

# Lenguajes de Consulta para “streaming” XML

Pablo Barceló

Departamento de Ciencias de la Computación  
Universidad de Chile

# XML: EXtensible Markup Language

*“XML ha sido propuesto como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en la internet, en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable...”*

Wikipedia

# Ventajas de XML

XML busca presentar información en la web de la forma más abstracta y reutilizable posible.

# Ventajas de XML

XML busca presentar información en la web de la forma más abstracta y reutilizable posible.

Algunas ventajas de XML:

- ▶ Fácil de ser entendido tanto por personas como por máquinas (al contrario de HTML).
- ▶ Permite especificar documentos que contienen tanto información como estructura semántica.
- ▶ Es **el** estándar para intercambiar información en la web (e.g., SII en Chile).

# ¿Cómo estudiar XML?

Usualmente un documento XML se consulta de la forma más simple posible: Se ingresa su URL y se obtiene el documento completo.

Sin embargo, los documentos XML son bases de datos, y en bases de datos esta no es realmente la forma en que se recupera la información.

# ¿Cómo estudiar XML?

Dado el tamaño de los actuales repositorios de información en la web y la necesidad de obtener respuestas cada vez más precisas, se hace cada vez más importante estudiar XML desde el punto de vista de las bases de datos:

- ▶ Fundamentos más claros.
- ▶ Explicación a problemas prácticos.
- ▶ Mayor eficiencia.
- ▶ Mayor granularidad, etc.

# ¿Cómo estudiar XML?

Dado el tamaño de los actuales repositorios de información en la web y la necesidad de obtener respuestas cada vez más precisas, se hace cada vez más importante estudiar XML desde el punto de vista de las bases de datos:

- ▶ Fundamentos más claros.
- ▶ Explicación a problemas prácticos.
- ▶ Mayor eficiencia.
- ▶ Mayor granularidad, etc.

Esa es la idea que guiará esta presentación.

# ¿Por qué estudiar XML?

XML ha dado un nuevo impulso al estudio de las bases de datos:

- ▶ Se considera un cambio de paradigma con respecto al modelo relacional.
- ▶ Ha reimpulsado la conexión entre teoría y práctica.
- ▶ Ha revalorizado problemas olvidados y creado también nuevos problemas.

# Tabla de contenidos

## XML: Documentos, DTDs, Lenguajes de consulta

### Lenguajes de consulta: Core XPath y FO

- Core XPath

- Lógica de primer orden

- XPath Condicional

### XML en formato 'streaming'

- XML y *streaming*

- 'Streaming' XML y FO

- Lenguajes de consulta con recursión

### Otros problemas

- 'Streaming' XML y autómatas usuales

- XML y 'data values'

Vista parcial de la BD de una biblioteca como documento XML:

```
<db>
  <book title="Algebra">
    <author name="Hungerford" aff="U. Washington">
    </author>
  </book>
  <book title="Real Analysis">
    <author name="Royden" aff="Stanford">
    </author>
  </book>
</db>
```

Un documento XML contiene información *semiestructurada*, y por tanto, puede ser mostrado como un árbol:



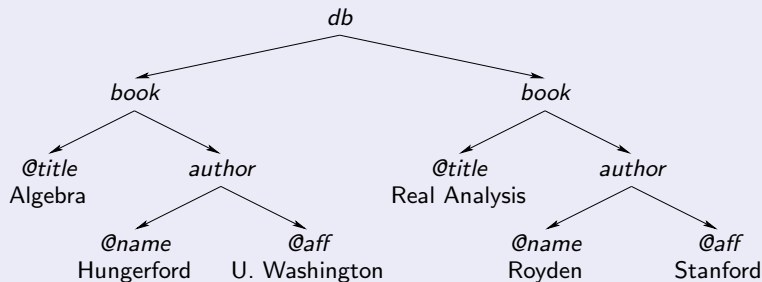
# DTDs: Document Type Definitions

XML permite también especificar contenido semántico muy básico en la forma de **DTDs**.

- ▶ Definen el esquema del documento.
- ▶ Facilitan el intercambio de datos.
- ▶ Son fáciles de especificar (se usan gramáticas libres de contexto).

# DTDs: Document Type Definitions

Por ejemplo,



satisface el siguiente DTD:  $db \rightarrow book^+$  (siempre asumimos que existe un orden entre *hermanos*).

## Lenguajes de consulta:

Permiten extraer información del documento XML y chequear validez con respecto a restricciones de integridad.

Lo ideal es que satisfazgan los siguientes criterios:

- ▶ Permitan *navegar* el documento.
- ▶ Sean suficientemente *expresivos*.
- ▶ Puedan ser *eficientemente* evaluados.
- ▶ Sea *fácil* especificar en ellos.

