Expresiones regulares (autómatas) **con variables y sus aplicaciones**

En este artículo estudiamos expresiones regulares que utilizan tanto símbolos de un alfabeto finito como variables. Tales variables se interpretan como símbolos en el alfabeto. Además, consideramos dos tipos de lenguajes definidos por estas expresiones: bajo la semántica *existencial,* una palabra pertenece al lenguaje de la expresión con variables E si pertenece al lenguaje definido por alguna expresión que se puede obtener desde E al reemplazar variables por símbolos; bajo la semántica *universal,* una palabra pertenece al lenguaje de la expresión con variables E si pertenece al lenguaje definido por toda expresión que se puede obtener desde E al reemplazar variables por símbolos. Tales lenguajes son regulares, y además demostramos que aparecen naturalmente en varias aplicaciones como consultar bases de datos de grafos con información incompleta y el análisis de programas. Para proveer un análisis computacional más sólido, mencionamos también ciertos resultados teóricos que ayudan a entender el comportamiento de las expresiones regulares con variables, así como la complejidad de algunos de los problemas de decisión más básicos asociados con ellas.

Organización: en la siguiente sección introducimos las definiciones básicas de lenguajes regulares y autómatas que son

Pablo Barceló
Profesor Asistente DCC, Universidad de Chile. Ph.D. in Computer Science, University of Toronto (2006); Magíster en Ciencias de la Computación, Pontificia Universidad Católica de Chile (2002); Ingeniero en Electricidad, Pontificia Universidad Católica de Chile. Áreas de interés: Bases de Datos, Lógica para la Ciencia de la Computación, autómata.
pbarcelo@dcc.uchile.cl

necesarias para entender la investigación realizada. Luego, motivamos la introducción de las expresiones regulares con variables con dos aplicaciones diferentes: análisis de programas y bases de datos de grafos con información incompleta. A partir de éstas definimos las dos semánticas para las expresiones regulares con variables: Existencial y Universal. En la siguiente sección listamos y explicamos las propiedades computacionales más importantes de estas expresiones, y finalmente discutimos las conclusiones de nuestro trabajo, así como los problemas relacionados que deseamos estudiar a futuro.

EXPRESIONES REGULARES Y AUTÓMATAS

Las *expresiones regulares* son un método gramatical ampliamente utilizado en computación para especificar conjuntos (posiblemente infinitos) de palabras sobre un alfabeto finito. Su objetivo básico es servir como una herramienta concisa y flexible para la especificación de patrones de texto.

De forma de hacer más fácil la presentación, a partir de ahora reducimos nuestro estudio al alfabeto binario A = {0,1}. Sin embargo, cabe notar que todo lo que mencionemos acerca de las expresiones regulares, y sus variantes, a lo largo del artículo es independiente de esta elección inicial. Es decir, todas las propiedades que mencionemos son también ciertas si trabajamos con cualquier alfabeto A′ distinto de A.

El conjunto de expresiones regulares se halla definido recursivamente por la siguiente gramática:

$$\varphi, \varphi' := \varnothing \mid \varepsilon \mid 0 \mid 1 \mid \varphi \cup \varphi' \mid \varphi \cdot \varphi' \mid \varphi*$$

Para facilitar la simplicidad de notación, el símbolo · comúnmente se elimina de las expresiones regulares (ya que es fácilmente distinguible dentro del contexto, los lugares en que aparece).

Como mencionamos anteriormente, cada expresión regular φ define un conjunto de palabras (es decir, un *lenguaje*) L(φ) sobre el alfabeto A. Tal lenguaje se define recursivamente como sigue:

Casos base:

1) $L(\varnothing) = \varnothing$. Es decir, ∅ denota al conjunto vacío.
2) $L(\varepsilon) = \{\}$. Esto es, ε denota a la palabra vacía.
3) $L(0) = \{0\}$ y $L(1) = \{1\}$. Esto es, el 0 denota a la palabra con un único símbolo 0, y análogamente para el 1.

