Enumeración de anotaciones con delay constante

Clase 27

IIC 2223

Prof. Cristian Riveros

Autómata con anotaciones (recordatorio)

Definición

Un autómata con anotaciones (AnnA) es una tupla:

$$\mathcal{N} \quad = \quad (Q, \Sigma, \Lambda, \Delta, I, F)$$

- Q es un conjunto finito de estados.
- Σ es el alfabeto de input.
- Λ es un conjunto finito de etiquetas (Labels).
- $\Delta \subseteq Q \times \left(\Sigma \cup \Sigma \times \mathcal{N}\right) \times Q$ es la relación de transición.
- $I \subseteq Q$ es un conjunto de estados iniciales.
- $F \subseteq Q$ es el conjunto de estados finales (o aceptación).

Las transiciones (p, a, ℓ, q) simbolizan que
al leer la letra a, está letra **será anotada** con ℓ

Ejecución de un AnnA (recordatorio)

Sea un AnnA $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$ y un documento $d = a_0 \ldots a_{n-1} \in \Sigma^*$.

Definiciones

Una **ejecución** ρ de $\mathcal{A}$ sobre d es una secuencia $\rho : p_0 \xrightarrow{t_0} p_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} p_n$ tal que cumple todas las siguientes condiciones:

- $p_0 \in I$
- para todo $i < n$, t_i es de la forma $t_i = a_i$ o $t_i = (a_i, \ell)$ para algún $\ell \in \Lambda$
- para todo $i < n$, $(p_i, t_i, p_{i+1}) \in \Delta$.

Se define la **anotación de ρ** como $\text{ann}(\rho) = \text{ann}_0(t_0) \cdot \ldots \cdot \text{ann}_{n-1}(t_{n-1})$:

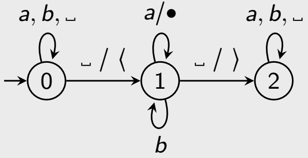
$$\text{ann}_i(t) = \begin{cases} (i, \ell) & t = (a, \ell) \\ \epsilon & t = a \end{cases}$$

Decimos que ρ es de **aceptación** ssi $q_n \in F$.

Ejecución de un AnnA (recordatorio)

Ejemplos de ejecuciones

$$d = \underbrace{\texttt{a}\ \ \texttt{abba}\ \ \texttt{ba}\ \ \texttt{b}}_{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10}$$



Algunas ejecuciones sobre d:

$$\rho_1 :\ p_0 \xrightarrow{a} p_0 \xrightarrow{\sqcup/\langle} p_1 \xrightarrow{a/\bullet} p_1 \xrightarrow{b} p_1 \xrightarrow{b} p_1 \xrightarrow{a/\bullet} p_1 \xrightarrow{\sqcup/\rangle} p_2 \xrightarrow{b} p_2 \xrightarrow{a} p_2 \xrightarrow{\sqcup} p_2 \xrightarrow{b} p_2$$

$$\rho_2 :\ p_0 \xrightarrow{a} p_0 \xrightarrow{\sqcup} p_0 \xrightarrow{a} p_0 \xrightarrow{b} p_0 \xrightarrow{b} p_0 \xrightarrow{a} p_0 \xrightarrow{\sqcup/\langle} p_1 \xrightarrow{b} p_1 \xrightarrow{a/\bullet} p_1 \xrightarrow{\sqcup/\rangle} p_2 \xrightarrow{b} p_2$$

$$\text{ann}(\rho_1) \ = \ (1, \langle)\,(2, \bullet)\,(5, \bullet)\,(6, \rangle) \qquad \text{ann}(\rho_2) \ = \ (6, \langle)\,(8, \bullet)\,(9, \rangle)$$

$$\texttt{a}\overset{\langle}{\sqcup}\overset{\bullet}{\texttt{a}}\texttt{bb}\overset{\bullet}{\texttt{a}}\overset{\rangle}{\sqcup}\texttt{ba}\sqcup\texttt{b} \qquad\qquad \texttt{a}\sqcup\texttt{abba}\overset{\langle}{\sqcup}\texttt{b}\overset{\bullet}{\texttt{a}}\overset{\rangle}{\sqcup}\texttt{b}$$

Output de un AnnA

Sea $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$ un autómata con anotaciones (AnnA).

