

در این مرحله ساختمان داده یک ماشین متناهی را پیاده سازی کرده و تشخیص دهید که یک رشته توسط این ماشین پذیرفته می‌شود یا خیر.

- نحوه ورودی گرفتن NFA

1. در خط اول ورودی، تعداد حالت ها و در خط بعدی یک مجموعه وارد می‌شود که حالت‌های (States) ماشین را در بر دارد و اولین عضو این مجموعه نیز حالت شروع ماشین است. (اعضای این مجموعه با فاصله از هم جدا شده اند.)
2. در خط سوم ورودی تعداد الفبا و در خط بعدی، مجموعه‌ی الفبای این ماشین به شما داده می‌شود. (اعضای این مجموعه با فاصله از هم جدا شده اند.)
3. در خط پنجم ورودی تعداد حالات پایانی و در خط بعدی، مجموعه‌ی حالت‌های پایانی ماشین به شما داده می‌شود. (اعضای این مجموعه با فاصله از هم جدا شده اند.)
4. در خط چهارم ورودی، یک عدد صحیح مثبت n داده می‌شود که بیانگر تعداد قوانین انتقال (Transition Rule) ماشین است.
5. سپس در هر یک از nخط بعدی، هر یک از قوانین انتقال به شما داده خواهد شد.

نکته :هر Transition به صورت یک سه‌تایی نمایش داده می‌شود. برای مثال qs,a,qd بیانگر این است که از حالت qs با الفبای a به حالت qd می‌رویم.

نکته $:بیانگر الفبای λ است.

در آخر یک استرینگ S به عنوان ورودی گرفته می‌شود. که کد شما باید تشخیص دهد که این رشته توسط NFA پذیرفته می‌شود یا خیر.

(در صورتی که این رشته پذیرفته می‌شد عبارت Accepted و در غیر این صورت Rejected را چاپ کنید.)

❖ برای مشاهده نمونه ورودی و خروجی به کوئرا مراجعه کنید.

2. تبدیل Regex به NFA (25 نمره)

در این مرحله شما باید با استفاده از الگوریتم تامپسون ریجکس داده شده را به NFA تبدیل کرده و سپس تشخیص دهید که یک رشته با این NFA پذیرفته می‌شود یا خیر.

برای مطالعه بیشتر و آشنایی با الگوریتم تامپسون می‌توانید از لینک های زیر کمک بگیرید

- https://medium.com/swlh/visualizing-thompsons-construction-algorithm-for-nfas-step-by-step-f92ef378581b
  در این راه ابتدا می توانید Regex را به حالت postfix در اورد و سپس به سادگی با ترکیب NFA ها آن را مانند یک عبارت ریاضی پردازش کنید.
  (برای پیدا کردن postfix یک عبارت می‌توانید از این لینک کمک بگیرید.
  https://www.geeksforgeeks.org/convert-infix-expression-to-postfix-expression/)
- https://en.wikipedia.org/wiki/Thompson%27s_construction
- https://www.youtube.com/watch?v=VbR1mGdP99s
- https://cyberzhg.github.io/toolbox/regex2nfa (با استفاده از این سایت می‌توانید ریجکس های خود را با این روش به NFA تبدیل کنید.)

در خط اول ورودی تعداد الفبا و در خط بعدی الفبای زبان مورد نظر وارد می‌شود.

در خط سوم الگوی ریجکس P وارد می‌شود و در خط چهارم نیز رشته S در ورودی به شما داده می‌شود.

شما باید در صورت پذیرفته شدن استرینگ S عبارت Accepted و در غیر این صورت عبارت Rejected را چاپ کنید.

- ❖ Regex های ورودی تنها شامل علامت های | و * و () هستند.
- ❖ ترتیب الویت های این اپراتور ها به صورت () > | > concat > * است.
- ❖ امکان دارد که این علامت ها در الفبای زبان وجود داشته باشند. در این صورت برای متمایز کردن حروف الفبا از اپراتور های ریجکس از یک \ استفاده می‌کنیم. به طور مثال در عبارت a* به این دلیل که قبل ستاره \ آمده به معنی حرف الفبای * است پس این ریجکس فقط رشته a* را پذیرش می‌کند. اما در عبارت a* قبل از ستاره

علامت \ نیامده است پس ستاره به عنوان یکی از اپراتور های ریجکس شناخته می‌شود و این ریجکس رشته های

a, aa, aaa, aaa, . . .