Casos inductivos:

4) $L(\varphi \cup \varphi') = L(\varphi) \cup L(\varphi')$. Es decir, como era de esperar el símbolo ∪ representa la unión de lenguajes.
5) $L(\varphi \cdot \varphi') = L(\varphi) \cdot L(\varphi')$, donde

   $L(\varphi) \cdot L(\varphi') = \{ww' \mid w \in L(\varphi) \text{ y } w' \in L(\varphi')\}$.

Esto quiere decir que el símbolo · representa la *concatenación* de lenguajes.

6) $L(\varphi*)$ se define como la unión de todos los lenguajes $L(\varphi)^i$, para i mayor o igual a 0, donde los lenguajes $L(\varphi)^i$ se definen inductivamente de la siguiente forma:

   $L(\varphi)^0 := \{\}$ y $L(\varphi)^{i+1} = L(\varphi)^i \cdot L(\varphi)$.

Esto es, el símbolo *, que se llama usualmente *clausura de Kleene*, representa la *concatenación* de un número arbitrario de veces de un lenguaje consigo mismo.

Por ejemplo, la expresión regular (01)* representa el lenguaje de todas las palabras de la forma 0101 ... 0101, mientras 0* ∪ 0*10* ∪ 0*10*10* representa el lenguaje de las palabras con *a lo más dos 1s*. Por supuesto, no todo conjunto de palabras puede ser representado por una expresión regular (es decir, no todo lenguaje es regular). Uno de los ejemplos más paradigmáticos es el lenguaje que contiene a todas las palabras de la forma $0^n 1^n$, para $n > 0$. Esto es, el lenguaje de todas las palabras que comienzan con una secuencia de 0s y luego siguen con una secuencia de 1s del mismo largo. Intuitivamente, este lenguaje no es regular porque las expresiones regulares carecen de la habilidad de "contar", es decir de especificar que hay la misma cantidad de 0s que de 1s en una palabra.

Por razones obvias, el conjunto de los lenguajes definidos por expresiones regulares se llaman *lenguajes regulares*. Aunque es probable que los lenguajes regulares hayan sido creados (¿o descubiertos?) por varios investigadores al mismo tiempo, usualmente se cita como su inventor, en los años cincuenta, al lógico estadounidense Stephen Kleene. También fueron estudiados por el famoso lingüista Noah Chomsky, quien los enmarcó en su jerarquía de gramáticas formales, siendo los lenguajes regulares uno de sus escalones más bajos.

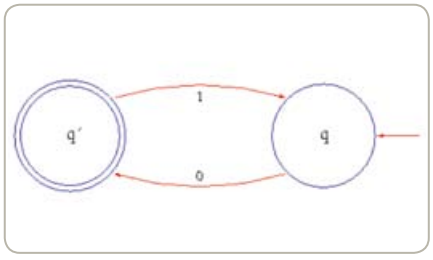
Una de las propiedades más importantes de los lenguajes regulares es su clausura con respecto a las combinaciones Booleanas usuales. Esto quiere decir que los lenguajes regulares son cerrados bajo unión, intersección y complemento. Note que la primera operación (unión) pertenece explícitamente a la gramática, y, por tanto, demostrar que los lenguajes regulares son cerrados bajo unión es trivial. Por otro lado, ni la intersección ni el complemento son operaciones en la gramática, y de hecho probar que los lenguajes regulares son cerrados bajo estas operaciones requiere una demostración no trivial.

Otra buena propiedad de los lenguajes regulares es su "robustez", en el sentido que pueden ser caracterizados de muchas formas diferentes, e.g. como gramáticas, algebraicamente, en términos del poder expresivo de ciertas lógicas, etc. Probablemente, junto con la caracterización gramatical de los lenguajes regulares en términos de expresiones regulares, que ya hemos visto, la otra caracterización

más famosa es la algebraica, en términos de *autómatas*. Esta equivalencia es particularmente importante, ya que en la práctica muchas veces es conveniente construir el autómata equivalente a una expresión regular para poder determinar las palabras que la satisfacen.

