ELiC11: Transductores

Clase 3: Transductores (2nda Parte)
y Pesos

Carlos Areces
carlos.areces@gmail.com

La clase pasada

- Autómatas Finitos
  - Operaciones sobre Autómatas
- Relaciones Regulares
  - Transductores finitos
  - Operaciones sobre TF
    - Granularidad
    - Reversion
    - Inversión
    - Pushing

La clase de hoy

- Repaso
- Operaciones sobre Transductores
  - Determinización
  - Eliminación de épsilon
  - Composición
  - Ejemplos
- Pesos
  - Autómatas
  - Transductores

Repaso

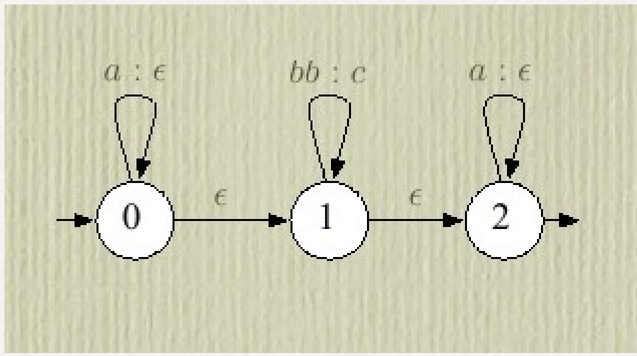
Relaciones Regulares

- Una **relación de cadenas** es un conjunto de pares de cadenas.

input : output

- La clase de las relaciones regulares es la clase de relaciones de cadenas más pequeña que incluye
  - La relación vacía: {}.
  - Las relaciones unitarias: $\{\epsilon:\epsilon\}$, $\{\epsilon:a\}$, $\{a:\epsilon\}$, $\{a:b\}$, ...
  - Cerradas bajo
    - Unión
    - Concatenación $\{(xx',yy') \mid (x,y) \in R, (x',y') \in R'\}$
    - Iteración

Transductores Finitos



Función de transición

$q_0\, a \vdash q_0$	$q_0 \vdash q_1$
$q_1\, bb \vdash c\, q_1$	$q_1 \vdash q_2$
$q_2\, a \vdash q_2$	$q_2 \sharp \vdash \sharp$

Una derivación:

$q_0 aaabba\# \vdash q_0 aabba\# \vdash q_0 abba\# \vdash$

$\vdash q_0 bba\# \vdash q_1 bba\# \vdash c q_1 a\#$

$\vdash c q_2 a\# \vdash c q_2\# \vdash c\#$

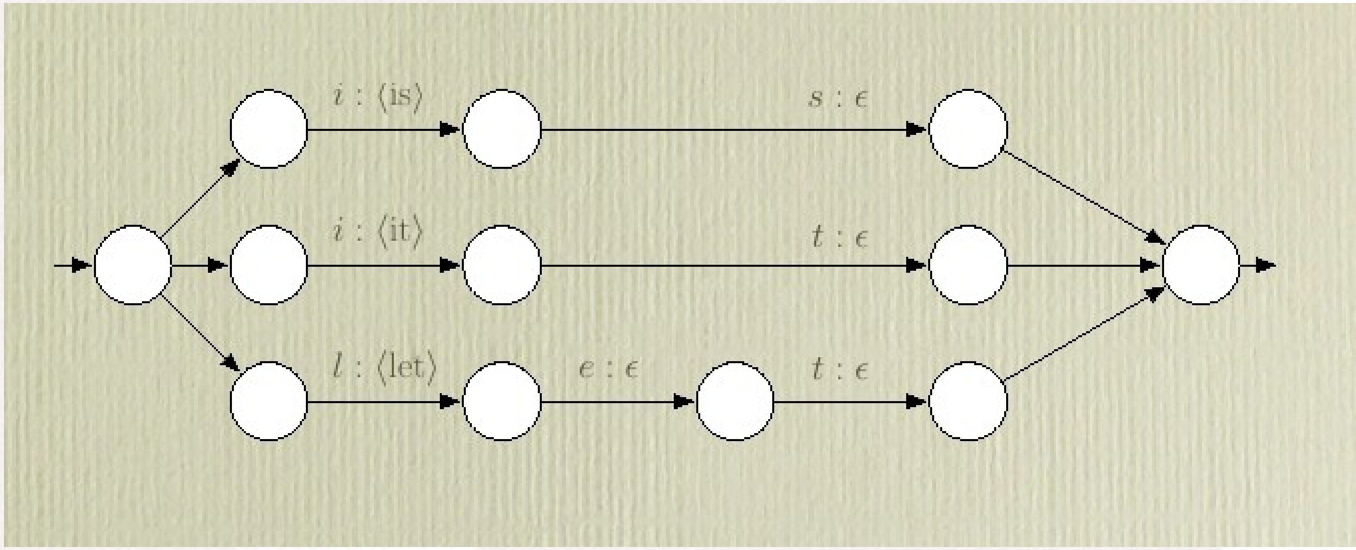
Def. de aceptación: aceptar s:t sii $q_0 s\# \vdash^* t\#$

Operaciones sobre transductores

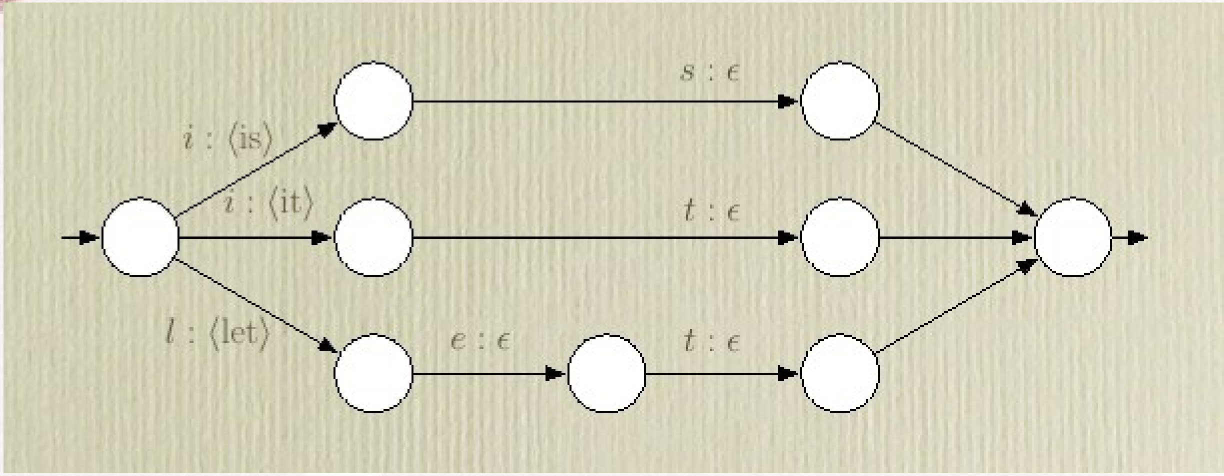
- Varias de las operaciones sobre autómatas también funcionan sobre transductores (más o menos). Y aparecen algunas nuevas:
  - Granularidad (de entrada y salida)
  - Reversión izquierda-derecha
  - ***Inversión***
  - ***Pushing***