# Tipos de consultas

Nos enfocaremos en lenguajes que permiten realizar las siguientes clases de consultas:

- ▶ **Booleanas:** Proporcionan como respuesta un SÍ o un NO.
  - ¿Es cierto que todo libro tiene un autor?
- ▶ **Unarias:** Permiten seleccionar un conjunto de elementos del documento.
  - Entrégueme la lista de todos los libros.

# Tipos de consultas

Nos enfocaremos en lenguajes que permiten realizar las siguientes clases de consultas:

- ▶ **Booleanas:** Proporcionan como respuesta un SÍ o un NO.
  - ¿Es cierto que todo libro tiene un autor?
- ▶ **Unarias:** Permiten seleccionar un conjunto de elementos del documento.
  - Entrégueme la lista de todos los libros.

# Tipos de consultas

Nos enfocaremos en lenguajes que permiten realizar las siguientes clases de consultas:

- ▶ **Booleanas:** Proporcionan como respuesta un SÍ o un NO.
  - ¿Es cierto que todo libro tiene un autor?
- ▶ **Unarias:** Permiten seleccionar un conjunto de elementos del documento.
  - Entrégueme la lista de todos los libros.

# Tipos de consultas

Nos enfocaremos en lenguajes que permiten realizar las siguientes clases de consultas:

- ▶ **Booleanas:** Proporcionan como respuesta un SÍ o un NO.
  - ¿Es cierto que todo libro tiene un autor?
- ▶ **Unarias:** Permiten seleccionar un conjunto de elementos del documento.
  - Entrégueme la lista de todos los libros.

# Tipos de consultas

Nos enfocaremos en lenguajes que permiten realizar las siguientes clases de consultas:

- ▶ **Booleanas:** Proporcionan como respuesta un SÍ o un NO.
  - ¿Es cierto que todo libro tiene un autor?
- ▶ **Unarias:** Permiten seleccionar un conjunto de elementos del documento.
  - Entrégueme la lista de todos los libros.

# Tabla de contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# Tabla de contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

## XPath:

- ▶ Lenguaje más popular para navegar, seleccionar y extraer valores desde documentos XML.
- ▶ Parte de lenguajes más complejos como XQuery y XSLT.
- ▶ Estándar de la W3C: <http://www.w3.org/TR/xpath>.
- ▶ Veremos que combina buenas propiedades de expresividad y complejidad de evaluación de consultas.

## XPath:

- ▶ Lenguaje más popular para **navegar, seleccionar y extraer** valores desde documentos XML.
- ▶ Parte de lenguajes más complejos como XQuery y XSLT.
- ▶ Estándar de la W3C: <http://www.w3.org/TR/xpath>.
- ▶ Veremos que combina buenas propiedades de expresividad y complejidad de evaluación de consultas.

Estudiaremos un fragmento simple, pero muy útil de XPath: **Core XPath**.

Estudiaremos un fragmento simple, pero muy útil de XPath: **Core XPath**.

Core XPath tiene dos tipos de fórmulas:

- ▶ **Filtros:** Incluye las etiquetas, cerrados bajo operaciones Booleanas, y pueden verificar si un camino satisfaciendo una propiedad dada puede empezar en el nodo.
- ▶ **Fórmulas de camino:** Verifican si un filtro es cierto en el primer elemento del camino, chequean en que dirección comienza un camino, y son cerradas bajo unión y composición.

Estudiaremos un fragmento simple, pero muy útil de XPath: **Core XPath**.

Core XPath tiene dos tipos de fórmulas:

- ▶ **Filtros:** Incluye las etiquetas, cerrados bajo operaciones Booleanas, y pueden verificar si un camino satisfaciendo una propiedad dada puede empezar en el nodo.
- ▶ **Fórmulas de camino:** Verifican si un filtro es cierto en el primer elemento del camino, chequean en que dirección comienza un camino, y son cerradas bajo unión y composición.

Estudiaremos un fragmento simple, pero muy útil de XPath: **Core XPath**.

Core XPath tiene dos tipos de fórmulas:

- ▶ **Filtros:** Incluye las etiquetas, cerrados bajo operaciones Booleanas, y pueden verificar si un camino satisfaciendo una propiedad dada puede empezar en el nodo.
- ▶ **Fórmulas de camino:** Verifican si un filtro es cierto en el primer elemento del camino, chequean en que dirección comienza un camino, y son cerradas bajo unión y composición.