Definición

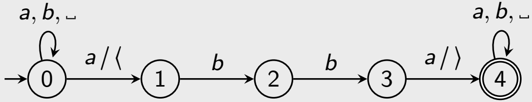
Se define la función $[\![\mathcal{N}]\!]$ tal que para todo documento $d \in \Sigma^*$:

$$[\![\mathcal{N}]\!](d) \;=\; \left\{ \text{ann}(\rho) \;\middle|\; \rho \text{ es una ejecución } \textbf{aceptación} \text{ de } \mathcal{N} \text{ sobre } d \right\}$$

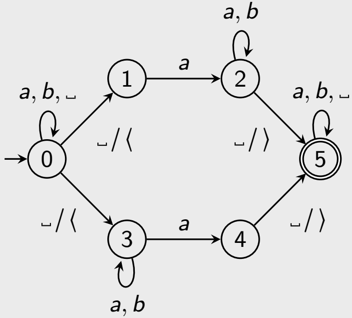
Determinismo para autómata con anotaciones

¿cuál de los dos AnnA diría que son deterministas?

1.



2.



Determinismo para autómata con anotaciones

Sea $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$ un autómata con anotaciones.

Definición

Decimos que $\mathcal{N}$ es **Input-Output determinista** (I/O-determinista) ssi $|I| = 1$ y para todo $(p, t_1, q_1), (p, t_2, q_2) \in \Delta$, si $t_1 = t_2$, entonces $q_1 = q_2$.

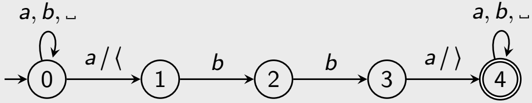
$\mathcal{N}$ es determinista al recibir
el documento y una anotación **simultáneamente**

¿qué ventaja tiene un autómata I/O-determinista?

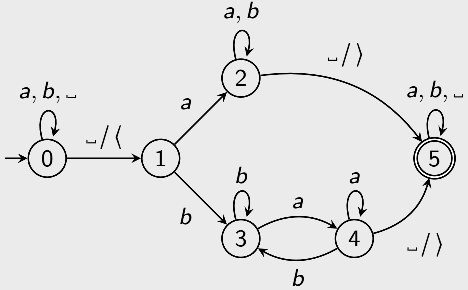
Determinismo para autómata con anotaciones

Ejemplos

1.



2'.



Todo AnnA se puede I/O-determinizar

Teorema

Para todo AnnA $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$,
existe un AnnA I/O-determinista $\mathcal{N}^{\text{det}}$ tal que $[\![\mathcal{N}]\!] = [\![\mathcal{N}^{\text{det}}]\!]$.

Demostración

Considere la determinización de $\mathcal{N}$ como:

$$\mathcal{N}^{\text{det}} = (2^Q, \Sigma, \Lambda, \Delta^{\text{det}}, q_0^{\text{det}}, F^{\text{det}})$$

- $2^Q = \{S \mid S \subseteq Q\}$ es el conjunto potencia de Q.
- $q_0^{\text{det}} = I$.
- $\Delta^{\text{det}} : 2^Q \times (\Sigma \cup \Sigma \times \Lambda) \to 2^Q$ tal que para todo $t \in \Sigma \cup (\Sigma \times \Lambda)$:

$$\Delta^{\text{det}}(S, t) = \{q \in Q \mid \exists p \in S.\ (p, t, q) \in \Delta\}$$

- $F^{\text{det}} = \{S \subseteq Q \mid S \cap F \neq \varnothing\}$.

$\blacksquare$

¿cómo evaluamos una regex?