Fin de repaso

Determinización

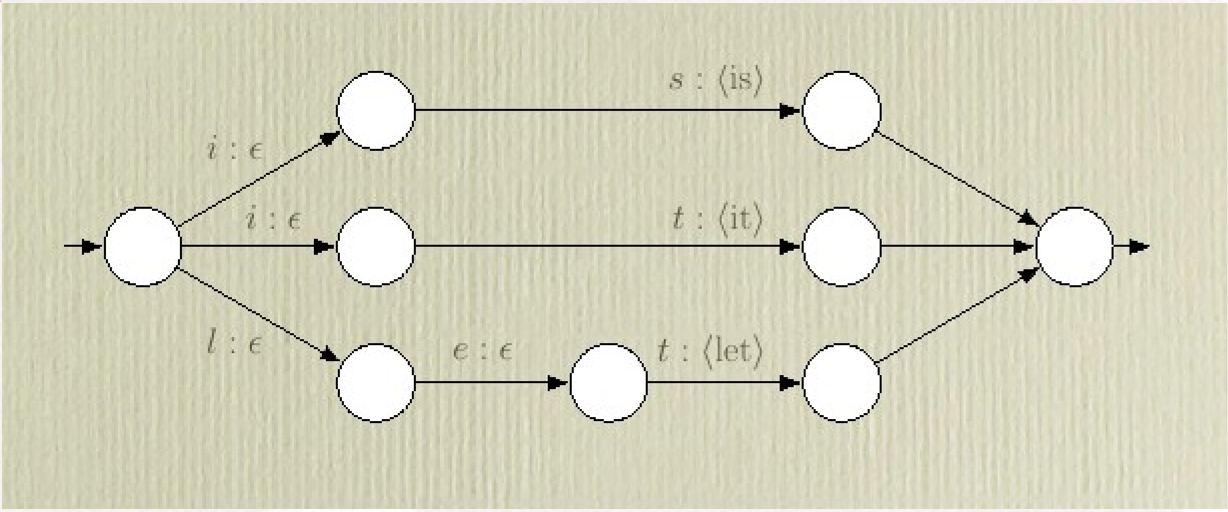


Determinización

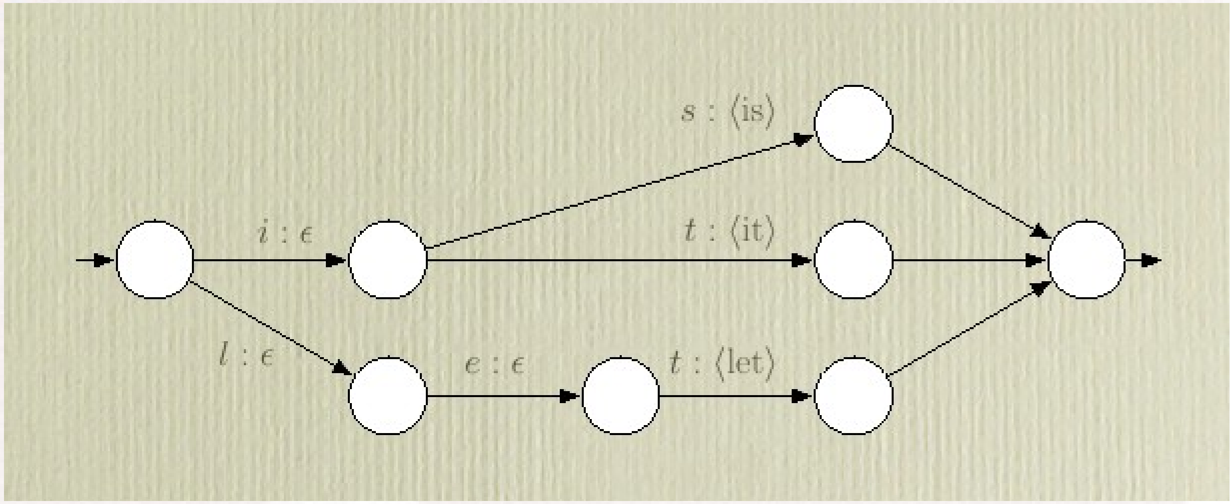


No se puede determinizar más sin pushing.

Determinización

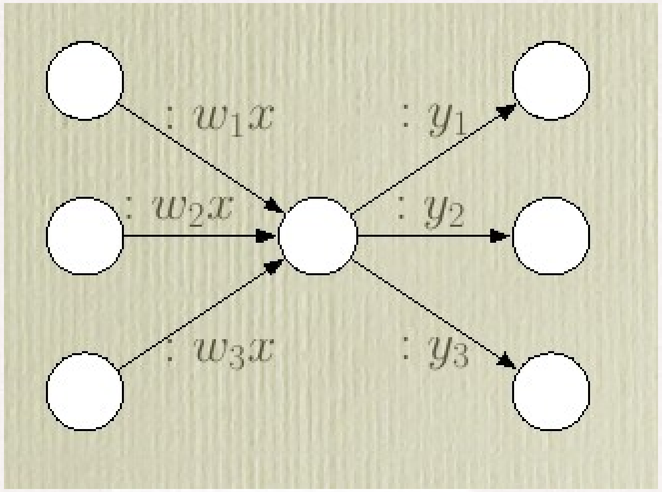


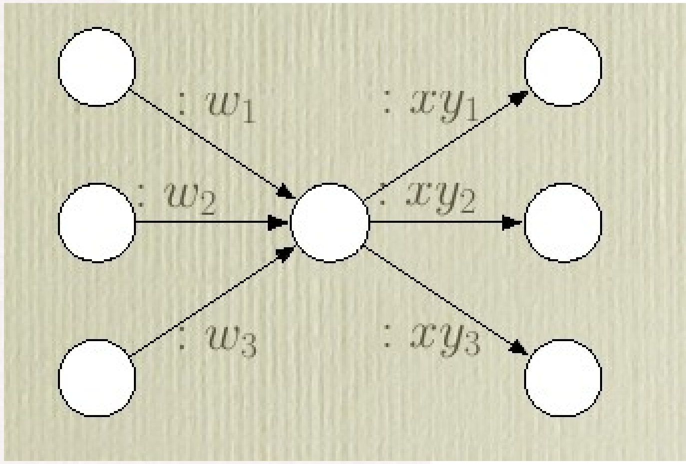
Determinización



Pushing

- Pushing puede hacerse no sólo sobre ϵ:

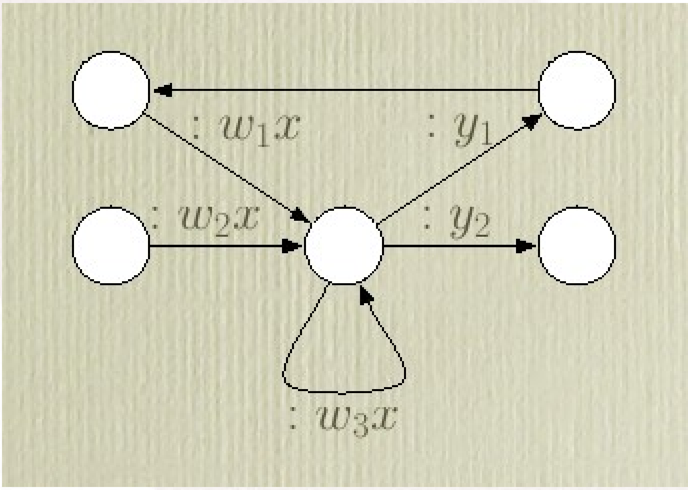




- Si todos los ejes de entrada terminan en x:
  - Remover x de los ejes de entrada
  - Agregar x a los ejes de salida

Pushing

- Pushing puede hacerse también sobre loops:



- Si todos los ejes de entrada terminan en x:
  - Remover x de los ejes de entrada
  - Agregar x a los ejes de salida

Limitaciones

- Los transductores determinísticos no reconocen todas las relaciones regulares. Lo mismo pasa con los transductores sin ϵ.

Limitaciones

- Los transductores determinísticos no reconocen todas las relaciones regulares. Lo mismo pasa con los transductores sin ϵ.
- Dar un transductor para los siguientes lenguajes:

  L1 = $\{w{:}v \mid w \in ab^*a,\ v \in aa\}$ (eliminación de b)

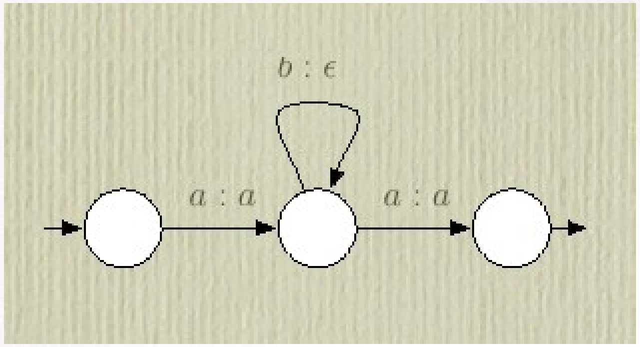
  L2 = $\{v{:}w \mid v \in aa,\ w \in ab^*a\}$ (inserción de b)

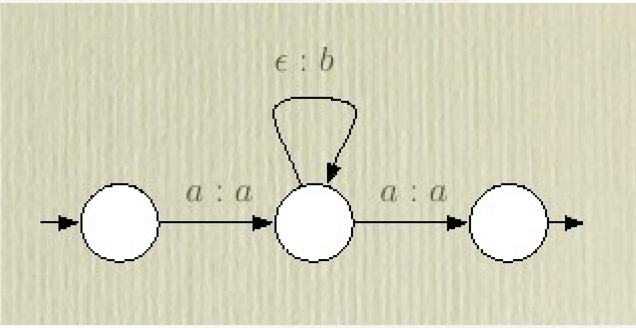
Limitaciones

- Los transductores determinísticos no reconocen todas las relaciones regulares. Lo mismo pasa con los transductores sin ϵ.
- Dar un transductor para los siguientes lenguajes:

  L1 = $\{w{:}v \mid w \in ab^*a,\ v \in aa\}$ (eliminación de b)

  L2 = $\{v{:}w \mid v \in aa,\ w \in ab^*a\}$ (inserción de b)





Composición

- Dados dos autómatas sin transiciones ϵ

  $T1 = \langle Q,\Sigma,\Sigma',\Delta,q_0\rangle$

  $T2 = \langle Q',\Sigma',\Sigma'',\Delta',q'_0\rangle$

  la composición de T1 y T2 ($T1 \circ T2$) se define como

Composición

- Dados dos autómatas sin transiciones ϵ

  $$T1 = <Q,\Sigma,\Sigma',\Delta,q_0>$$

  $$T2 = <Q',\Sigma',\Sigma'',\Delta',q'_0>$$

  la composición de T1 y T2 ($T1 o T2$) se define como

  $$T1 o T2 = <QxQ',\Sigma,\Sigma'',\Delta'',(q_0,q'_0)>$$

  donde

Composición

- Dados dos autómatas sin transiciones ϵ

$$T1 = <Q,\Sigma,\Sigma',\Delta,q_0>$$

$$T2 = <Q',\Sigma',\Sigma'',\Delta',q'_0>$$

la composición de T1 y T2 (T1oT2) se define como

$$T1 o T2 = <QxQ',\Sigma,\Sigma'',\Delta'',(q_0,q'_0)>$$

donde

$$\Delta'' = \{(q_s,q_s')a \vdash b(q_d,q_d') \mid$$

$$q_s a \vdash c q_d,\ q'_s c \vdash b q'_d \text{ para algún } c \in \Sigma'\}$$

Fun with FST

Ejemplos

- Construir un FST que dado un número en binario retorne el resultado de dividirlo por 2 si la división es exacta.