Recordemos que un autómata es una tupla $T = (Q,q,F,\delta)$, donde (1) Q es un conjunto finito de *estados,* (2) q es un estado particular en Q denominado estado *inicial,* (3) F es el subconjunto de Q que contiene a los estados *finales,* y (4) δ es una función parcial de *transición,* que se define desde $Q \times \{0,1\}$ en Q. El autómata T *acepta* una palabra $w = a_1\, a_2 \cdots a_n$ sobre alfabeto {0,1} si el autómata puede "correr" sobre w desde el estado inicial a uno final siguiendo las reglas dictadas por la función de transición. Formalmente, si existe función $\rho : \{0,1,\ldots,n\} \rightarrow Q$ tal que (a) $\rho(0) = q$, (b) $\rho(i+1) = \delta(\rho(i),a_{i+1})$, para todo i entre 0 y (n-1), y (c) $\rho(n) \in F$. El lenguaje aceptado por el autómata T se define como el conjunto de todas las palabras aceptadas por T.

Por ejemplo, el siguiente autómata acepta exactamente el lenguaje regular 0(10)*:



Los estados son los círculos azules y las transiciones son las flechas en rojo. El estado inicial es q, denotado por la flecha de entrada sin etiqueta. El estado final es q', denotado por el doble círculo. Desde el estado q la única posible transición es al estado q', pero sólo si el autómata está leyendo el símbolo 0 en la palabra. De la misma forma, desde el estado q' la única posible transición es al estado q, pero sólo si el autómata está leyendo el símbolo 1 en la palabra.

El hecho que un autómata reconozca un lenguaje que es regular no es una coincidencia, ya que el famoso teorema de Kleene establece la equivalencia de los lenguajes regulares y aquellos aceptados por autómatas.

> ***Teorema de Kleene:*** *sea L un lenguaje. Entonces las siguientes afirmaciones son equivalentes:*
>
> *1) L es regular (es decir, L = L(φ) para alguna expresión regular φ).*
>
> *2) L es aceptado por algún autómata T.*

Es interesante ver que, en cierto sentido, el Teorema de Kleene expresa que los lenguajes regulares pueden ser entendidos tanto *declarativamente* - aquellos que son *especificados* por las expresiones regulares - como *proceduralmente* - aquellos que son *aceptados* por los autómatas. Este tipo de equivalencias aparecen en distintas áreas de la computación y son comúnmente muy relevantes. Por un lado, la parte declarativa permite al usuario especificar lo que desea, mientras la parte procedural ayuda en la implementación y optimización del sistema. Un ejemplo paradigmático es el de los lenguajes de consulta para bases de datos relacionales: el usuario especifica sus consultas en SQL, mientras el sistema las ejecuta utilizando el álgebra relacional.

Es importante destacar que las expresiones regulares y su contraparte algebraica, los autómatas, tienen variadas aplicaciones en distintas áreas de la computación. Éstas incluyen, entre innumerables otras, lingüística computacional [Mar05], compiladores [ALSU06], búsqueda en texto [CR03, NR07], datos semiestructurados [Bun97, BSV99], verificación de software [VW86], análisis de programas [NNH10], etc.

EXPRESIONES REGULARES CON VARIABLES

Nuestra investigación trata sobre el tema de las Expresiones Regulares con Variables (ERVs), las que permiten describir sucintamente otras expresiones regulares más complejas. Para motivar el problema comenzaremos con una aplicación de estas expresiones en el análisis de programas [LRYS04].

Utilizamos el alfabeto que contiene las operaciones sobre los elementos del programa, e.g. variables, punteros, archivos, etc. (por ejemplo, **def**, para definir una variable, **use**, para usarla, **malloc**, para localizar un puntero, entre otros). A estos símbolos comúnmente sigue una variable; e.g. **def**(x) significa definir la variable x.

En este caso las ERVs sirven para describir y detectar cierto tipo de *bugs* (i.e. comportamiento indeseado) en el programa. Por ejemplo, la expresión:

$$(\neg \textbf{def}(x))^* \ \textbf{use}(x),$$

donde $\neg$**def** es una abreviación para el complemento de la expresión regular def, que identifica aquellas variables que han sido utilizadas sin antes ser definidas. En general, dada una ERV E y un programa P, nos interesa encontrar los posibles valores de las variables que hacen que E se satisfaga en P (estos valores corresponden a los *bugs* de P). En términos más formales, nos interesan aquellas palabras que son definidas por *alguna* expresión regular sin variables que se pueda obtener desde E reemplazando a las variables por símbolos del alfabeto. Es decir, en este caso interpretamos existencialmente la semántica de la expresión regular con variables.