# XPath: Sintaxis

Formalmente viene dado por la siguiente sintaxis:

Caminos básicos:

```
paso ::= child | parent | right | left
```

Expresiones para caminos:

```
camino ::= paso | paso* | camino/camino |  
          camino ∪ camino | ?test
```

Filtros:

```
test ::= etiqueta | ⟨camino⟩ | ¬test | test ∧ test
```

# Core XPath: Ejemplos de consultas

Un ejemplo de fórmula de camino es:

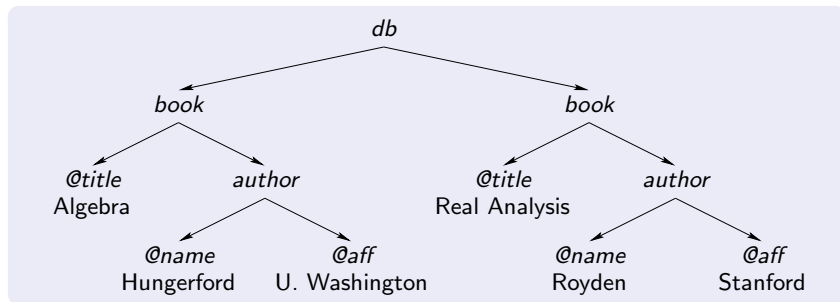
```
parent*/child*/?book
```

Un ejemplo de filtro es:

```
<parent*/child*/?book> ^ db
```

# Core XPath: Ejemplo de semántica

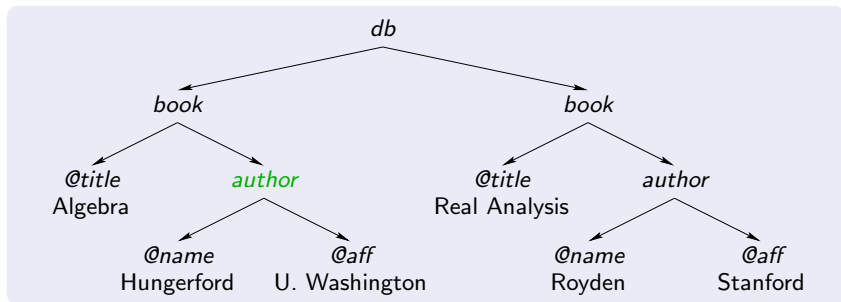
Ilustraremos la semántica de Core XPath con un ejemplo de fórmula de camino:



Consulta XPath: `parent*/child*/?book`

# Core XPath: Ejemplo de semántica

Ilustraremos la semántica de Core XPath con un ejemplo de fórmula de camino:

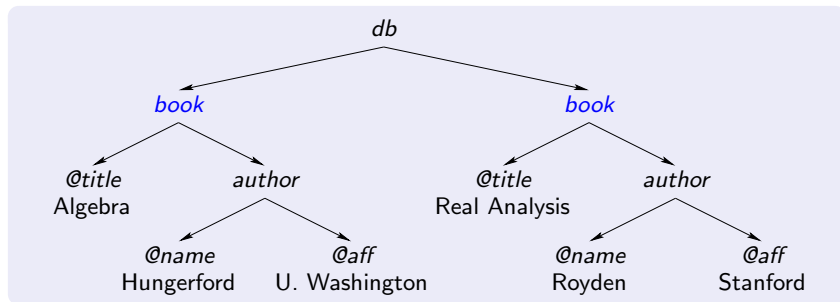


Consulta XPath: `parent*/child*/?book`

Respuesta:

# Core XPath: Ejemplo de semántica

Ilustraremos la semántica de Core XPath con un ejemplo de fórmula de camino:



Consulta XPath: `parent*/child*/?book`

Respuesta: los nodos azules.

# Core XPath es un buen lenguaje

Core XPath cumple muchos de nuestros criterios:

- ▶ Es fácil de usar.
- ▶ Permite navegar el documento.

# Core XPath es un buen lenguaje

Core XPath cumple muchos de nuestros criterios:

- ▶ Es fácil de usar.
- ▶ Permite navegar el documento.

Además, es eficiente:

**Teorema (Gottlob, Kock & Pichler; 2005)**

*Computar el conjunto de nodos que satisfacen una expresión  $\phi$  de Core XPath en un documento  $D$  se puede realizar en tiempo lineal.*

# ¿Y su expresividad?

## Problema

*¿Cómo podemos saber si una propiedad es expresable en Core XPath?*

# ¿Y su expresividad?

## Problema

*¿Cómo podemos saber si una propiedad es expresable en Core XPath?*

En otras palabras: ¿Con respecto a qué podemos medir la expresividad de Core XPath?

# Tabla de contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

**Base de datos relacional:** Información es almacenada en relaciones (tablas).

Lenguaje de consulta natural: **Lógica de primer orden (FO).**

- ▶ Ha sido estudiada por más de 100 años.
- ▶ Sintaxis y semántica bien definida.
- ▶ Expresividad bien entendida: que se puede y que no se puede decir.
- ▶ Complejidad bien entendida.

**Base de datos relacional:** Información es almacenada en relaciones (tablas).

Lenguaje de consulta natural: **Lógica de primer orden (FO)**.

