DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Validación de archivos PNG

Teoría de Lenguajes
Segundo Cuatrimestre de 2021

Integrante	LU	Correo electrónico
Turco, Darío Juan Nicolás	193/19	diort.2018@gmail.com
Vigneti, Enzo Emanuel	216/18	evigneti@gmail.com
Herrera, Leon	105/18	leonazulado@gmail.com

Índice

1. Introducción

En el presente trabajo nos dedicamos a resolver el problema de, dado un archivo de texto como entrada, poder identificar si su contenido corresponde a una codificación de una partida de ajedrez en una simplificación del formato PGN (*Portable Game Notation*).

Para ello nos valdremos del lenguaje de programación `Python` y de la biblioteca `PLY` la cual nos brinda una forma sencilla de crear un parser para la gramática. Esta biblioteca trae una implementación de `lex` y de `yacc` para `Python`, los cuales son un analizador léxico y un generador de parser respectivamente.

La biblioteca `PLY` nos permite crear un parser LR(1) el cual construiremos de forma tal que, además de verificar si una cadena de texto pertenece a la gramática o no, indicara también cual es el máximo nivel de comentarios sin captura.

2. Metodología

Para el desarrollo del parser que valida las cadenas que cumplen con el formato establecido, tomamos la decisión de hacerlo de manera incremental. Tomamos partes mínimas y las fuimos expandiendo hasta reconocer cualquier cadena valida posible.

2.1. Metadatos

Lo primero que hicimos fue utilizar solo los token necesarios para los metadatos y con eso pudimos hacer que el parser reconociera únicamente cadenas que describan los metadatos.

$S \rightarrow file$

$file \rightarrow metadataInicio$

$metadataInicio \rightarrow [metadata]\ metadataSegment$

$metadataSegment \rightarrow [metadata]\ metadataSegment \mid [metadata]\ metadataSegment\ newline \mid Empty$

$metadata \rightarrow stringMeta1\ ESPACIO\ "stringMeta2"$

Donde $stringMeta1$ y $stringMeta2$ secuencian posibles caracteres para su respectiva sección de la metadata.

$file$ nos dejará parsear varias partidas en el futuro. $metadataInicio$ nos impone que haya siempre al menos un renglón de metadata antes de cada partida. $metadataSegment$ nos secuencia varios renglones de metadatos. $metadata$ nos describe los contenidos de los corchetes, con los strings, el espacio y las comillas.

2.2. Jugadas

Luego agregamos que la posibilidad de reconocer jugadas sin capturas, jaques, modificadores o comentarios. Por ultimo nos aseguramos de que las partidas terminen con un score valido, ya sea que resulte en una victoria para algún jugador o en un empate. De esta manera obtuvimos un parser que podía aceptar las partidas con metadata seguida de jugadas bien numeradas, sin ningún modificador de movimiento y con el score usado como indicador de que la partida termino.

$metadataInicio \rightarrow [metadata]\ metadataSegment\ partida$

$partida \rightarrow jugadasigJugada$

$jugada \rightarrow NUM\ .\ ESPACIO\ movimiento\ movimiento\ ESPACIO$

$sigJugada \rightarrow jugada\ sigJugada \mid score$

$movimiento \rightarrow pieza\ FILALETRA\ NUM\ ESPACIO$

$score \rightarrow NUM - NUM \mid NUM\ SLASH\ NUM - NUM\ SLASH\ NUM$

Donde $pieza$ puede ser un indicador de pieza o nada.

Viendo el número de jugada y pasándolo como atributo sintetizado a jugada, revisamos si en la producción de sigJugada se cumple que el número de sigJugada es el de jugada $+1$, o si sigJugada es un score su número es 0 y el de jugada es cualquier número. De ser así, pasamos como atributo sintetizado el número de Jugada hacía arriba en el árbol de derivación.

En el movimiento nos cercioramos de que el número de la casilla esté entre 1 y 9

2.3. Modificadores de Movimientos y Enroque

Una vez obtenido eso, agregamos la posibilidad de poner modificadores a los movimientos, los cuales podían ser jaque(+), mate(++), maravillosa jugada(!), mala jugada(?). En nuestra implementación decidimos que, en caso de haber un jaque o mate y un modificador de buena o mala jugada, el carácter + debe ir primero que el modificador de buena o mala jugada. Además en esta iteración agregamos la posibilidad de que un movimiento sea un enroque.

