Desencriptación Mediante Técnicas de Síntesis de Programas

Universidad Nacional Autónoma de México
Facultad de Ciencias
Síntesis de Programas

Juan Daniel Cortez Rojas
danielitos0901@ciencias.unam.mx

María Ximena Lezama Hernández
lezama@ciencias.unam.mx

27 de Noviembre de 2018

Resumen

La meta que se quiere alcanzar con este trabajo es averiguar hasta donde los principios de la síntesis de programas nos ayuda o facilita la desencriptación de las llaves mediante la síntesis por ejemplos, mas específicamente usando la técnica de síntesis de abajo hacia arriba(*Button-Up*) y centrándonos en ciertos tipos de encriptacion básicos.

1. Introducían

Como el peligro de ser descubierto era grande, solo había una forma en la que podía lograr transmitir el mensaje: esto era raspando la cera de un par de tablillas de madera plegables, escribiendo en la madera debajo de lo que Xerxes intentaba hacer, y luego cubriendo el mensaje con cera de nuevo. De esta manera, las tabletas, aparentemente en blanco, no causarían problemas con los guardias a lo largo del camino. Cuando el mensaje llegó a su destino, nadie pudo adivinar el secreto, hasta que, según tengo entendido, la hija de Cleomenes, Gorgo, que era la esposa de Leonidas, adivinó y les dijo a los demás que si raspaban la cera, encontrarían algo escrito en la madera debajo. Esto se hizo; el mensaje fue revelado y leído, y luego pasado a los otros griegos[1].

-Heródoto

La idea de cifrar mensajes o información viene de la necesidad de un canal de comunicación confiable y seguro en el cual se necesitaba que si un enemigo interceptaba el mensaje no fuera capas de saber que estaba escrito en el, pero a comparación de un aliado que sabia el código de cifrado, si llegaba a salvo dicho mensaje podría ser leído sin ningún problema por el destinatario.

El deseo de mantener el secreto ha significado que las naciones hayan desarrollado departamentos de creación de códigos, que eran responsables de garantizar la seguridad de las comunicaciones al inventar e implementar los mejores códigos posibles para el cifrado de mensajes, de manera que se intentaba que la información no pudiese ser leída mas que por quienes conocían dicho código.

Del otro lado de la moneda se encuentran los descifradores, los cuales se dedican a descifrar los mensajes codificados y obtener la información que se escondía tras palabras o símbolos sin aparente significado. Cuando los descifradores desarrollan una nueva forma estrategia que revela alguna debilidad de un código, luego el código ya no es útil ya que se conoce la forma de descifrarlo de forma sencilla.

En el presente articulo se busca la obtención de el código de cifrado usado en ciertos ejemplos, de los cuales conocemos una serie de mensajes y el resultado de codificación codificarlos bajo un cierto al-

[1] Marcus Tullius Cicero, Fragmento extraido de *The Histories*

goritmo de cifrado, para fines de experimentación se usaran dos algoritmos de cifrado muy sencillos los cuales son *El cifrado César* y *El cifrado afín*.

2. **Cifrados**
   Se usaran dos tipos de algoritmos de cifrado básico los cuales son *El cifrado César* y *El cifrado afín*, a continuación desarrollaremos una breve descripción de los algoritmos.

   - *El cifrado César*
     también conocido como cifrado por desplazamiento, código de César o desplazamiento de César, es una de las técnicas de cifrado más simples y más usadas. Es un tipo de cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Por ejemplo, con un desplazamiento de 3, la A sería sustituida por la D (situada 3 lugares a la derecha de la A), la B sería reemplazada por la E, etc. Este método debe su nombre a Julio César, que lo usaba para comunicarse con sus generales.

     El cifrado César muchas veces puede formar parte de sistemas más complejos de codificación, como el cifrado Vigenére, e incluso tiene aplicación en el sistema ROT13. Como todos los cifrados de sustitución alfabética simple, el cifrado César se descifra con facilidad y en la práctica no ofrece mucha seguridad en la comunicación.

   - *El cifrado afín*
     Es un tipo de cifrado que usa el mismo alfabeto para el texto plano que para el texto cifrado que utiliza una expresión matemática para determinar la posición en el alfabeto (según el orden normal) del carácter del texto cifrado asociado. En este cifrado la clave viene definida por dos valores numéricos a y b. Sea m el tamaño de alfabeto del texto plano.