- ▶ Ha sido estudiada por más de 100 años.
- ▶ Sintaxis y semántica bien definida.
- ▶ Expresividad bien entendida: que se puede y que no se puede decir.
- ▶ Complejidad bien entendida.

Pero existe un problema: Difícil de optimizar.

Un segundo lenguaje de consulta: **Álgebra Relacional**.

- ▶ Combinación de operaciones algebraicas: selección, proyección, join, unión, diferencia, ...
- ▶ Fácil de implementar y optimizar.
- ▶ Una de las razones para el éxito de las bases de datos relacionales.

Un segundo lenguaje de consulta: **Álgebra Relacional**.

- ▶ Combinación de operaciones algebraicas: selección, proyección, join, unión, diferencia, ...
- ▶ Fácil de implementar y optimizar.
- ▶ Una de las razones para el éxito de las bases de datos relacionales.

Conocemos bien el poder expresivo del álgebra relacional:

Teorema (Codd; 1972)

*Algebra relacional = FO.*

## De vuelta a XML: FO y Core XPath

Sería ideal si pudiéramos establecer una relación similar entre FO y Core XPath (recuerde que sólo consideramos consultas Booleanas y unarias).

- ▶ Esto mostraría inmediatamente cuán expresivo es Core XPath.

# De vuelta a XML: FO y Core XPath

Sería ideal si pudiéramos establecer una relación similar entre FO y Core XPath (recuerde que sólo consideramos consultas Booleanas y unarias).

- ▶ Esto mostraría inmediatamente cuán expresivo es Core XPath.

Sin embargo, esto no es posible:

## Teorema (Marx, de Rijke; 2004)

*Existe una consulta expresable en FO sobre documentos XML que no puede ser expresada en Core XPath.*

Nota: La expresividad de Core XPath es la de las consultas en FO que sólo mencionan dos variables.

# Tabla de contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# XPath condicional

Una pregunta inmediata es:

- ▶ ¿Podemos agregar operadores naturales a Core XPath, de tal forma que la lógica resultante tenga la misma expresividad que FO?

# XPath condicional

Una pregunta inmediata es:

- ▶ ¿Podemos agregar operadores naturales a Core XPath, de tal forma que la lógica resultante tenga la misma expresividad que FO?

Marx contestó afirmativa y elegantemente esta pregunta:

## Teorema (Marx; 2004)

*Las consultas expresables en FO son precisamente las que se pueden expresar en Core XPath extendido con un operador condicional (XPath condicional).*

# FO y XPath condicional: Complejidad

Además, la complejidad de evaluación de consultas de XPath condicional sigue siendo **lineal**.

# FO y XPath condicional: Complejidad

Además, la complejidad de evaluación de consultas de XPath condicional sigue siendo **lineal**.

Esto contrasta con la complejidad de evaluación de consultas en FO: **PSPACE** (complejidad usual) y **no-elemental** en el tamaño de la consulta (complejidad paramétrica [Frick & Grohe, 2002])

# FO y XPath condicional: Complejidad

Además, la complejidad de evaluación de consultas de XPath condicional sigue siendo **lineal**.

Esto contrasta con la complejidad de evaluación de consultas en FO: **PSPACE** (complejidad usual) y **no-elemental** en el tamaño de la consulta (complejidad paramétrica [Frick & Grohe, 2002])

Implica dos cosas:

- ▶ Traducciones desde FO a XPath condicional son “grandes”.
- ▶ FO es más “sucinto” que XPath condicional.

# Tabla de Contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# Tabla de Contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# Documentos XML en formato *streaming*

Hasta aquí hemos asumido que nuestro documento XML es un árbol.

Sin embargo, en muchas aplicaciones, principalmente en comercio electrónico, el documento XML se recibe *on line*, es decir, como un *stream* de datos:

*abaāaaābbābā...*

En tales aplicaciones no hay tiempo para preprocesar el documento.

- ▶ Debemos procesar el documento como si fuera una palabra con *tags* que se abren y se cierran.

# Documentos XML en formato *streaming*

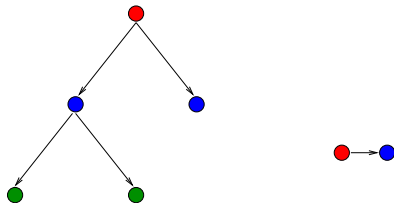
Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