Una segunda aplicación de las ERVs viene del área de *Bases de Datos de Grafos* (GDBs). En su forma más simple una GDB es un par (V,E), donde V es un conjunto finito de

vértices y E es un subconjunto de triples en V × A × V, donde A es un alfabeto finito. Es decir, E es un conjunto de arcos etiquetados en A. Por ejemplo, sea C el conjunto de todas las ciudades del mundo y F el conjunto de todos los triples de la forma (c,a,c′), tal que c y c′ son ciudades en C y a es el nombre de una aerolínea que vuela en forma directa desde c hasta c′. Entonces H = (C,F) es una GDB.

El lector familiarizado con los temas de autómatas notará que una GDB no es más que un autómata *no determinista*, donde los vértices corresponden a los estados y las arcos a las transiciones. Como es bien sabido, estos autómatas no entregan mayor poder expresivo a su versión determinista (la que fue definida en la sección anterior). Es decir, todo lenguaje definido por un autómata no determinista puede también ser definido por un autómata, y por tanto también por una expresión regular. Esto quiere decir que las GDBs son, en esencia, expresiones regulares.

Como es usual en cualquier modelo de base de datos, las GDBs vienen acompañadas de su propio lenguaje de consulta, el que permite extraer información a partir de la información almacenada. Una de las características más importantes de las GDBs, y que en particular la hacen distinas de las bases de datos relacionales, es que en ellas estamos tan interesados en consultar la "topología" de los datos como los datos mismos. En particular, muchas veces estamos interesados en "navegar" el grafo, es decir, en recorrerlo recursivamente desde un lado a otro. Una típica consulta de esta forma es verificar si un par de nodos se haya unido por una palabra en un lenguaje regular dado. Por ejemplo, continuando con la GDB H = (C,F), uno podría querer verificar si existen vuelos (no necesariamente directos) entre Santiago y Toronto que sólo utilicen a LAN o Air Canada. Para ello se debe verificar si existe un camino desde Santiago a Toronto en H tal que la concatenación de las etiquetas en ese camino pertenece al lenguaje regular definido por (LAN ∪ Air Canada)*.

> Dado que la mayoría de los problemas de decisión clásicos de autómatas y expresiones regulares se vuelven intratables en la presencia de variables, es importante en el futuro desarrollar heurísticas que permitan trabajar con ellas en ciertos contextos importantes.

En la mayoría de las aplicaciones modernas, en que los datos son constantemente transferidos, intercambiados y tienen niveles no menores de incertidumbre, es necesario proveer un modelo flexible que permita especificar que ciertos datos son desconocidos o simplemente no están disponibles. Este modelo se denomina de información *incompleta*. Usualmente en este escenario buscamos las respuestas a las consultas que son independientes de la interpretación faltante. Tales respuestas se denominan *certeras*, ya que son invariantes frente a la reparación que establezcamos sobre la base de datos.

En un reciente artículo [ABR11] hemos comenzado el estudio de este tema sobre GDBs. Una de las mayores fuentes de incompletitud en este escenario es la pérdida de información estructural, principalmente la pérdida de la información contenida en la etiqueta de un arco. Como ya mencionamos, cada GDB puede ser descrita por un autómata, y por tanto por una expresión regular. Eso quiere decir que las GDBs con información estructural incompleta pueden ser descritas por expresiones regulares con variables (donde las variables representan la información estructural faltante).

Recuerde que en la presencia de información incompleta estamos interesados en las respuestas certeras a una consulta. En particular, pensando en el lenguaje de consulta específico de las GDBs, quisiéramos detectar si dos nodos se hayan unidos por una palabra en un lenguaje regular dado, independiente de la interpretación de la información faltante (las variables). Esto quiere decir que, al contrario del caso anterior, en este escenario nos interesa una interpretación *universal* de las ERVs que representan nuestras GDBs con información parcial.