`PROBLEMA:` Evaluación de regex

`INPUT:` una regex R y
un documento d

`OUTPUT:` Enumerar todos los mappings en $[\![R]\!](d)$

1. Transformamos R a un vset autómata funcional $\mathcal{A}_R$.
2. Transformamos $\mathcal{A}_R$ a un AnnA $\mathcal{N}_R$.
3. Determinizamos $\mathcal{N}_R$ a un AnnA I/O determinista $\mathcal{N}_R^{\text{det}}$.
4. Calculamos los resultados $[\![\mathcal{N}_R^{\text{det}}]\!](d \cdot \#)$.

¿cómo computamos todas las anotaciones de $[\![\mathcal{N}_R^{\text{det}}]\!](d \cdot \#)$?

Outline

Enumeración con delay constante

Enumerable compact sets

Algoritmo de evaluación para AnnA

Algunas conclusiones

Outline

Enumeración con delay constante

Enumerable compact sets

Algoritmo de evaluación para AnnA

Algunas conclusiones

Problemas de enumeración

Definición

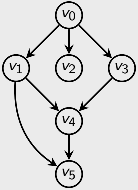
- Sea $\mathcal{C}$ una clase de instancias (inputs) y Ω un alfabeto (quizás infinito).
- Sea $[\![\cdot]\!] : \mathcal{C} \to 2^{\Omega^*}$ una función tal que para cada I, $[\![I]\!] = \{out_1, out_2, \ldots, out_k\}$ es un conjunto finito de outputs $out_i \in \Omega^*$.

Ejemplo

$\mathcal{C}$: la clase de todos los grafos $G = (V, E)$ tal que G es acíclico (no tiene ciclos) y tiene un solo nodo raíz v_0.

Ω: conjunto de vértices.

$[\![G]\!]$: todos los caminos (secuencias) máximales desde v_0.



$$[\![G]\!] = \{\, v_0 v_1 v_5,\, v_0 v_1 v_4 v_5,\, v_0 v_2,\, v_0 v_3 v_4 v_5 \,\}$$

Problemas de enumeración

Definición problema de enumeración

`INPUT:` Instancia $I \in \mathcal{C}$.

`OUTPUT:` Enumerar $[\![I]\!] = \{out_1, out_2, \ldots, out_k\}$.

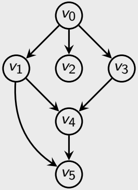
Con **enumerar** nos referimos a computar e imprimir en memoria:

$$\#out_1\#out_2\#\ldots\#out_k\#$$

para algún orden de $[\![I]\!]$ y donde $\#$ es un símbolo separador especial.

Ejemplo

`INPUT:`



`OUTPUT:` $\# v_0 v_1 v_5 \# v_0 v_1 v_4 v_5 \# v_0 v_2 \# v_0 v_3 v_4 v_5 \#$

Problemas de enumeración

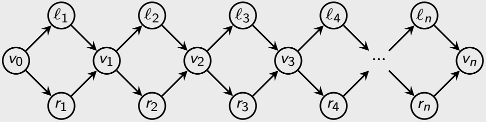
Definición problema de enumeración

`INPUT:` Instancia $I \in \mathcal{C}$.

`OUTPUT:` Enumerar $[\![I]\!] = \{out_1, out_2, \ldots, out_k\}$.

Ejemplo problematico

`INPUT:` G :



- ¿cuántos nodos tiene el grafo? $n + 2n + 1$ nodos.
- ¿cuántos caminos máximales tiene el grafo? 2^n nodos.

¿cómo encontrar un algoritmo **eficiente** si el output puede ser exponencial?

Algoritmos de enumeración con delay constante

Definición problema de enumeración

`INPUT:` Instancia $I \in \mathcal{C}$.

`OUTPUT:` Enumerar $[\![I]\!] = \{out_1, out_2, \ldots, out_k\}$.