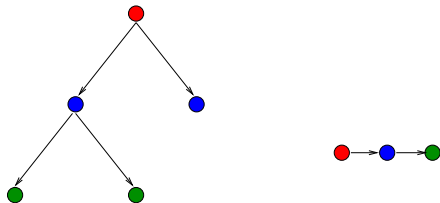
Por ejemplo,



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

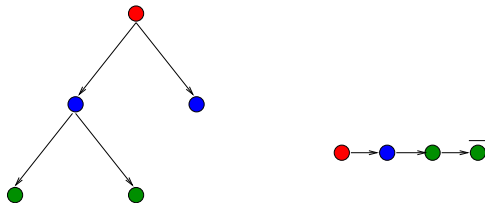
Por ejemplo,



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

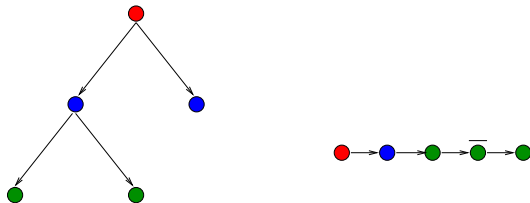
Por ejemplo,



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

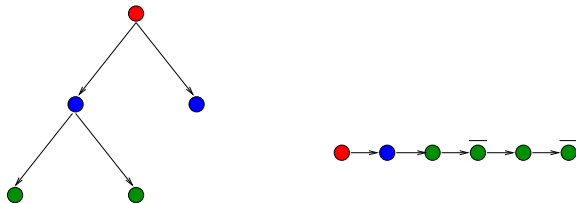
Por ejemplo,



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

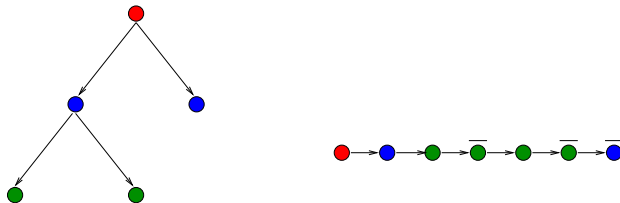
Por ejemplo,



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

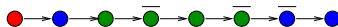
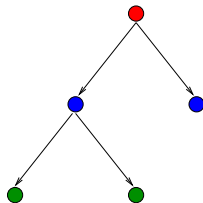
Por ejemplo,



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

Por ejemplo,

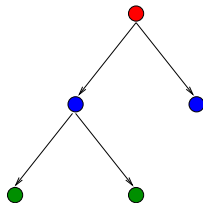




# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

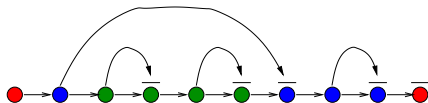
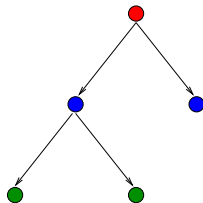
Por ejemplo,



# Documentos XML en formato *streaming*

Intuitivamente, el 'stream' de datos corresponde a una búsqueda en profundidad dentro del documento XML.

Por ejemplo,



# 'Streaming' XML y palabras anidadas

Por tanto,

Documento XML en formato 'streaming'

=

String + relación binaria bien anidada

Estas **palabras anidadas** también aparecen en otras áreas de ciencia de la computación:

- ▶ En **verificación** representan programas procedurales donde el control está dado por un autómata apilador.

# Tabla de Contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

Podemos inmediatamente extrapolar nuestra pregunta anterior:

## Pregunta

¿Existe un lenguaje a la XPath que sobre documentos XML en formato 'streaming' (palabras anidadas) tiene el mismo poder expresivo que FO?

# 'Streaming' XML, palabras anidadas y FO

Podemos inmediatamente extrapolar nuestra pregunta anterior:

## Pregunta

¿Existe un lenguaje a la XPath que sobre documentos XML en formato 'streaming' (palabras anidadas) tiene el mismo poder expresivo que FO?

La respuesta es **SI**, aunque nuestro lenguaje es ligeramente más complejo que los anteriores: tenemos que agregar operadores para diferentes tipos de caminos sobre palabras anidadas.

# 'Streaming' XML, palabras anidadas y XPath

Teorema (Alur, Arenas, B., Etesami, Immerman, Libkin, '07)

*Existe una versión de XPath sobre palabras anidadas (que contiene varios tipos diferentes de operadores de camino) que tiene la misma expresividad de FO.*

Observaciones:

- ▶ La complejidad de evaluación de consultas en esta lógica es también lineal.
- ▶ El resultado tiene implicaciones para verificación: Para ésta lógica podemos chequear satisfacibilidad en tiempo exponencial en el tamaño de la fórmula.

# Tabla de Contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# ¿Por qué queremos recursión?

Ninguno de los lenguajes que hemos visto hasta ahora puede verificar validez con respecto al siguiente DTD:

$$db \rightarrow (book \cdot book)^*$$

En otras palabras, ninguno de los lenguajes anteriores es capaz de expresar el conjunto de los lenguajes regulares (en particular, *paridad*).