$movimiento \rightarrow pieza\ casillas\ mate\ calidad\ ESPACIO \mid O-O\ ESPACIO \mid O-O-O\ ESPACIO$

$casillas \rightarrow NUM\ captura\ FILALETRA\ NUM \mid EQUIS\ FILALETRA\ NUM$
$\mid FILALETRA\ NUM\ destinoPosible \mid FILALETRA\ captura\ FILALETRA\ NUM$

$destinoPosible \rightarrow captura\ FILALETRANUM \mid Empty$

Donde $casilla$ representa las distintas posibilidades de que casillas aparecen en el movimiento: origen, captura y destino. $captura$ puede o no contener una $EQUIS$. $mate$ puede o no contener un jaque $(+)$ o un mate $(++)$. $calidad$ denota si el movimiento fue comentado con un !, ? o no.

2.4. Comentarios

Luego, agregamos la posibilidad de poner comentarios, los cuales pueden tener cualquier contenido dentro. Estos comentarios pueden estar delimitados por paréntesis o por llaves, sin embargo sea cual sea su delimitador estos tienen que estar bien balanceados. Además en esta iteración agregamos que si el comentario corresponde al primer movimiento puede seguir el numero de jugada con tres puntos.

$jugada \rightarrow$
$NUM.\ ESPACIO\ movimiento\ primerComentario\ movimiento\ segundoComentario\ ESPACIO$

$primerComentario \rightarrow comentario\ ESPACIO\ tresPuntos \mid Empty$

$segundoComentario \rightarrow comentario\ ESPACIO \mid Empty$

$tresPuntos \rightarrow NUM\ ...\ ESPACIO \mid Empty$

$comentario \rightarrow (contenidoComentario) \mid \{contenidoComentario\}$

$contenidoComentario \rightarrow palabraComentario\ ESPACIO\ contenidoComentario \mid palabraComentario$

Donde $palabraComentario$ secuencia los caracteres que pueden aparecer en las palabras del comentario o ser un comentario anidado.

En $tresPuntos$ pasamos como atributo sintetizado el valor del número de jugada, hacemos lo mismo en $primerComentario$ para pasarlo a $jugada$, donde se compara con el numero de jugada y nos cercioramos que sean iguales.

Asumimos que los comentarios no podían tener un número arbitrario de espacios.

Para contar el máximo nivel de comentarios anidados, usamos un atributo sintetizado, $nivel$, para cada palabra del comentario. Toda producción lo tiene seteado en 1 excepto un comentario anidado. Estos calculan su atributo $nivel$ sumándole 1 al nivel del comentario padre. Luego comparamos el nivel de la palabra producida con el del contenido que le siguió (de existir) y pasamos el máximo entre ambos a quien los produjo.

2.5. Capturas en comentarios

Finalmente nos encargamos de verificar si un comentario tiene capturas en su interior y que nivel tienen, con eso podemos calcular el máximo nivel sin captura de cada comentario.

$$
\begin{aligned}
palabraComentario \rightarrow\ & caracteresnormales\ stringComentario \\
| \ & x\ stringComentario \\
| \ & PIEZA\ seguimientoPieza \\
| \ & FILALETRA\ seguimientoOrigenCasilla \\
| \ & NUM\ seguimientoOrigenNum \\
| \ & comentario
\end{aligned}
$$

$$
stringcomentario \rightarrow contenidoStringComentario\ stringComentario \mid Empty
$$

$$
contenidoStringComentario \rightarrow caracteresnormales \mid x \mid FILALETRA \mid PIEZA \mid NUM
$$

$$
\begin{aligned}
seguimientoPieza \rightarrow\ & FILALETRA\ seguimientoOrigenCasilla\ Empty \\
| \ & NUM\ seguimientoOrigenNum \\
| \ & caracteresnormales\ stringComentario \\
| \ & x\ stringComentario \\
| \ & PIEZA\ stringComentario \\
| \ & Empty
\end{aligned}
$$

$$
\begin{aligned}
seguimientoOrigenCasilla \rightarrow\ & x\ seguimientoCaptura \\
| \ & NUM\ seguimientoOrigenNum \\
| \ & caracteresnormales\ stringComentario \\
| \ & FILALETRA\ stringComentario \\
| \ & PIEZA\ stringComentario \\
| \ & Empty
\end{aligned}
$$