     Para definir que carácter del alfabeto sustituye a cada carácter se aplica la fórmula $(ax + b)mod(m)$, donde x es la posición del carácter al que le estamos buscando sustituto. El resultado se usa como índice en el orden predefinido del alfabeto. Por ejemplo con a=5 y b=15 y el alfabeto del castellano de m=27 letras. La "a" se convertirá en $(5 * 1 + 15)mod(27) = 20$. Por tanto el carácter asociado será el que ocupa la posición 20, la "s".

Como podemos ver las dos codificaciones funcionan de una manera muy simple usando una codificación que asigna un numero a una letra conforme a la posición en que se encuentre la letra en orden lexicográfico además de operaciones básicas sobre números enteros las cuales son suma y el operador de modulo, por lo que esto restringe de forma muy especifica cuales son las operaciones y elementos sobre los cuales se hará la búsqueda del programa que decodifique los mensajes.

Por ende también plantearemos las siguientes restricciones sobre la búsqueda:

- Se usara el alfabeto ingles el cual cuenta con un pequeño numero de caracteres(26 en su totalidad), ya que esto nos brinda mucha facilidad en la codificación y un rango mas pequeño de búsqueda de no admitir caracteres especiales como son la tilde en las letras además de la Virgulilla para la ñ.
- limitaremos el tamaño de búsqueda a 25 iteraciones, ya que el programa de codificación no debería ser tan largo

Teniendo en cuenta estas restricciones podemos empezar a planear la implementación del sistema de síntesis por ejemplos usando la técnica de abajo para arriba.

Para fines de practicidad se usara el lenguaje de programación *HASKELL* el

cual gracias a su sistema de coincidencia de patrones y de construcciones de gramáticas simples que nos facilitara mucho las cosas para manejar las producciones de la gramática que generara nuestro programa.

3. **Síntesis por ejemplos y Técnica de abajo hacia arriba(Button-Up)**

   La idea de construir programas a través de ejemplos se remonta a la década de 1970, cuando Patrick Winston en el Instituto Tecnológico de Massachusetts(MIT) publicó el trabajo seminal *"Aprendizaje de descripciones estructurales a partir de ejemplos*[2]*"*. En el cual ni si quiera había un acercamiento a automatizar la búsqueda de programas que resolvieran un problema en especifico.

   La síntesis por ejemplos se basa en restringir la restricción del espacio de búsqueda de programas y de la posterior verificación de correctud de los programas candidatos a ser una solución al problema. Este método usa una serie de ejemplares para el problema y al mismo tiempo la solución al problema enfocado solo a estos ejemplares, así tenemos una serie de parejas de Ejemplares-Resultado además de la definición formal de la gramática usada para los programas, es decir de las reglas de producción que nos permitirán crear nuevos programas a partir de los ya conocidos. Teniendo a los candidatos para resolver el problema y los ejemplares con su solución se procede a verificar que tomando como entrada cada uno de los ejemplares y cada uno de los candidatos de como resultado la solución ya conocida con lo que diremos que hemos encontrado al candidato que resuelva nuestro problema, en caso contrario con los candidatos ya conocidos se generan nuevos programas para su posterior verificación. Se hace esto de forma continua limitando el espacio de búsqueda haciendo que se puedan generar nuevos programas solo una cierta cantidad de veces, si para antes de eso no se ha encontrado un candidato viable entonces terminamos el proceso y no tenemos una solución al problema.

   EL algoritmo de síntesis de abajo hacia arriba(Buttom-Up) trabaja de una manera muy fácil ya que construye todos y cada uno de los posibles programas que se pueden generar con la gramática especificada, empieza a generar programas basándose en las producciones que nos llevan a un estado terminal. Como podemos ver esto trae un inconveniente ya que el espacio de búsqueda de programas crece de forma enorme en cada iteracionese la nueva gegeneracióne programas, incluso para gramáticas pequeñas el espacio de bubúsquedas demasiado grande.

   Una técnica usada para limitar o recortar el espacio de búsqueda es eliminar programas que sean *"equivalentes"*. Pero a que nos referimos con esto? se toma como programa equivalente a cada uno de los programas que con una misma entrada nos devuelven el mismo resultado, así no importa si se usa uno u otro el resultado es el mismo.