# ¿Por qué queremos recursión?

Ninguno de los lenguajes que hemos visto hasta ahora puede verificar validez con respecto al siguiente DTD:

$$db \rightarrow (book \cdot book)^*$$

En otras palabras, ninguno de los lenguajes anteriores es capaz de expresar el conjunto de los lenguajes regulares (en particular, *paridad*).

Se ha sugerido que el poder expresivo de un buen lenguaje de consulta para las estructuras consideradas aquí (árboles y palabras anidadas) debe ser el de los **lenguajes regulares** (más que FO).

# Repaso de autómatas

Recordemos que sobre palabras los lenguajes regulares son exactamente los aceptados por autómatas.

# Repaso de autómatas

Recordemos que sobre palabras los lenguajes regulares son exactamente los aceptados por autómatas.

Sobre árboles también existe una noción de autómata:

- ▶ El estado en un nodo depende de su etiqueta y del estado en sus hijos.

# Repaso de autómatas

Recordemos que sobre palabras los lenguajes regulares son exactamente los aceptados por autómatas.

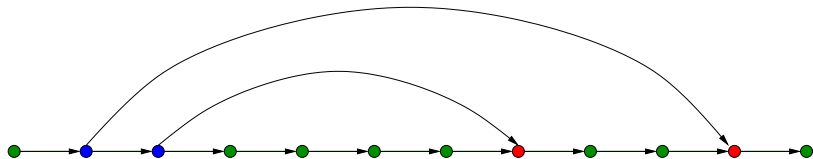
Sobre árboles también existe una noción de autómata:

- ▶ El estado en un nodo depende de su etiqueta y del estado en sus hijos.

Los autómatas sobre palabras anidadas han sido recientemente propuestos [Alur & Madhusudan, '04]:

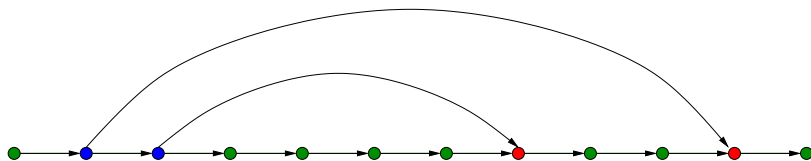
- ▶ El estado en un nodo depende de su etiqueta y del estado en su antecesor lineal y jerárquico.

# Autómatas sobre palabras anidadas



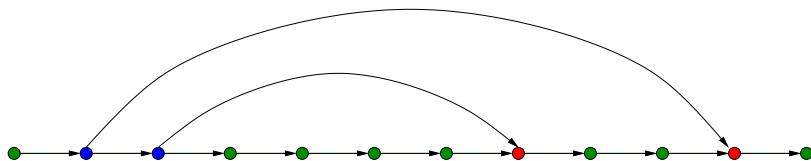
$q_0$

# Autómatas sobre palabras anidadas



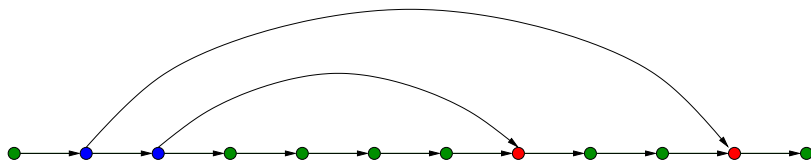
$q_0 \rightarrow q_1$

# Autómatas sobre palabras anidadas



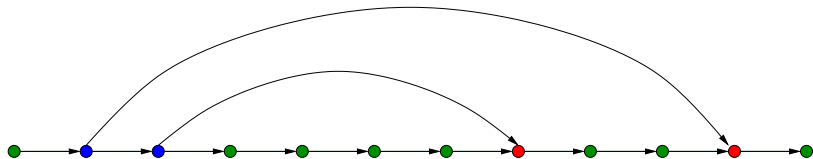
$q_0 \rightarrow q_1$

# Autómatas sobre palabras anidadas



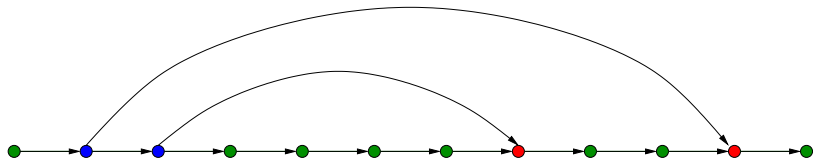
$q_0 \rightarrow q_1 \rightarrow q_2$

# Autómatas sobre palabras anidadas



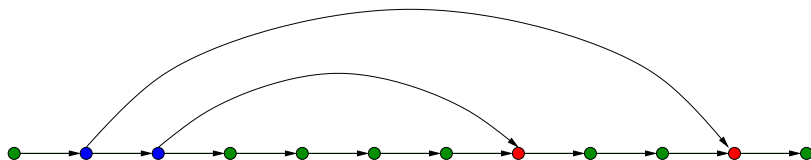
$q_0 \rightarrow q_1 \rightarrow q_2$

# Autómatas sobre palabras anidadas



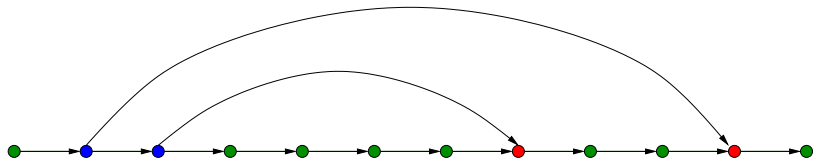
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$