Un algoritmo de enumeración para el problema anterior tiene **delay constante** si funciona en dos fases:

1. **Preprocesamiento**: el algoritmo recibe I y realiza algún cómputo.
2. **Enumeración**: terminada 1., el algoritmo imprime secuencialmente:

$$\underset{\substack{\uparrow \\ 0}}{\#}\ out_1\ \underset{\substack{\uparrow \\ 1}}{\#}\ out_2\ \underset{\substack{\uparrow \\ 2}}{\#}\ \ldots\ \underset{\substack{\uparrow \\ k-1}}{\#}\ out_k\ \underset{\substack{\uparrow \\ k}}{\#}$$

donde existe $C \in \mathbb{N}$ tal que para todo $i \leq k$ el tiempo entre imprimir el $(i-1)$-ésimo y el i-ésimo símbolo $\#$ es menor a $C \cdot |out_i|$.

El tiempo para entregar el siguiente output **solo depende de su tamaño**.

Algoritmos de enumeración con delay constante

1. **Preprocesamiento**: el algoritmo recibe I y realiza algún cómputo.
2. **Enumeración**: terminada 1., el algoritmo imprime secuencialmente:

$$\#out_1\#out_2\#\ldots\#out_k\#$$

donde existe $C \in \mathbb{N}$ tal que para todo $i \leq k$ el tiempo entre imprimir el $(i-1)$-ésimo y el i-ésimo símbolo $\#$ es menor a $C \cdot |out_i|$.

Function `enum-maxpaths` $(G = (V, E))$

  Preprocesar G y encontrar v_0

  `rec-maxpaths` (G, v_0, v_0)

  print $\#$

Function `rec-maxpaths` (G, v, s)

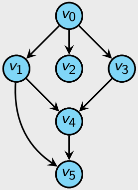
  if $\{u \mid (v,u) \in E\} = \varnothing$ **then**

    print $\# \cdot s$

  else

    foreach $u \in \{u \mid (v,u) \in E\}$ **do**

      `rec-maxpaths` $(G, u, s \cdot u)$



$$\#v_0 v_1 v_5\#v_0 v_1 v_4 v_5\#v_0 v_2\#v_0 v_3 v_4 v_5\#$$

Algoritmos de enumeración con delay constante

Un algoritmo de enumeración para el problema anterior
tiene **delay constante** si funciona en dos fases:

1. **Preprocesamiento**: el algoritmo recibe I y realiza algún cómputo.
2. **Enumeración**: terminada 1., el algoritmo imprime secuencialmente:

   $$\#out_1\#out_2\#\ldots\#out_k\#$$

   donde existe $C \in \mathbb{N}$ tal que para todo $i \leq k$ el tiempo entre imprimir el $(i-1)$-ésimo y el i-ésimo símbolo $\#$ es menor a $C \cdot |out_i|$.

Definición

Si el algoritmo de enumeración con delay constante toma tiempo $\mathcal{O}(f(|I|))$ diremos que nuestro algoritmo **toma tiempo $\mathcal{O}(f)$**.

Ejemplo

Para el algoritmo de enumeración para caminos máximos,
el preprocesamiento (ej. encontrar v_0) tomará tiempo $\mathcal{O}(|G|)$.

Por lo tanto, el algoritmo de enumeración toma **tiempo lineal** en G.

¿cómo evaluamos una regex?

1. Transformamos R a un vset autómata funcional $\mathcal{A}_R$.
2. Transformamos $\mathcal{A}_R$ a un AnnA $\mathcal{N}_R$.
3. Determinizamos $\mathcal{N}_R$ a un AnnA I/O determinista $\mathcal{N}_R^{\text{det}}$.
4. Calculamos los resultados $[\![\mathcal{N}_R^{\text{det}}]\!](d \cdot \#)$.

`PROBLEMA:`	Evaluación de AnnA I/O-determinista
`INPUT:`	un AnnA I/O-determinista $\mathcal{N}$ y un documento d
`OUTPUT:`	Enumerar todas las anotaciones en $[\![\mathcal{N}]\!](d)$.

Queremos una algoritmo de enumeración
con **delay constante** y **tiempo** $\mathcal{O}(|\mathcal{N}| \cdot |d|)$.