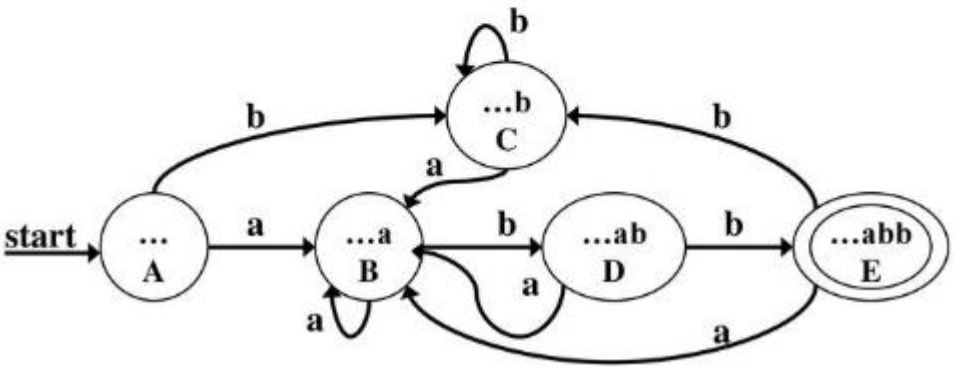
را پذیرش می‌کند.

- ❖ تضمین می‌شود که ریجکس های ورودی درستند و حرف \ درون الفبای زبان نیست.
- ❖ برای نشان دادن λ از $ استفاده می‌شود.
- ❖ برای مشاهده نمونه ورودی و خروجی به کوئرا مراجعه کنید.

3. تبدیل NFA به DFA (25 نمره)

در این مرحله NFA ایجاد شده در مرحله قبل را با استفاده از Subset Construction تبدیل به DFA می‌کنیم. یکی از ویژگی های خوب DFA ها امکان نشان دادن آن ها با تنها یک جدول از استیت ها و هر حرف الفبا است. این ویژگی باعث می‌شود که این ماشین ها بسیار سریع تر از ماشین های غیر قطعی باشند.

State	Input Symbol	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



برای اشنایی و مطالعه بیشتر درباره Subset Construction می‌توانید از لینک های زیر استفاده کنید:

- https://www.geeksforgeeks.org/conversion-from-nfa-to-dfa/
- https://www.youtube.com/watch?v=--CSVsFIDng
- http://web.stanford.edu/class/archive/cs/cs103/cs103.1202/notes/Guide%20to%20the%20Subset%20Construction.pdf

در ورودی به شما یک NFA داده می‌شود شما باید پس از تبدیل آن به DFA تعداد استیت های DFA حاصل شده را چاپ کنید.

❖ برای مشاهده نمونه های ورودی و خروجی به کوئرا مراجعه کنید.

4. ساده سازی DFA (25 نمره)

در این مرحله DFA به دست آمده از مرحله قبل را ساده سازی کنید. برای اینکار دو الگوریتم moore و Hopcraft با اردر های زمانی زمانی n^2 و $nlogn$ موجود است. برای پیاده سازی این قسمت از هرکدام از این دو الگوریتم می‌توانید استفاده کنید.(الگوریتم تدریس شده در کلاس الگوریتم moore است)

برای آشنایی و مطالعه بیشتر الگوریتم ها می‌توانید به لینک های زیر مراجعه کنید:

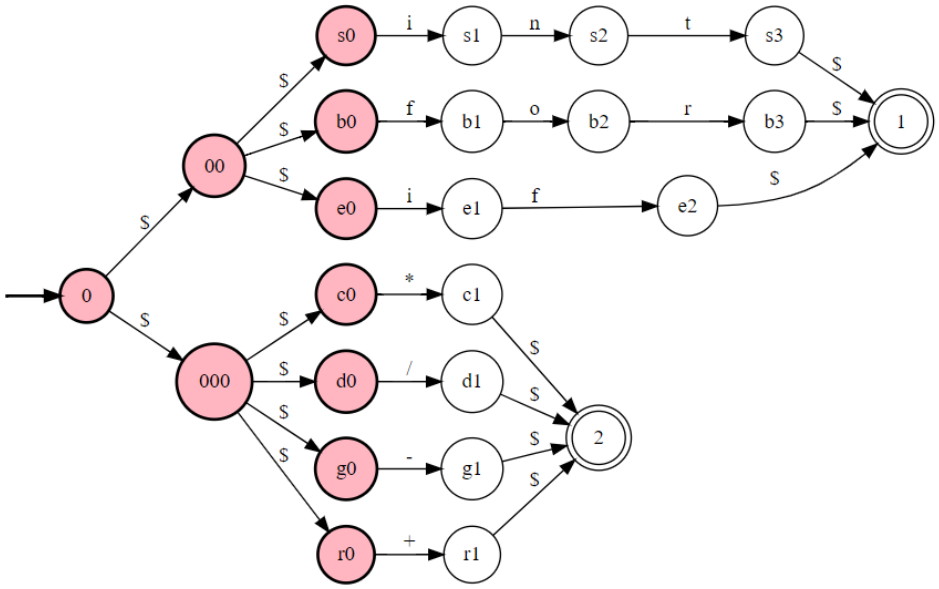
- https://www.geeksforgeeks.org/minimization-of-dfa/
- https://www.youtube.com/watch?v=DV8cZp-2VmM
- https://www.youtube.com/watch?v=7W2lSrt8r-0
- صفحه 62 کتاب لینز

در ورودی به شما یک DFA داده می‌شود. شما باید بعد از ساده سازی تعداد استیت های این DFA را خروجی بدهید.