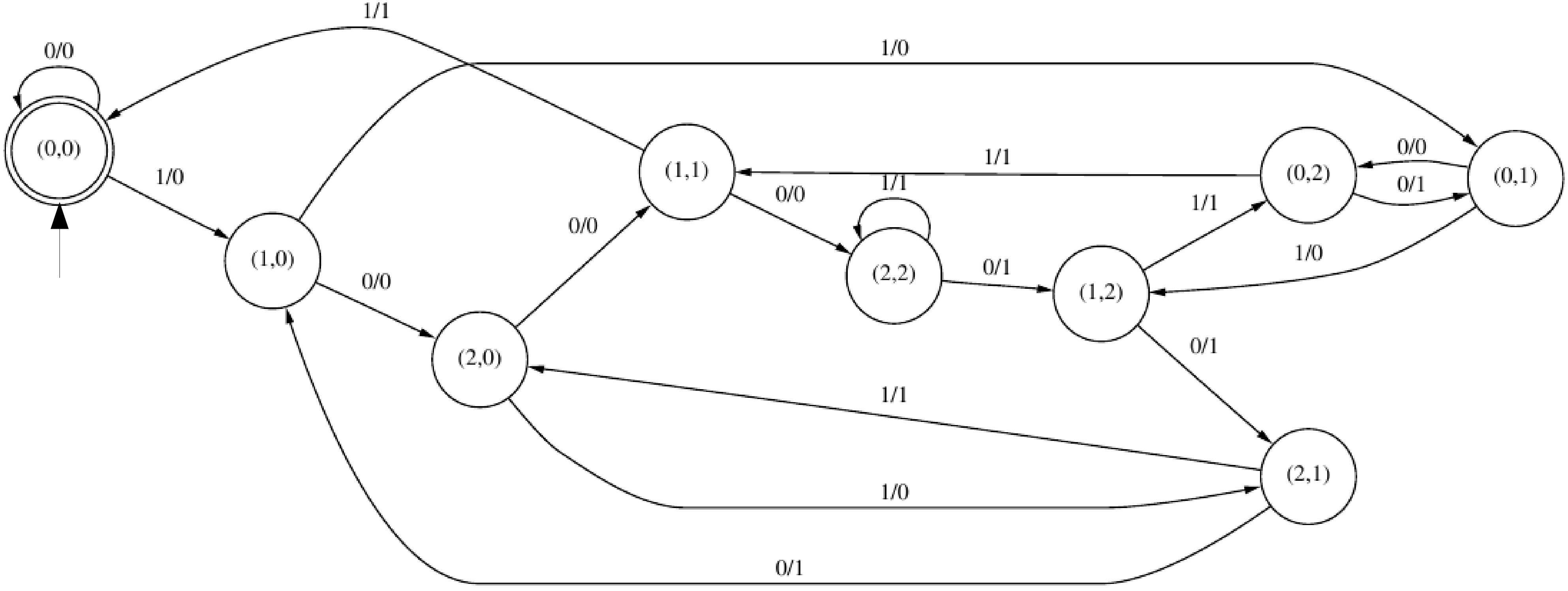
Ejemplos

- Construir un FST que dado un número en binario retorne el resultado de dividirlo por 2 si la división es exacta.
- Construir un FST que dado un número en binario retorne el resultado de dividirlo por 3 si la división es exacta.

Ejemplos

- Construir un FST que dado un número en binario retorne el resultado de dividirlo por 2 si la división es exacta.
- Construir un FST que dado un número en binario retorne el resultado de dividirlo por 3 si la división es exacta.
- Construir un FST que dado un número en binario retorne el resultado de dividirlo por 9 si la división es exacta.

Ejemplos


1/1
0/0
(0,0)
1/0
(1,0)
1/0
(1,1)
0/0
(2,0)
0/0
0/0
1/1
(2,2)
0/1
(1,2)
1/1
(0,2)
0/0
0/1
(0,1)
1/0
0/1
1/1
(2,1)
1/0
0/1

Uno de lingüística computacional

- Spelling:
  - Supongamos que queremos modelar la regla que permite agregar CO a otras palabras. Ej.
    - <CO> + design => codesign

Uno de lingüística computacional

- Spelling:
  - Supongamos que queremos modelar la regla que permite agregar CO a otras palabras. Ej.
    - <CO> + design => codesign
    - <CO> + author => coauthor

Uno de lingüística computacional

- Spelling:
  - Supongamos que queremos modelar la regla que permite agregar CO a otras palabras. Ej.
    - <CO> + design => codesign
    - <CO> + author => coauthor
    - <CO> + organizer => co-organizer

Uno de lingüística computacional

- Spelling:
  - Supongamos que queremos modelar la regla que permite agregar CO a otras palabras. Ej.
    - <CO> + design => codesign
    - <CO> + develop => codevelop
    - <CO> + author => coauthor
    - <CO> + organizer => co-organizer
- Este es un caso típico de reglas con excepciones: Hacer un FST T_g para el caso general, y un T_e para la excepción y componerlos: $T_g \, o \, T_e$.

Ultimo

- Cada vez que enviamos un SSM estamos usando un transductor:
  - {a,b,c} => 2 {d,e,f} => 3 {g,h,i} => 4
  - {j,k,l} => 5 {m,n,o} => 6 {p,q,r,s} => 7
  - {t,u,v} => 8 {w,x,y,z} => 9 {▫} => #
- Es fácil construir este transductor T_{cel} y tenemos, por ejemplo:
  - En T_{cel}: $q_0 casa\# \vdash^* 2272\#$

Ultimo

- Cada vez que enviamos un SSM estamos usando un transductor:
  - {a,b,c} => 2 {d,e,f} => 3 {g,h,i} => 4
  - {j,k,l} => 5 {m,n,o} => 6 {p,q,r,s} => 7
  - {t,u,v} => 8 {w,x,y,z} => 9 {▫} => #
- Es fácil construir este transductor T_{cel} y tenemos, por ejemplo:
  - En T_{cel}: $q_0 casa\# \vdash^* 2272\#$
- Como hacemos al reves? (de números a palabras).