$$
\begin{aligned}
seguimientoOrigenNum \rightarrow\ & X\ seguimientoCaptura \\
| \ & caracteresnormales\ stringComentario \\
| \ & PIEZA\ stringComentario \\
| \ & NUM\ stringComentario \\
| \ & FILALETRA\ stringComentario \\
| \ & Empty
\end{aligned}
$$

$$
\begin{aligned}
seguimientoCaptura \rightarrow\ & FILALETRA\ NUM\ mate\ seguimientoMovConCaptura \\
| \ & caracteresnormales\ stringComentario \\
| \ & EQUIS\ stringComentario \\
| \ & PIEZA\ stringComentario \\
| \ & NUM\ stringComentario \\
| \ & Empty
\end{aligned}
$$

$seguimientoMovConCaptura \rightarrow contenidoStringComentario\ stringComentario \mid Empty$

Con las producciones de seguimiento, se explora la posibilidad de que la palabra que parseamos sea un movimiento. Las opciones son que se empiece con una pieza, un número, una letra para fila o una captura. A partir de estos símbolos empezamos nuestro seguimiento, donde buscamos los símbolos que deberían seguir para que sea un movimiento con captura. Si en algún momento se desvía de lo que buscamos, usamos $stringComentario$ para parsear el resto de la palabra, ya que no nos importa lo que sea, ya podemos decir que no es un movimiento con captura.

Luego de identificar que es un movimiento y tiene una captura, con $seguimientoMovConCaptura$ nos aseguramos de que no tenga ningún otro símbolo antes del $ESPACIO$ que delimita las palabras. Sintetizamos un atributo $tieneCaptura$ que indica si hay un movimiento con captura o no.

Este atributo se pasa por arriba en el árbol de producción. Las demás producciones que no conducen a un movimiento con captura también lo sintetizan diciendo que no existe esa captura. Cuando el atributo llega a la palabra, se anota su nivel como el nivel máximo de comentario sin captura (si no la hay) o anota 0 (si tiene captura). Este nuevo atributo se compara con las palabras siguientes y se sintetiza el máximo nivel sin captura o 0 de ser el caso en que en ese mismo nivel hay una palabra con captura nivel 0 y en otra hay nivel 1 (que tiene captura en el mismo nivel). Cuando la producción es un comentario anidado, se le suma 1 a ese nivel (de la misma forma que al nivel de comentario).

2.6. Nivel Máximo de Comentario sin capturas en todo el archivo

Para poder encontrar el mayor nivel de comentarios anidados sin capturas en todo el archivo buscamos primero el mayor nivel de comentarios anidados sin capturas en cada partida. Para ello agregamos varios atributos sintetizados, a los comentarios les agregamos el nivel , el máximo nivel sin captura y el numero de jugada al que corresponde. Luego los comentarios le pasan el máximo nivel a sus jugadas, después cada partida se queda con el máximo nivel de sus jugadas y, finalmente, el archivo se queda con el máximo nivel de sus partidas.

$file \rightarrow metadataInicio\ file \mid metadataInicio$

Donde $metadataInicio$ contiene el máximo nivel sin captura de su partida.

3. Problemas encontrados

A la hora de programar el parser nos encontramos con varios problemas:

- Al principio tratamos al texto de los comentarios como un string de caracteres que podría contener movimientos y comentarios, los cuales deberían estar separados por espacios. Esto presentó los problemas de esperar dos espacios entre un movimiento y un comentario y entre los símbolos delimitadores de los comentarios.

  Para solucionar esto, supusimos que un comentario debería estar compuesto de ”palabras”separadas por espacios en blanco. Estas palabras podrían ser strings de texto, movimientos o comentarios. También asumimos que nunca podrá haber dos espacios seguidos ni espacios entre los delimitadores de comentario y la primer y última palabra respectivamente.

  Implementar esto significó agregar nuevos no-terminales para agregar los espacios a un comentario que se usará en una jugada, para secuenciar las palabras y los espacios dentro de un comentario, para describir como interpretar los contenidos de un comentario.

- Llegando al final nos dimos cuenta de que no contemplamos el hecho de que un jugador puede rendirse en cualquier momento, en particular el segundo jugador es el que trae problemas, pues estábamos esperando una movida valida de su parte.

Tuvimos que cambiar la jugada para que pueda o tener un segundo movimiento y un segundo comentario, o que tenga un score porque se rinde un jugador. Para esto creamos un nuevo no-terminal, lo cambiamos en Jugada. También cambiamos la siguiente jugada para que tenga una producción vacía.