❖ برای مشاهده نمونه های ورودی و خروجی به کوئرا مراجعه کنید.

5. ساخت Lexical Scanner Generator

در این مرحله تمام مراحل پیش را به هم وصل کرده و یک Lexical Scanner Generator می‌سازیم. ابتدا باید مقداری کد NFA خود را تغییر دهید تا در صورت پذیرش رشته استیت نهایی‌ای که رشته در آن پذیرفته شده را نیز خروجی بدهد.



به طور مثال در NFA بالا برای رشته int مقدار 1 و برای * مقدار 2 و برای test مقدار Rejected را برگرداند.

توجه کنید که این قابلیت باید بعد از تبدیل NFA به DFA و ساده کردن DFA نیز حفظ شود. حتی امکان پذیر است که در مرحله تبدیل NFA به DFA استیت های نهایی با چند توکن وجود داشته باشند. به عبارتی امکان دارد که رشته ورودی در بیش از یک استیت نهایی پذیرفته شود. به طور مثال عبارت int را در نظر بگیرید. این عبارت در اکثر کامپایلر ها می‌تواند به عنوان یک Keyword یا یک نام متغیر دسته بندی شود. در این مواقع معمولا یک اولویت بندی برای انواع مختلف دسته ها مشخص می‌شود. به طور مثال در اکثر کامپایلر ها اولویت دسته Keyword از Identifier بالا تر است. پس کلمه int معمولا در دسته keyword قرار می‌گیرد.

نکته دیگری در مورد ساده سازی DFA با تعریف جدید استیت های نهایی وجود دارد. الگوریتم های ساده سازی در صورتی که دو استیت به ازای هر ورودی به یک استیت نهایی

برسند، آن دو را یکسان فرض می‌کنند اما از آن جایی که اینجا استیت های نهایی با هم یکسان نیستند احتیاج به تعریف جدیدی از یکسانی داریم.

- **نحوه کار Lexical Scanner**

بعد از ساختن Lexical Scanner طبق ریجکس های داده شده، باید تابعی برای پردازش کد ها توسط اسکنر پیاده سازی کنید. نحوه کار یک اسکنر ساده به این صورت است که از اولین حرف کد ورودی شروع به کار می‌کند سپس در هر مرحله بیشترین تعداد حروف ممکن که به یک استیت نهایی ختم می‌شوند را دسته بندی می‌کند و اینکار را تا انتهای متن ادامه می‌دهد.

به طور مثال کلمه ای مانند forever را در نظر بگیرید. در صورتی که اسکنر در هر مرحله بیشترین تعداد حروف ممکن را نمی‌خواند این کلمه را به عنوان دو توکن for – Keyword و ever – Identifier تشخیص داده می‌شد. اما در کامپایلر های واقعی این کلمه باید به عنوان یک توکن forever – Identifier تشخیص داده شود.

همچنین در صورتی که در هر مکان هیچ زیر رشته قابل قبولی وجود نداشت مشخص است که یک خطای سینتکسی در کد وجود دارد و باید عبارت Syntax Error بازگردانده شود.
(اسکنر ها واقعی معمولا خط و تعداد کاراکتر هر خط را نیز برای پیدا کردن مکان دقیق خطا ذخیره می‌کنند ولی پیاده سازی این قابلیت در این پروژه از شما خواسته نشده است.)

در ورودی ابتدا به شما تعداد الگو های زبان مورد نظر داده می‌شود سپس در n خط بعدی n الگو به شکل Regex TokenType آمده است. اولویت بندی توکن ها از بالا به پایین است.

- در ریجکس ها برای نشان دادن کاراکتر های ’)‘ ’ ‘ ’*‘ ’(‘ ’|‘ از یک \ قبل از کاراکتر مورد نظر استفاده شده. همچنین ریجکس*(|\n|\r) به عنوان whitespace شناخته می‌شود و درنظر گرفته نمی‌شود.(این ریجکس در ورودی نیامده است و می‌توانید آن را به صورت هارد کد اضافه کنید.)

در ادامه تعداد خطوط کد و سپس کد ورودی به Lexical Scanner آمده است.

شما باید در خروجی ابتدا تعداد توکن ها و سپس توکن های این کد را به فرم

Token TokenType چاپ کنید.