Ultimo

- Cada vez que enviamos un SSM estamos usando un transductor:
  - {a,b,c} => 2   {d,e,f} => 3   {g,h,i} => 4
  - {j,k,l} => 5   {m,n,o} => 6   {p,q,r,s} => 7
  - {t,u,v} => 8   {w,x,y,z} => 9   {▫} => #
- Es fácil construir este transductor T_{cel} y tenemos, por ejemplo:
  - En T_{cel}: $q_0 casa\# \vdash^* 2272\#$
- Como hacemos al reves? (de números a palabras).
  - En inv(T_{cel}): $q_0 2272\# \vdash^* casa\#$

Ultimo

- Cada vez que enviamos un SSM estamos usando un transductor:
  - {a,b,c} => 2 {d,e,f} => 3 {g,h,i} => 4
  - {j,k,l} => 5 {m,n,o} => 6 {p,q,r,s} => 7
  - {t,u,v} => 8 {w,x,y,z} => 9 {▫} => #
- Es fácil construir este transductor T_{cel} y tenemos, por ejemplo:
  - En T_{cel}: $q_0 casa\# \vdash^* 2272\#$
- Como hacemos al reves? (de números a palabras).
  - En inv(T_{cel}): $q_0 2272\# \vdash^* casa\#$

    $q_0 2272\# \vdash^* bbqb\#$

Ultimo

- Cada vez que enviamos un SSM estamos usando un transductor:
  - {a,b,c} => 2 {d,e,f} => 3 {g,h,i} => 4
  - {j,k,l} => 5 {m,n,o} => 6 {p,q,r,s} => 7
  - {t,u,v} => 8 {w,x,y,z} => 9 {▫} => #
- Es fácil construir este transductor T_{cel} y tenemos, por ejemplo:
  - En T_{cel}: $q_0 casa\# \vdash^* 2272\#$
- Como hacemos al reves? (de números a palabras).
  - En inv(T_{cel}): $q_0 2272\# \vdash^* casa\#$

    $q_0 2272\# \vdash^* bbqb\#$
- Componer con un diccionario: T_{dic} o inv(T_{cel})

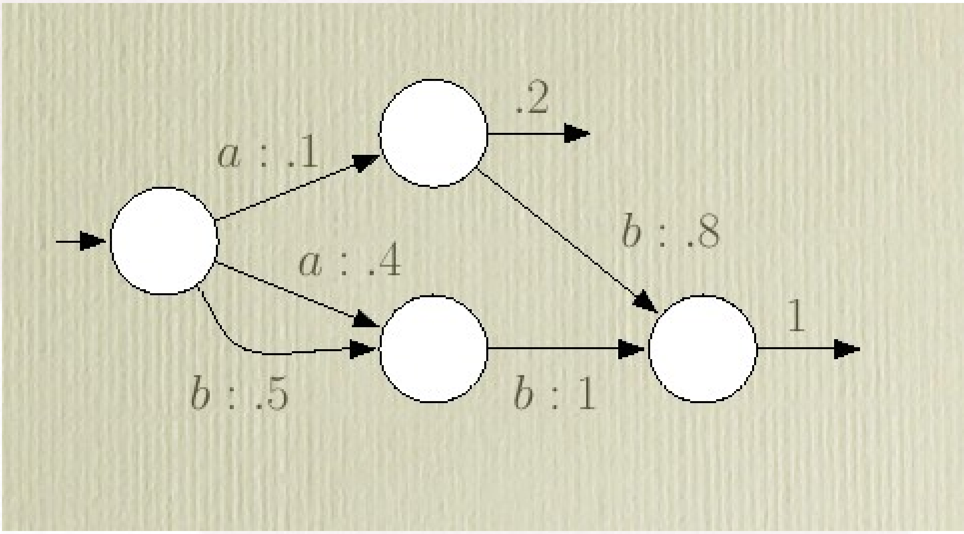
Pesos

Autómatas con pesos

- Volvamos a autómatas
- Un autómata tiene **diferentes caminos** (uno para cada palabra que reconoce, y en el caso de un autómata no determinístico, más de uno).
- Si queremos asignar **probabilidades** a las palabras de un lenguaje, una opción natural es asignarle **pesos a los caminos** del autómata que genera el lenguaje

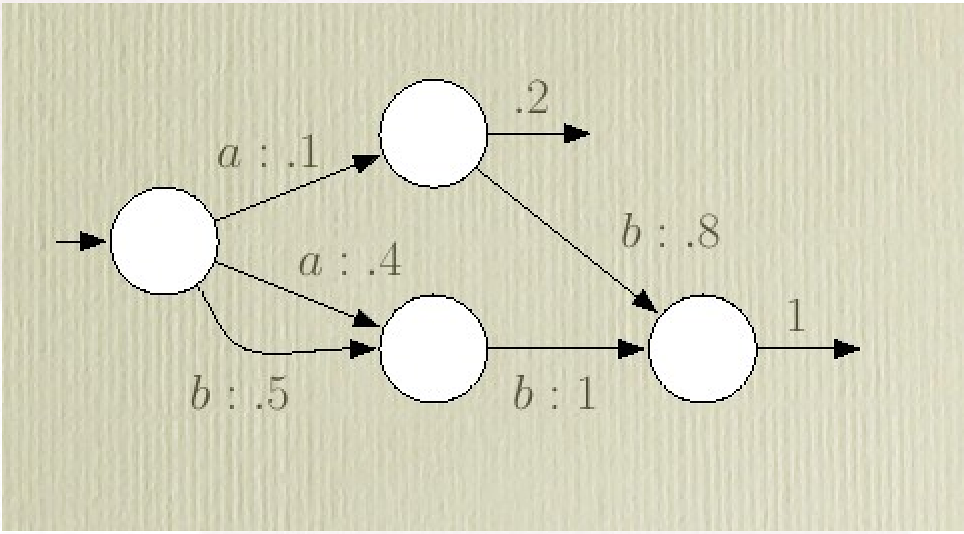
Autómatas con pesos

- Miremos este autómata



Autómatas con pesos

- Miremos este autómata



- El lenguaje que reconoce tiene sólo tres cadenas
  - a, bb, ab
- Cuál es la más probable?