Podemos entonces definir la semántica de una ERV E. Primero debemos entender cómo interpretar sus variables. Para ello ocupamos el concepto de valuación, que no es más que una función $\eta : V \to A$, donde V es el conjunto de variables mencionadas en E. Esto es, η es una función que a cada variable le asigna una letra del alfabeto. Definimos además la aplicación de η sobre E, denotado $\eta(E)$, como la expresión regular *sin variables* que se obtiene desde E al reemplazar simultáneamente cada variable x en E por el símbolo $\eta(x)$.

Dada una expresión regular E con variables, definimos su semántica tanto en términos

existenciales como universales (motivadas por las aplicaciones mencionadas más arriba) como sigue:

a) La semántica existencial de E es el lenguaje $L_{\cup}(E)$ que se define como la unión de todos los lenguajes de la forma $L(\eta(E))$, donde η es una valuación de E. Esto es, una palabra pertenece a $L_{\cup}(E)$ si y sólo si es aceptada por alguna expresión regular E' sin variables que se puede obtener desde E al reemplazar simultáneamente a sus variables por símbolos del alfabeto.

b) La semántica universal de E es el lenguaje $L_{\cap}(E)$ que se define como la intersección de todos los lenguajes de la forma $L(\eta(E))$, donde η es una valuación de E. Esto es, una palabra pertenece a $L_{\cap}(E)$ si y sólo si es aceptada por toda expresión regular E' sin variables que se puede obtener desde E al reemplazar simultáneamente a sus variables por símbolos del alfabeto.

Por ejemplo, sea E la expresión $(0 \cup 1)^*$ xy $(0 \cup 1)^*$. Entonces, la palabra 00 pertenece a $L_{\cup}(E)$, como lo atestigua la valuación $\eta(x) = \eta(y) = 0$. Por otro lado, la palabra 10011 pertenece a $L_{\cap}(E)$ y no hay palabra de largo menor que también pertenezca a este lenguaje. Esto se debe al hecho que xy es una subexpresión de E, y que, por tanto, si una palabra w pertenece a $L_{\cap}(E)$ entonces w contiene a toda palabra de largo 2 como subpalabra. Es posible demostrar por una simple enumeración de casos que no hay palabra más corta que 10011 que tenga esta propiedad (y note que 10011 sí la tiene).

Note que cada ERV E utiliza un número finito de variable y que, por tanto, el número de posibles valuaciones para E es también finito. Esto nos permite hacer una importante primera observación: dado que los lenguajes regulares son cerrados bajo unión e intersección, entonces tanto $L_{\cup}(E)$ como $L_{\cap}(E)$ son también lenguajes regulares. Esto quiere decir que las ERVs no agregan poder expresivo a las expresiones regulares sin variables. Pero como veremos en la próxima sección, lo que sí aportan es la capacidad de expresar algunos lenguajes de forma más sucinta.

Es importante hacer notar que ésta no es la única semántica posible para las expresiones regulares con variables. De hecho, hace varias décadas la comunidad de lenguajes formales viene estudiando la clase de los *patrones* [Sal03], que son nada más que concatenaciones de letras y variables. En el caso de los patrones, sin embargo, las valuaciones permiten reemplazar variables no sólo por letras sino también por palabras arbitrarias en el alfabeto (bajo una semántica existencial). Esto hace que los patrones, al contrario de las ERVs, puedan expresar propiedades mucho más allá del mundo regular. Por ejemplo, el patrón xx define el conjunto de todas aquellas palabras de la forma ww, donde w es una palabra cualquiera. Este lenguaje no es regular y, de hecho, ni siquiera es *context-free*. Este aumento en expresividad conlleva, como es de esperar, problemas en términos de la decibilidad de algunos problemas básicos de decisión (por ejemplo, la equivalencia) [JSSY05], que sí son decidibles para las expresiones regulares (y, por tanto también para las ERVs, dado que éstas no pueden expresar lenguajes no regulares).