# Autómatas sobre palabras anidadas



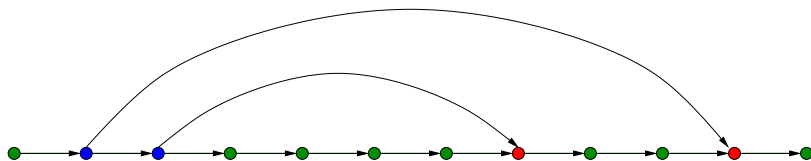
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$

# Autómatas sobre palabras anidadas



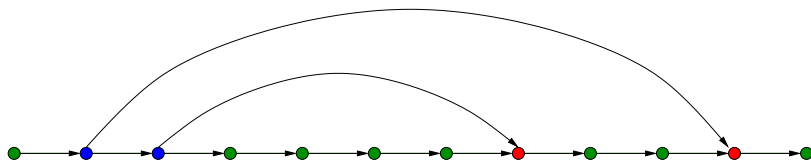
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2$

# Autómatas sobre palabras anidadas



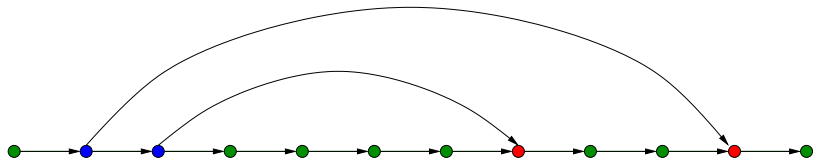
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2$

# Autómatas sobre palabras anidadas



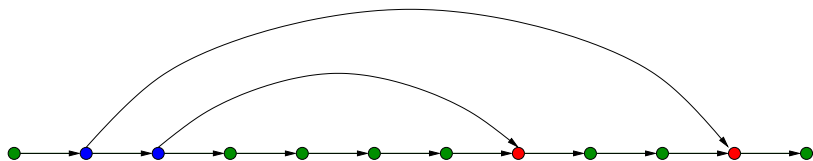
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4$

# Autómatas sobre palabras anidadas



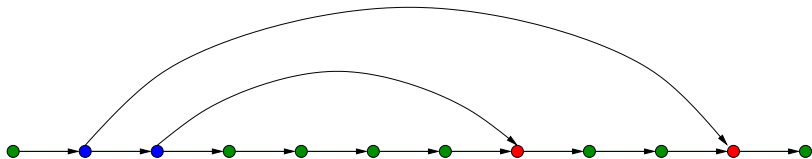
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4$

# Autómatas sobre palabras anidadas



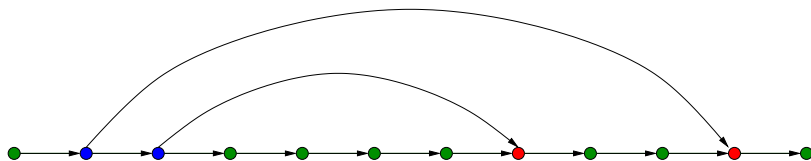
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5$

# Autómatas sobre palabras anidadas



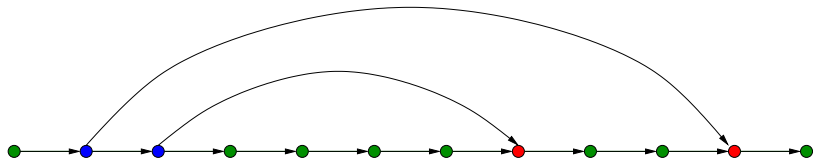
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5$

# Autómatas sobre palabras anidadas



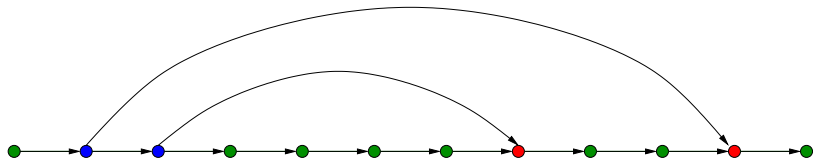
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5 \rightarrow q_1$