Autómatas con pesos

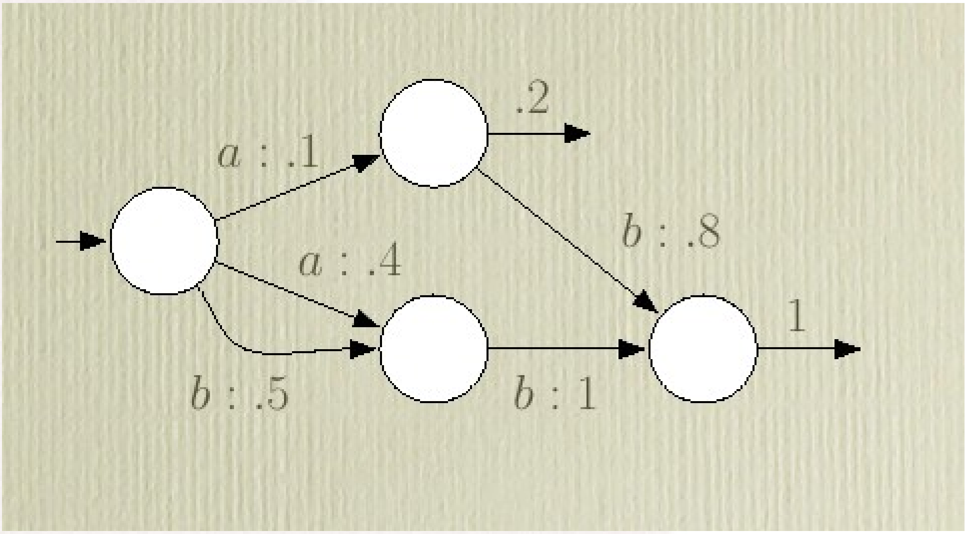
Miremos este autómata

a

bb

ab

1



Autómatas con pesos

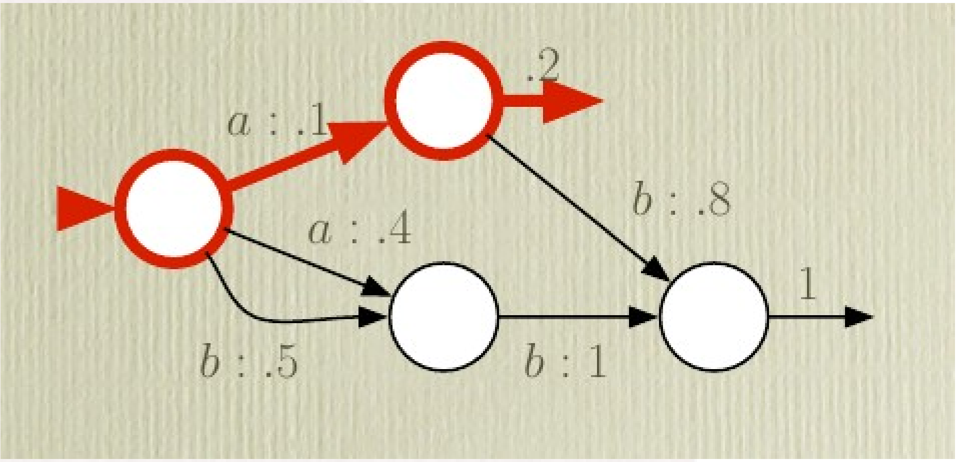
Miremos este autómata

a .02

bb

ab

1



Autómatas con pesos

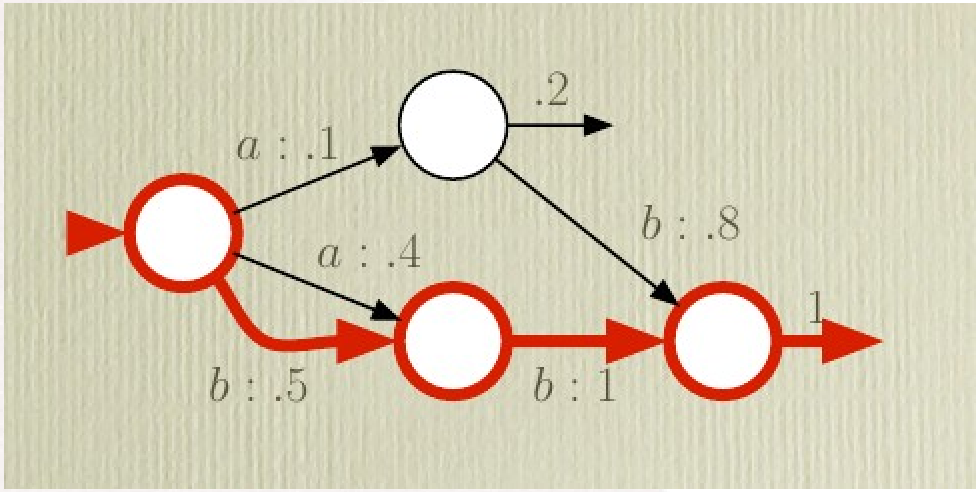
Miremos este autómata

a .02

bb .5

ab

1

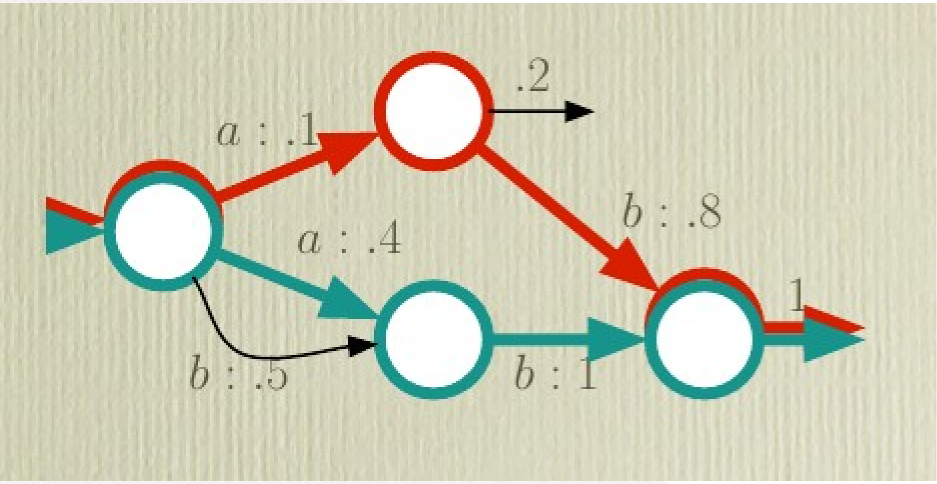


Autómatas con pesos

Miremos este autómata

a	.02
bb	.5
ab	.48

1	

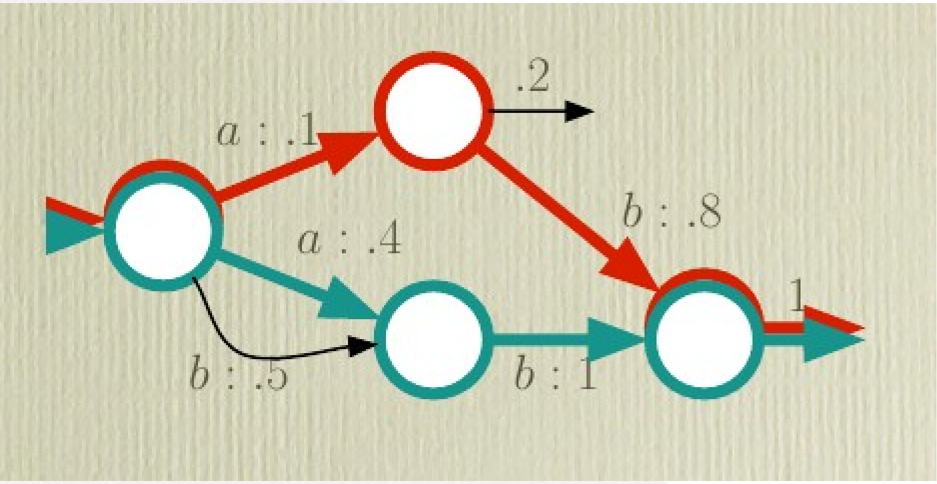


Autómatas con pesos

Miremos este autómata

a	.02
bb	.5
ab	.48

1	



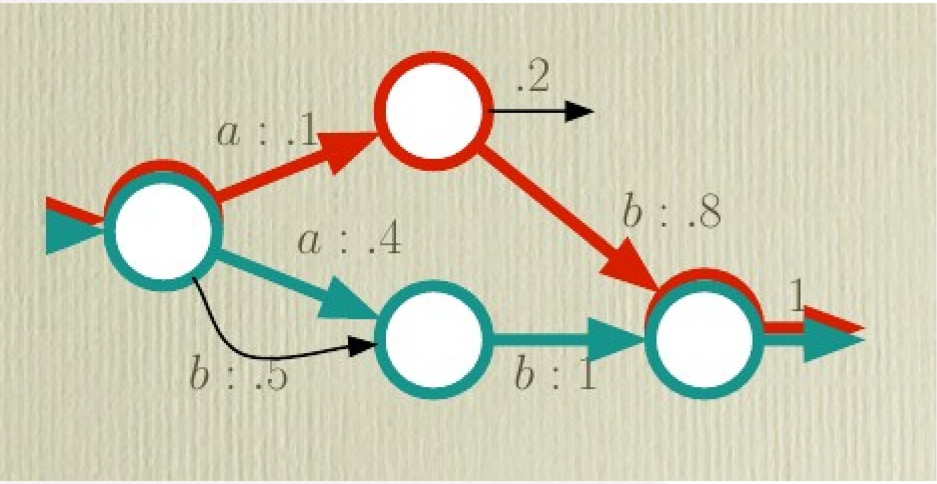
Qué estamos haciendo?

Autómatas con pesos

Miremos este autómata

a	.02
bb	.5
ab	.48

1	



Qué estamos haciendo?

- Multiplicamos sobre el camimo
- Sumamos entre caminos