$jugada \rightarrow NUM.\ movimiento\ primerComentario\ posibleRendicion$

$posibleRendicion \rightarrow movimiento\ segundoComentario\ ESPACIO \mid score$

$sigJugada \rightarrow jugada\ sigJugada \mid score \mid Empty$

Con estos cambios agregamos el uso de numero de jugadas negativos para indicar una jugada con rendición; y debemos asegurarnos de que si hay una rendición, la siguiente jugada sea vacía.

- Luego de terminar la primer linea de metadata, teníamos que permitir que estas lineas se repitieran una cantidad indeterminada de veces, siempre espaciadas un salto de linea. Para esto en un principio pensamos en preprocesar la entrada eliminando todos los saltos de linea, sin embargo esa idea no nos pareció buena ya que le sacaba responsabilidades al parser. Para esto creamos un token llamado *newline* el cual puede aparecer o no al final de cada linea de metadata. Luego no se esperara un newline en ninguna otra parte del archivo.
- Otro problema que tuvimos a la hora de crear el parser fue que las cadenas de la metadata y la de los comentarios no pueden tener cualquier carácter, por ejemplo en la primer cadena de la metadata no se puede tener espacios y en la segunda no se puede tener comillas. Esto nos obligo a que las producciones que matchean estas partes del archivos tengan una producción en común, la cual es la producción *caracteresnormales* y luego usamos eso para crear las producciones que matchean el contenido de la metadata o de los comentario, la cual puede contener *caracteresnormales* o bien los caracteres específicos del string que estamos intentando matchear. Lo mismo surgió con los comentarios, donde tuvimos que usar los tokens que pueden iniciar un Movimiento para buscar movimientos con capturas, con lo que tuvimos que sacarlos de los caracteres normales y agregarlos a las producciones de los contenidos de los strings de metadata y comentarios.

4. Ejemplos de uso

Para usar nuestro parser se debe abrir el archivo `parser.py` con `Python 3`, luego el archivo esperara a que se le pase un archivo de texto con la cadena a parsear. Finalmente si dicha cadena es aceptada aparecerá el mensaje 'Máximo nivel sin captura: X' donde X es el máximo nivel sin capturas de la entrada.

En la carpeta entregada tenemos algunas partidas de pruebas, si se ejecuta en consola el siguiente código se puede correr una de ellas:

```
python3 parser.py
Partida > partida1.txt
Máximo nivel sin captura: 0
```

El archivo `partida1.txt` es uno adjuntado en la entrega el cual tiene en su interior una partida valida sin comentario, por lo tanto tiene máximo nivel sin captura igual a 0. El contenido es el siguiente:

```
[prueba "loca"]
[Nzscf5qWgtg~NVX "56B~n~nQIeAhy"]
[gvk7dXkliRpR "2LAkQJGhz81"]
[~NFS5lBHW4Mm~NmJsP "e4ZhVulzl"]

[GArzOdNa~nITcsbFO9ES "WUodxeqxI"]

1. a4 e3 1-0
```

También adjuntamos otros ejemplos para correr, todos se ejecutan de la misma manera, solo hay que cambiar el nombre del `txt` a correr.

5. Conclusiones

En conclusión en este trabajo aprendimos a crear un parser usando el poder de la biblioteca PLY. El cual es de tipo LR y por lo tanto sabemos que su complejidad va a ser O(n) donde n es la cantidad de caracteres de la entrada. Además esto se debe a que todos los atributos calculados se resuelven en O(1).

Adicionalmente, pensamos que un buen posible agregado para el futuro, es validar además de la gramática, que la partida que se está jugando sea valida. Para esto es necesario programar cuales son las reglas del ajedrez y verificar que los movimientos sea validos según estas reglas. Además pensamos que también estaría bueno poder visualizar la partida que se esta parseando con el objetivo de tener un entendimiento mas profundo de que se esta jugando. También es interesante poder tener partidas que tengan una cantidad indeterminada de espacios en el texto de los comentarios.

También queremos destacar que los temas vistos en la materia a lo largo del cuatrimestre fueron de gran utilidad a la hora de realizar el trabajo, en especial los temas de traducciones dirigida por la sintaxis y parsers LR. Además de que la materia nos dio un entendimiento general sobre gramáticas el cual nos permitió manejarnos con mas facilidad a la hora de agregar o cambiar producciones de la gramática.