Outline

Enumeración con delay constante

Enumerable compact sets

Algoritmo de evaluación para AnnA

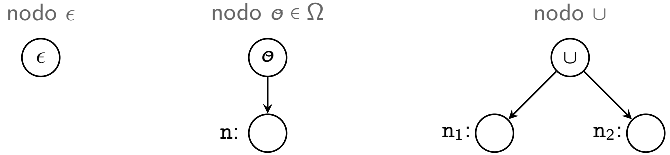
Algunas conclusiones

Enumerable Compact Set

Sea Ω un alfabeto (quizás infinito).

Definición

Un **enumerable compact set** (ECS) es una estructura de datos compuesta por nodos y conexiones, donde los nodos pueden ser de 3 tipos:

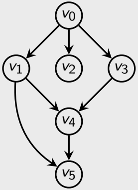


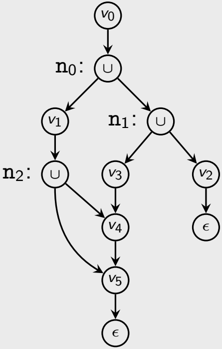
Cada nodo define un conjunto finito en Ω^*, recursivamente:

$$[\![\epsilon]\!] = \{\epsilon\} \qquad [\![\theta]\!] = [\![\mathtt{n}]\!] \cdot \{\theta\} \qquad [\![\cup]\!] = [\![\mathtt{n}_1]\!] \cup [\![\mathtt{n}_2]\!]$$

Enumerable Compact Set

Ejemplo de una ECS y sus resultados





$$
\begin{aligned}
[\![v_4]\!] &= \{v_5 v_4\} \\
[\![\mathtt{n}_2]\!] &= \{v_5,\, v_5 v_4\} \\
[\![v_1]\!] &= \{v_5 v_1,\, v_5 v_4 v_1\} \\
[\![\mathtt{n}_1]\!] &= \{v_5 v_4 v_3,\, v_2\} \\
[\![\mathtt{n}_0]\!] &= \{v_5 v_1,\, v_5 v_4 v_1,\\
&\qquad v_5 v_4 v_3,\, v_2\}
\end{aligned}
$$

Problema de enumeración para ECS

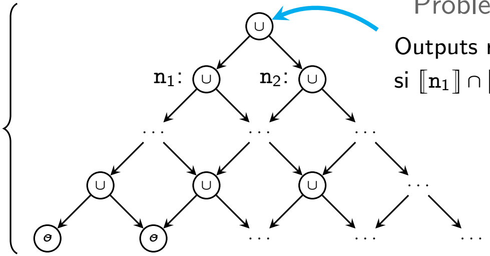
`INPUT:` un ECS $\mathcal{E}$ y
un nodo $\mathtt{n}$ de $\mathcal{E}$

`OUTPUT:` Enumerar todos los resultados $[\![\mathtt{n}]\!]$.

¿podemos **enumerar** todos los resultados con **delay constante**?

Problema 1.

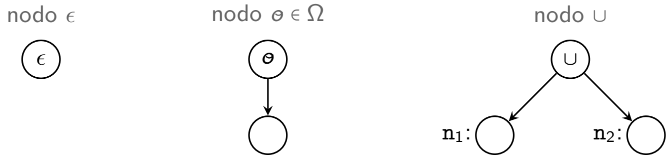
Delay dependerá de esta altura



Problema 2.

Outputs repetidos si $[\![\mathtt{n}_1]\!] \cap [\![\mathtt{n}_2]\!] \neq \emptyset$

Restricciones a Enumerable Compact Set



Definiciones

0. Se define el odepth de un nodo recursivamente como:

$$\text{odepth}(\epsilon) = 0 \qquad \text{odepth}(\theta) = 0 \qquad \text{odepth}(\cup) = \text{odepth}(\mathtt{n}_1) + 1$$

1. Un ECS $\mathcal{E}$ es ***k*-acotado** si $\text{odepth}(\mathtt{n}) \leq k$ para todo nodo $\mathtt{n}$ en $\mathcal{E}$.