Transductores otra vez

- El autómata con pesos que acabamos de ver, es un transductor, pero que genera ***probabilidades*** en vez de cadenas.
- Los transductores pueden generar ***en cualquier semi-anillo*** (semi-ring).

Semi-Anillos

- Un conjunto K con operaciones de suma y producto que satisfagan las siguientes propiedades
  - **Asociatividad +**: $(x + y) + z = x + (y + z)$
  - **Conmutatividad +**: $x + y = y + x$
  - **Asociatividad ***: $(x * y) * z = x * (y * z)$
  - **Identidad +**: $x + 0 = 0 + x = x$
  - **Identidad de ***: $x * 1 = 1 * x = x$
  - **Idempotencia 0**: $x*0 = 0*x = 0$

Ejemplos de semi-anillos

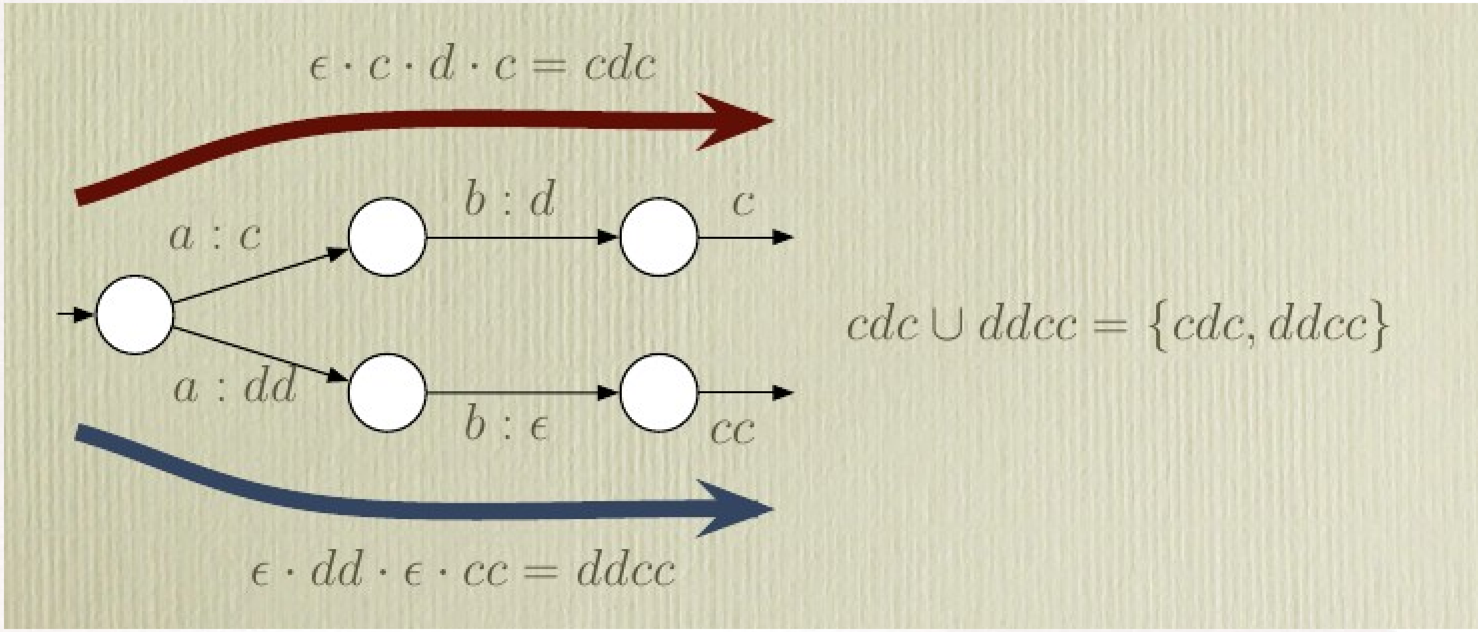
- Los conjuntos de cadenas forman un semi-anillo:
  - Suma:
  - Producto:
  - 0:
  - 1:

Ejemplos de semi-anillos

- Los conjuntos de cadenas forman un semi-anillo:
  - Suma: unión
  - Producto: concatenación
  - 0: {}
  - 1: $\{\epsilon\}$

Ejemplos de semi-anillos

Producto (.) : sobre caminos
Suma (U) : entre caminos

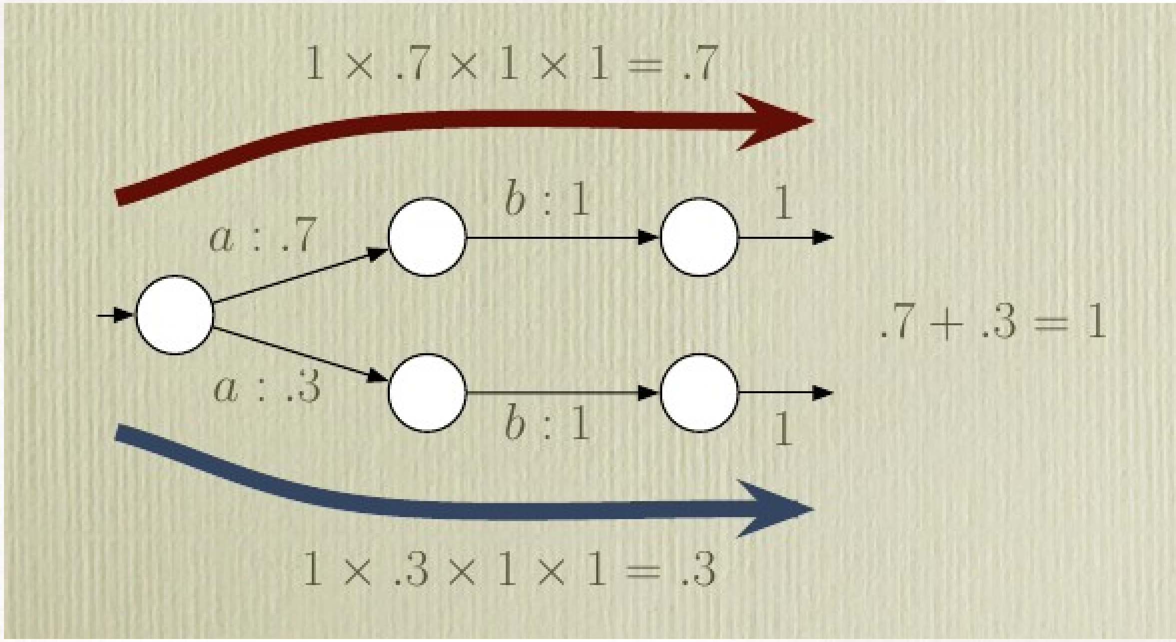


Ejemplos de semi-anillos

- Las probabilidades sobre [0..1] forman un semi-anillo:
  - Suma: +
  - Producto: *
  - 0: 0
  - 1: 1

Ejemplos de semi-anillos

Producto (*) : sobre caminos
Suma (+) : entre caminos



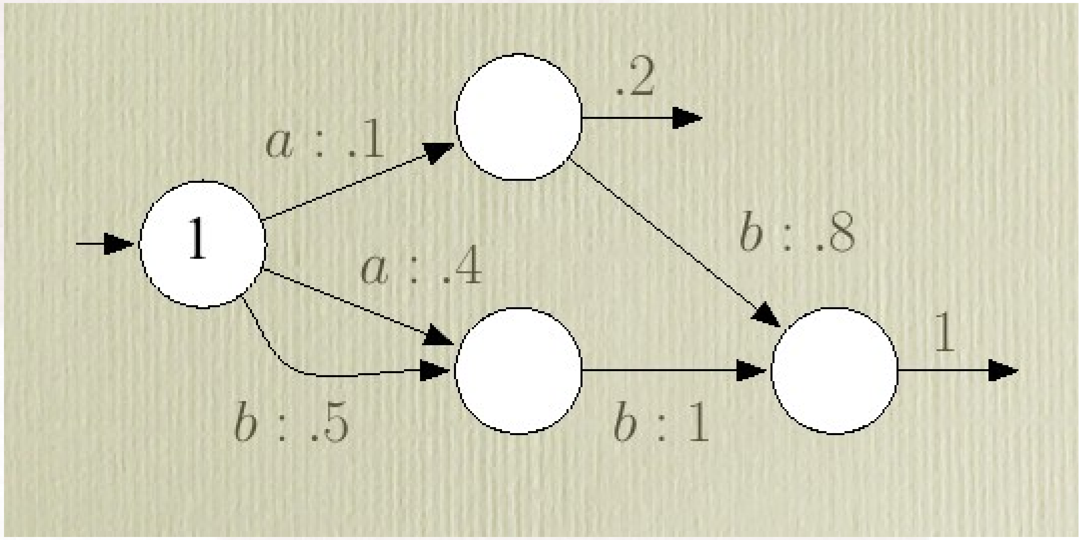
Autómatas con pesos

- Resumiendo entonces:
  - Un autómata con pesos no es otra cosa que un transductor generando sobre un semi-anillo particular.
  - Todas las propiedades de transductores se aplican a los autómatas con pesos.

Camino más corto

- Los autómatas con pesos son usados para calcular **“best guess”** en forma eficiente.
  - Si el autómata representa un conjunto de posibles opciones (e.g., alternativas de generación, scoring, etc).
  - Podemos usar el algoritmo de Dijkstra de camino más corto para encontrar la mejor opción.

Camino más corto



Transductores con pesos

- Vimos entonces que los autómatas con pesos son transductores
- Qué seran entonces los transductores con pesos?

Transductores con pesos

- Vimos entonces que los autómatas con pesos son transductores
- Qué seran entonces los transductores con pesos?
- Ayudita:
  - Siempre podemos ver una tripla como un par donde la primer componente es un par:

(a,b,c) ~ ((a,b),c)