# Autómatas sobre palabras anidadas



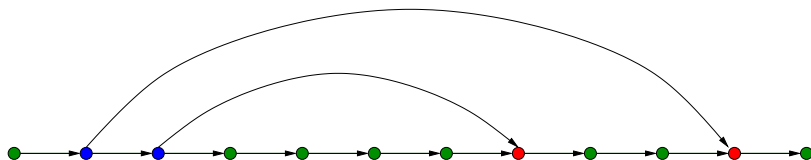
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_1$

# Autómatas sobre palabras anidadas



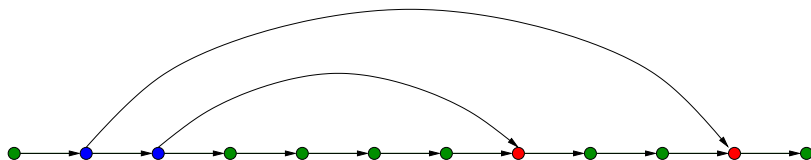
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5 \rightarrow q_1 \rightarrow q_7$

# Autómatas sobre palabras anidadas



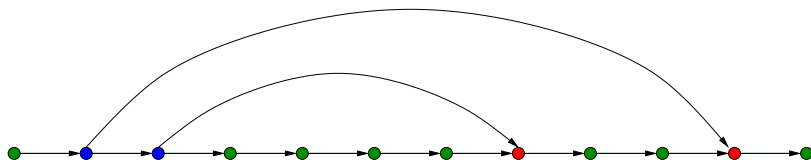
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5 \rightarrow q_1 \rightarrow q_7$

# Autómatas sobre palabras anidadas



$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5 \rightarrow q_1 \rightarrow q_7 \rightarrow \dots$

# Autómatas sobre palabras anidadas



$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5 \rightarrow q_1 \rightarrow q_7 \rightarrow \dots$

La condición de aceptación es igual que para el caso de palabras: Al último elemento se le debe haber asignado un estado final.

# Propiedades de autómatas sobre palabras anidadas

La clase de lenguajes regulares sobre palabras anidadas tiene muchas de las buenas propiedades de su similar sobre palabras [Alur & Madhusudan, '04]:

- ▶ Es cerrada bajo combinaciones Booleanas y determinización.
- ▶ Los problemas clásicos (inclusión, equivalencia, etc.) son decidibles.
- ▶ Admite de varias formulaciones equivalentes (es una clase robusta).

# Propiedades de autómatas sobre palabras anidadas

La clase de lenguajes regulares sobre palabras anidadas tiene muchas de las buenas propiedades de su simil sobre palabras [Alur & Madhusudan, '04]:

- ▶ Es cerrada bajo combinaciones Booleanas y determinización.
- ▶ Los problemas clásicos (inclusión, equivalencia, etc.) son decidibles.
- ▶ Admite de varias formulaciones equivalentes (es una clase robusta).

En particular, tienen la misma expresividad que la extensión de FO con cuantificación sobre conjuntos (MSO).

# Caracterizaciones de MSO sobre palabras anidadas

En [Arenas, B., Libkin, '07] logramos caracterizar el poder expresivo de MSO sobre palabras anidadas con ciertas lógicas temporales que admiten una evaluación de consultas eficiente (lineal).

Además extendemos este resultado a consultas unarias donde no existe una noción establecida de autómatas. La evaluación continua siendo lineal.

Este resultado podría ser adaptado para otros lenguajes interesantes, e.g. datalog.

# Tabla de Contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# Tabla de Contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# 'Streaming' XML y memoria finita

Los autómatas sobre palabras anidadas no tienen memoria finita:  
La altura del stack depende del nivel de anidación.

Algunos autores piensan que esto es un problema. Por tanto,

- ▶ ¿Cuáles clases de palabras anidadas pueden ser reconocidas usando autómatas usuales (memoria finita)?

[Segoufin & Vianu; 2002] comenzó el estudio de este problema (validación con respecto a DTDs, consultas unarias, etc.)

# Tabla de Contenidos

XML: Documentos, DTDs, Lenguajes de consulta

Lenguajes de consulta: Core XPath y FO

Core XPath

Lógica de primer orden

XPath Condicional

XML en formato 'streaming'

XML y *streaming*

'Streaming' XML y FO

Lenguajes de consulta con recursión

Otros problemas

'Streaming' XML y autómatas usuales

XML y 'data values'

# XML y 'data values'

El gran problema hasta ahora de nuestro modelo es que asumimos que nuestro alfabeto es finito. Sin embargo, los documentos XML realmente contienen 'data values' en cada nodo (strings, IDrefs, etc).

Ninguno de nuestros resultados extiende a este nuevo modelo. Además con este modelo muchos problemas básicos se vuelven indecidibles.

Hasta que no obtengamos resultados sobre este nuevo modelo no podemos decir honestamente que hemos homologado el resultado de Codd sobre documentos XML.