2. Un ECS $\mathcal{E}$ es **sin repeticiones** si para todo nodo $\cup$ (como arriba) en $\mathcal{E}$ se cumple que $[\![\mathtt{n}_1]\!] \cap [\![\mathtt{n}_2]\!] = \varnothing$.

Enumeración con delay constante para ECS

Definiciones

0. Se define el odepth de un nodo recursivamente como:

$$\text{odepth}(\epsilon) = 0 \qquad \text{odepth}(\sigma) = 0 \qquad \text{odepth}(\cup) = \text{odepth}(\mathtt{n}_1) + 1$$

1. Un ECS $\mathcal{E}$ es ***k*-acotado** si $\text{odepth}(\mathtt{n}) \leq k$ para todo nodo $\mathtt{n}$ en $\mathcal{E}$.

2. Un ECS $\mathcal{E}$ es **sin repeticiones** si para todo nodo $\cup$ (como arriba) en $\mathcal{E}$ se cumple que $[\![\mathtt{n}_1]\!] \cap [\![\mathtt{n}_2]\!] = \varnothing$.

Teorema

Para $k \geq 1$ fijo, existe un algoritmo que, para todo ECS ***k*-acotado** y **sin repeticiones** $\mathcal{E}$ y nodo $\mathtt{n}$, enumera $[\![\mathtt{n}]\!]$ con delay constante y tiempo $\mathcal{O}(1)$.

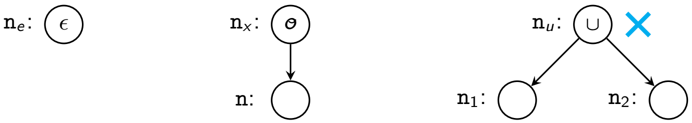
Demostración: ejercicio.

Operaciones para un ECS

Deseamos soportar las siguientes **interfaz** sobre un ECS $\mathcal{E}$:

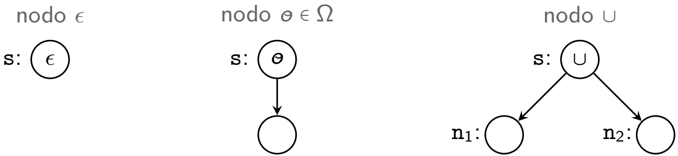
$\mathtt{n}_e := \mathcal{E}.\mathtt{epsilon}$	$\mathtt{n}_x := \mathcal{E}.\mathtt{extend}(\mathtt{n}, \mathit{o})$	$\mathtt{n}_u := \mathcal{E}.\mathtt{union}(\mathtt{n}_1, \mathtt{n}_2)$
$[\![\mathtt{n}_e]\!] = \{\epsilon\}$	$[\![\mathtt{n}_x]\!] = [\![\mathtt{n}]\!] \cdot \{\mathit{o}\}$	$[\![\mathtt{n}_u]\!] = [\![\mathtt{n}_1]\!] \cup [\![\mathtt{n}_1]\!]$
	t.q. $\mathit{o} \in \Omega$	t.q. $[\![\mathtt{n}_1]\!] \cap [\![\mathtt{n}_2]\!] = \varnothing$

¿cómo implementamos estás operaciones?



¿cómo soportamos estas operaciones manteniendo k-**acotado**?

Nodos seguros

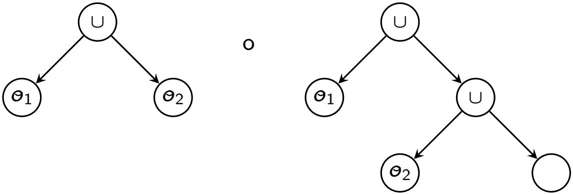


Definición

Decimos que un nodo s es **seguro** (safe) si cumple que:

- s es nodo ϵ o nodo θ, o
- s es nodo $\cup$, $\mathrm{odepth}(\mathtt{s}) \leq 1$ y $\mathrm{odepth}(\mathtt{n}_2) \leq 1$.