   Para el diseño del algoritmo necesitaremos dos funciones auxiliares muy importantes, las cuales a grandes rasgos hemos explicado su funcionamiento, así pues necesitaremos una función *"grow"* y una *"elimEquiv"* las cuales se dedicaran a generar la nueva iteración de programas usando los que ya se conocen y eliminar los programas que sean equivalentes, con ayuda de estas dos funciones se puede implementar el algoritmo de síntesis.

4. **Implementación**

   La idea generar para atacar el problema planteado en las Secciones previas, es implementando una serie de funciones que ya fueron planteadas previamente, de manera

[2]Patrick H. Winston, Learning Structural Descriptions From Examples, 1970

que al recibir una tupla de valores ((holasoyunmensaje), (krodvrbxqphqvdmh))[3] la cual la primera entra sería nuestra entrada [4] del programa y la segunda tupla la salida, nos regrese un programa que nos codifique la entrada de manera que se parezca a la salida planteada.

La gramática de la cual partiremos en una gramática muy sencilla la cual solo contará de de letras, sin importar si son mayúsculas o minúsculas, las cuales tienes una asociación numérica conforme a su posición en el alfabeto de manera que al codificar simplemente estemos usando números lo cual hará más sencillo evaluar y un dato tipo *Exp* el cual será una expresión llamada PLUS, la cual cuando se evalué solo sea la suma de dos numero enteros modulo el numero de letras dentro de nuestro alfabeto, en este caso también delimitamos nuestras letras a solo el alfabeto ingles y trabajar directamente con solo 26 letras. En el caso del cifrado Afín es análogo al cifrado Cesar, mas que con la excepción que las expresiones estarán conformadas no solo por PLUS, si no que agregaremos la función MOD, la cual calculará el modulo de una suma siguiendo la descripción del cifrado que queremos que nos devuelva la computadora.

Las funciones a implementar son: *isCorrect*, *elimEquiv*, *grow* y *buttomup*. *isCorrect* es una funcion la cual comparará los programas generados con la salidas de nuestro programa uno a uno. La funcion tomara valores una lista de resultados que es el segundo elemento de la tupla, y los resultados de nuestros programas generados (evaluados), los cuales si coincide, regresamos que se cumple la hipótesis por lo que es *True* de otra manera regresa *False*.

Otra función a implementar será *elimEquiv* la cual tomará una lista de programas, la cual va de una lista de *Exp*, y de una lista la cual sera la entrada pero codificada al numero de la posición en la que se encuentra cada carácter de la cadena, eliminaremos, todos aquellos programas que una vez interpretados con las entradas sean iguales, con ayuda de una función auxiliar que elimine los repetidos de una lista.

Con esas dos funciones nos queda solo definir como operará nuestro*grow* y *buttomup*. El *grow* será una función que va de una lista de programas *numops*, la lista de constantes las cuales serán nuestras letras (recordemos que las letras tienen una codificación por la posición dentro del alfabeto) y regresará una lista de programas que representan el nuevo nivel de operaciones que se realizaron. La función *grow* identificará en caso de tener dos expresiones, de que expresión se trata y agregará un nuevo nivel agregando la expresión con los elementos de la lista de constantes.

Finalmente el *buttomup* será la función que utilice todas las anteriores, la cual va de un entero el cual es el globalBnd, la lista de expresiones que variará conforme al cifrado, una lista de constantes que serán nuestras letras, y devolverá una lista con las expresiones que resuelvan el cifrado. La función la realizaremos el *grow* de la lista de expresiones y las constantes. A esa lista resultante eliminaremos los equivalentes, si la respuesta es correcta regresamos la lista que llevamos de programas la cual concatenaremos a la llamada recursiva de la misma función con el globaBnd incrementado en uno, en caso contrario hacemos la llamada recursiva al *grow* con el globalBnd incrementado en uno.

[3] En este ejemplo se uso cifrado Cesar

[4] Desde un principio trabajamos bajo el supuesto que no existen caracteres especiales, espacios, y/o números en nuestras E/S

References

[1] https://www.tutorialspoint.com/es/ipv4/ipv4_address_classes.htm