PROPIEDADES COMPUTACIONALES DE LAS ERVs

Aunque las ERVs constituyen un modelo simple de especificación de propiedades importantes en distintas áreas de la computación, hasta el momento de nuestro trabajo nadie había iniciado un estudio sistemático de sus propiedades computacionales. A continuación detallamos algunas de las más importantes conclusiones obtenidas a lo largo de nuestro estudio [ABR11a]:

1) Hemos mencionado en la sección anterior que las ERVs sólo pueden definir, bajo ambas semánticas, lenguajes regulares. Esto se debe al hecho que los lenguajes regulares son cerrados bajo unión e intersección, y la semántica existencial y universal de una ERV corresponden a la unión e intersección, respectivamente, de todas las ERVs sin variables obtenidas desde E al reemplazar sus variables por letras en el alfabeto (i.e. por sus valuaciones). Sin embargo, el número de valuaciones de E es exponencial en el número de sus variables, y, por tanto, $L_{\cup}(E)$ y $L_{\cap}(E)$ pueden ser expresados, respectivamente, por expresiones regulares de tamaño exponencial, en el caso de la semántica existencial, y doble exponencial, en el caso de la semántica universal.

Esto sugiere fuertemente que las ERVs son al menos exponencialmente más sucintas que su versión sin variables. Es decir, que ERVs de tamaño polinomial pueden definir lenguajes que sólo pueden ser definidos por expresiones regulares de tamaño exponencial. Note que esto no se sigue directamente de lo explicado en el párrafo anterior, ya que es necesario demostrar que existen ERVs E de tamaño polinomial tal que $L_{\cup}(E)$ (o $L_{\cap}(E)$) *requieren* para ser expresados de expresiones regulares de tamaño exponencial. Éste es uno de los principales resultados de nuestro artículo:

> ***Teorema:*** *existe una familia* $\{En\}_{n>0}$ *de ERVs de tamaño polinomial en n, tal que cualquier:*
>
> *a) expresión regular, o*
>
> *b) autómata (determinista o no determinista),*
>
> *que define* $L_{\cup}(E_n)$ *(respectivamente,* $L_{\cap}(E_n)$*) es de tamaño al menos exponencial (respectivamente, doble exponencial) con respecto a* E_n.

Esto significa que efectivamente las ERVs son al menos exponencialmente más sucintas que su versión sin variables, y que este resultado es robusto, en el sentido que aplica a todos los modelos de computación equivalentes a las expresiones regulares que hemos descrito en el presente artículo.

2) Existe otra interesante forma de demostrar que las ERVs permiten describir sucintamente lenguajes regulares complejos. Un problema clásico en lenguajes regulares es el de intersección, definido de la siguiente forma:

dadas expresiones regulares E_1 , ... , E_k de tamaño O(n), determine el tamaño de la menor expresión E que es equivalente a la intersección de todos los E_i´s (tal E existe ya que las expresiones regulares son cerradas bajo intersección). Es bien sabido que en algunos casos el menor E que cumple la propiedad expresada arriba es de tamaño $O(n^k)$ (es decir, exponencial). Sin embargo, utilizando ERVs y la semántica universal podemos expresar la intersección polinomialmente:

Teorema: *Sean E_1 , ... , E_k expresiones regulares de tamaño O(n). Existe ERV E de tamaño O(nk) tal que $L_\cap(E)$ es equivalente a la intersección de todos los E_i´s.*

Por supuesto, todo este poder de concisión de las ERVs tiene costos al momento de analizar la complejidad de ciertas tareas básicas de las expresiones regulares, lo que veremos en el próximo punto.

3) El problema básico de decisión para una expresión regular es el de verificar si una palabra pertenece al lenguaje definido por ella. Esto es, dada expresión regular φ y palabra w, ¿es cierto que $w \in L(\varphi)$? Es fácil ver que este problema puede ser resuelto eficientemente. Una demostración usual procede como sigue: convierta φ en un autómata no determinista equivalente. Se sabe que esto se puede hacer en tiempo polinomial. Luego verifique si la palabra w es aceptada por el autómata, lo que también puede realizarse polinomialmente.

Para el caso de las ERVs esto no es tan fácil. Imagine que dada ERV E queremos verificar si una palabra w pertenece a $L_\cup(E)$ o $L_\cap(E)$. La técnica de primero convertir a $L_\cup(E)$ o $L_\cap(E)$ en un autómata equivalente T, para luego verificar si w es aceptado por T, nos acarrearía costos computacionales insalvables, ya que sabemos que T puede ser de tamaño exponencial en E. Es decir este procedimiento nos entrega un algoritmo al menos exponencial, y, por tanto, no implementable computacionalmente.