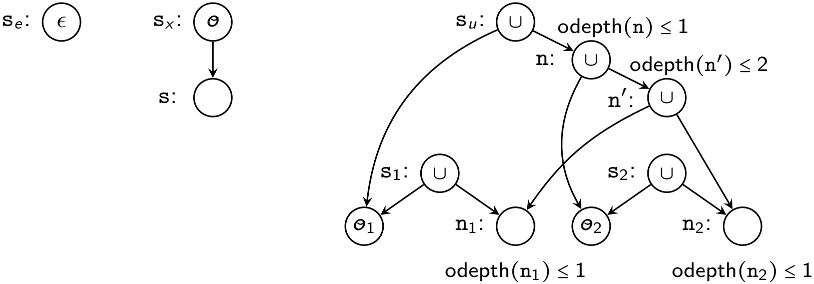
En otras palabras, si s es nodo-$\cup$ y seguro, entonces s es de dos formas:



Operaciones para un ECS sobre nodos seguros

Para nodos **seguros** $\mathtt{s}$, $\mathtt{s}_1$, $\mathtt{s}_2$ soportamos las operaciones sobre un ECS $\mathcal{E}$:

$\mathtt{s}_e := \mathcal{E}.\mathtt{epsilon}$	$\mathtt{s}_x := \mathcal{E}.\mathtt{extend}(\mathtt{s}, \mho)$	$\mathtt{s}_u := \mathcal{E}.\mathtt{union}(\mathtt{s}_1, \mathtt{s}_2)$
$[\![\mathtt{s}_e]\!] = \{\epsilon\}$	$[\![\mathtt{s}_x]\!] = [\![\mathtt{s}]\!] \cdot \{\mho\}$	$[\![\mathtt{s}_u]\!] = [\![\mathtt{s}_1]\!] \cup [\![\mathtt{s}_1]\!]$
	t.q. $\mho \in \Omega$	t.q. $[\![\mathtt{s}_1]\!] \cap [\![\mathtt{s}_2]\!] = \varnothing$



Si $\mathcal{E}$ es 2-**acotado**, entonces el resultado también es 2-**acotado**.

Resumen sobre ECS

Teorema

Para un ECS 2-**acotado** y **sin repetidos** $\mathcal{E}$, y un nodo cualquiera $\mathtt{n}$ de $\mathcal{E}$:

1. $[\![\mathtt{n}]\!]$ se puede enumerar con **delay constante** y tiempo $\mathcal{O}(1)$.
2. Las operaciones `epsilon`, `extend` y `union` **sobre nodos seguros** mantienen la propiedad de 2-**acotado** y **retorna nodos seguros**.
3. Los nodos input de las operaciones **mantienen sus resultados**.
4. Cada operación toma **tiempo constante** $\mathcal{O}(1)$.

Usaremos nuestra estructura ECS para la evaluación de AnnA

Outline

Enumeración con delay constante

Enumerable compact sets

Algoritmo de evaluación para AnnA

Algunas conclusiones

Algoritmo de evaluación con delay constante

Sea $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, q_0, F)$ un **AnnA I/O-determinista**, $d = a_0 \dots a_{n-1}$ un **documento**, $\mathcal{E}$ un **ECS**, y S un **arreglo de nodos** indexados por Q.

Function `eval-delayconstante` $(\mathcal{A}, d)$

$S \leftarrow [\texttt{null}]$

$S[q_0] \leftarrow \mathcal{E}.\texttt{epsilon}$

for $i = 0$ **to** $n-1$ **do**

$S_{\text{old}} \leftarrow S$

$S \leftarrow [\texttt{null}]$

foreach $(p, a_i, q) \in \Delta$ **such that** $S_{old}[p] \neq \texttt{null}$ **do**

if $S[q] = \texttt{null}$ **then** $S[q] \leftarrow S_{\text{old}}[p]$

else $S[q] \leftarrow \mathcal{E}.\texttt{union}\big(S[q], S_{\text{old}}[p]\big)$

foreach $(p, a_i, \ell, q) \in \Delta$ **such that** $S_{old}[p] \neq \texttt{null}$ **do**

$\texttt{n} \leftarrow \mathcal{E}.\texttt{extend}\big(S_{\text{old}}[p], (i, \ell)\big)$

if $S[q] = \texttt{null}$ **then** $S[q] \leftarrow \texttt{n}$

else $S[q] \leftarrow \mathcal{E}.\texttt{union}\big(S[q], \texttt{n}\big)$

foreach $q \in \{p \in Q \mid S[p] \neq \texttt{null}\} \cap F$ **do**

enumerate$(\mathcal{E}, S[q])$

Análisis del algoritmo

- $\mathcal{N}$ es I/O-determinista y, por lo tanto, $\mathcal{E}$ es **sin duplicados**.
- Cada operación en $\mathcal{E}$ toma tiempo constante.
- Los resultados en $\mathcal{E}$ se pueden enumerar con delay constante.

...entonces, el algoritmo enumera $[\![\mathcal{N}]\!](d)$
con delay constante y en tiempo $\mathcal{O}(|\mathcal{N}| \cdot |d|)$.

Outline

Enumeración con delay constante

Enumerable compact sets

Algoritmo de evaluación para AnnA

Algunas conclusiones

Sobre los resultados de extracción de información

Son un **resumen/ideas** de los siguientes artículos:

Maturana, Riveros, Vrgoc
"*Document Spanners for Extracting Incomplete Information.*" PODS 2018.

Florenzano, Riveros, Ugarte, Vansummeren, Vrgoc
"*Constant Delay Algorithms for Regular Document Spanners.*" PODS 2018.

Grez, Riveros, Ugarte
"*A Formal Framework for Complex Event Processing.*" ICDT 2019.

Muñoz, Riveros
"*Streaming Enumeration on Nested Documents.*" ICDT 2022.

Amarilli, Jachiet, Muñoz, Riveros
"*Efficient Enumeration for Annotated Grammars.*" PODS 2022.

Bucchi, Grez, Quintana, Riveros, Vansummeren
"*CORE: a COmplex event Recognition Engine.*" VLDB 2022.

+ artículos de otros grupos + investigación en curso.

!RE{MATCH} ABOUT

Query

```
(^|\n)!x{[A-Z][a-z]{4,}} !y{([A-Z][a-z ]+)+}($|\n)
```

RUN

Text

```
Nicolas Van Sint Jan
Vicente Calisto
Marjorie Bascunan
Oscar Carcamo
Cristian Riveros
Domagoj Vrgoc
```

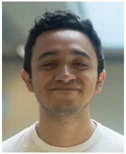
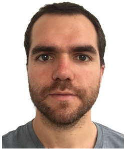
IMPORT FILE

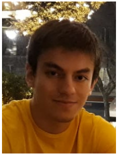
Matches

id	x	y
0	Domagoj	Vrgoc
1	Cristian	Riveros
2	Oscar	Carcamo
3	Marjorie	Bascunan
4	Vicente	Calisto
5	Nicolas	Van␣Sint␣Jan

‹ 1 ›

Desarrollo de investigación / REmatch

(Faltan varios)

Sobre mis temas de investigación

Manejo de datos:

- Big data.
- Datos streaming.
- Extracción de información.

Grafos de datos:

- Web semántica.
- Base de datos de grafos.
- Centralidad de datos.

Lógica / Lenguajes formales:

- Teoría de modelos finitos.
- Teoría de automatas.

Teoría de la Computación:

- Complejidad computacional.
- Algoritmos aleatorios.

Proyectos de implementación

1. REmatch **Motor de extracción de información**
2. CORE **Base de datos streaming**
3. MilleniumDB **Base de datos de grafos**

Estan **invitados a colaborar** en cualquiera de estos proyectos

(solo escribanme y pregunten)

FIN :(

Gracias!