Es posible demostrar que la complejidad del problema es un poco mejor, aunque no mucho más, utilizando el concepto de valuaciones. Note que para verificar que w pertenece a a $L_\cup(E)$ basta "adivinar" una valuación η para E y verificar, en tiempo polinomial, que w está en a $L(\eta(E))$. Dado que η es un testigo de tamaño polinomial en E, podemos concluir que el problema puede ser resuelto en *nondeterministic polynomial time* (NP). De la misma forma, verificar si w está en $L_\cap(E)$ puede ser resuelto en el complemento, coNP, de la clase NP.

Lamentablemente, salvo en casos bastante restringidos, el problema de verificar si una palabra w está en $L_\cup(E)$ puede ser completo para la clase NP. Esto dice que, bajo suposiciones ampliamente diseminadas en la comunidad, el problema es computacionalmente intratable. En particular, la complejidad del problema coincide con la complejidad del problema de satisfacibilidad o del vendedor viajero, ambos considerados inherentemente exponenciales.

CONCLUSIONES

Hemos definido la clase de las ERVs, analizado su aplicabilidad en dos áreas distintas de la computación y estudiado algunas de sus propiedades computacionales básicas. Dado que la mayoría de los problemas de decisión clásicos de autómatas y expresiones regulares se vuelven intratables en la presencia de variables, es importante en el futuro desarrollar heurísticas que permitan trabajar con ellas en ciertos contextos importantes (e.g. Bases de Datos de Grafos).

Otro problema importante es el de la clausura de las ERVs con respecto a las combinaciones Booleanas. Por ejemplo, sabemos que si E y E′ son ERVs entonces $L_\cup(E) \cap L_\cup(E')$ sólo puede ser representado, en algunos casos, por una expresión regular sin variables de tamaño al menos exponencial en E y E´. Sin embargo, es aún posible que el mismo lenguaje pueda ser representado por una ERV de tamaño polinomial. Este tipo de resultados podrían ser útiles en contextos dinámicos en que las ERVs son permanentemente actualizadas, modificadas y consultadas, como es el caso de las Bases de Datos de Grafos con información incompleta.

AGRADECIMIENTOS

Este trabajo fue realizado en colaboración con Leonid Libkin y Juan Reutter, ambos de la Universidad de Edinburgo. Mi trabajo fue patrocinado por el proyecto Fondecyt 1110171. BITS

BIBLIOGRAFÍA

[1] [ALSU06] A. Aho, M. Lam, R. Sethi, J. Ullman. Compilers: Principles, Techniques and Design. Addison-Wesley, 2nd edition, 2006.

[2] [BLR11] P. Barceló, L. Libkin, J. Reutter. Querying Graph Patterns. PODS 2011.

[3] [BLR11a] P. Barceló, L. Libkin, J. Reutter. Parameterized Regular Expressions and Their Languages. Enviado.

[4] [Bun97] P. Buneman. Semistructured Data. PODS 1997.

[5] [BSV99] P. Buneman, D. Suciu, V. Vianu. Data on the Web. Morgan Kauffman, 1999.

[6] [CR03] M. Crochemore, W. Rytter. Jewels of Stringology. WSP, 2003.

[7] [JSSY95] T. Jiang, A. Salomaa, K. Salomaa, S. Yu. Decision problems for patterns. JCSS, 50(1), 53-63, 1995.

[8] [LRYS04] Y. Liu, T. Rothamel, F. Yu, S. Stoller, N. Hu. Parametric Regular Path Queries. PLDI 2004.

[9] [Mar05] C. Martín-Vide. Formal Grammars and Languages. The Oxford Handbook of Computational Linguistics, 2005.

[10] [NR07] G. Navarro, M. Raffinot. Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences. Cambridge University Press, 2002.

[11] [NNH10] F. Nielson, H. Nielson, C. Hankin. Principles of Program Analysis. Springer-Verlag, 2010.

[12] [Sal03] K. Salomaa. Patterns. Bulletin of the EATCS, 2003.

[13] [VW86] M. Vardi, P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. LICS 1986.