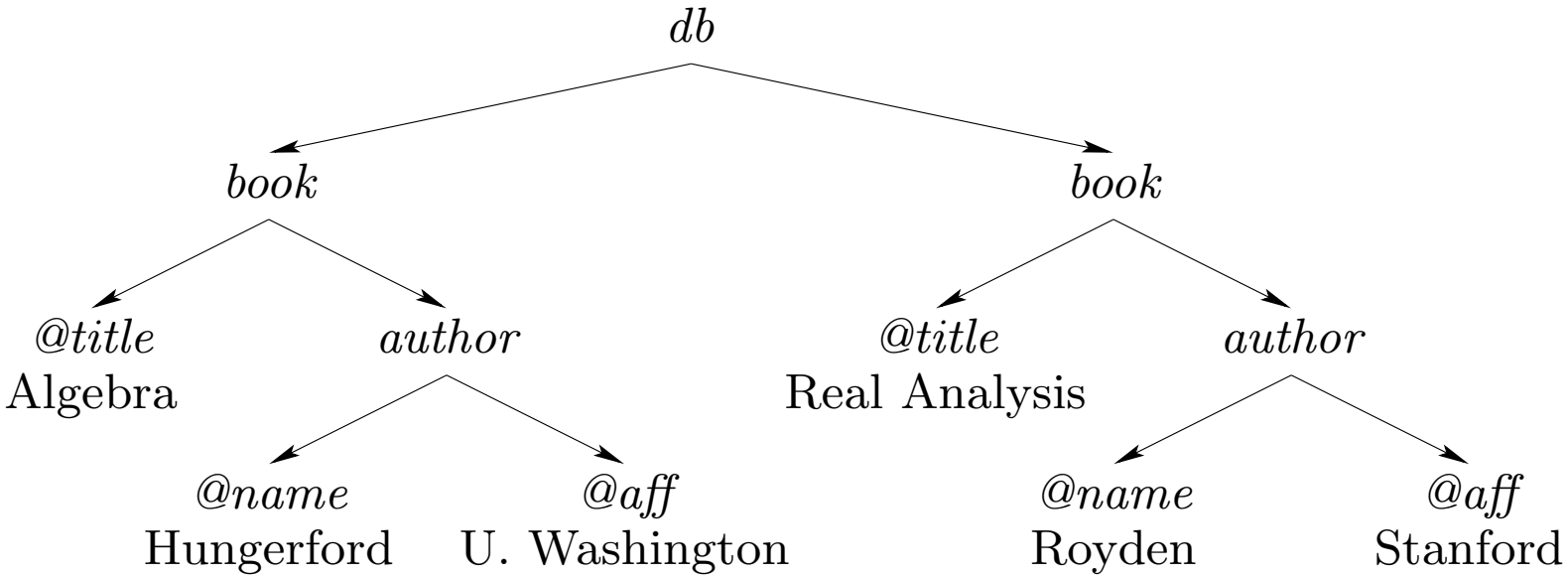
IIC3432 - Tópicos Avanzados en Bases de Datos
Extracción de Información en XML

Nuestro objetivo: Documentos 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>
```

Documentos XML como árboles



Lenguajes de consulta para XML: Operaciones básicas

Filtrar: Seleccionar valores desde un documento XML.

- Navegación, selección, extracción.

Mezclar: Integrar valores desde múltiples fuentes.

- Join, agregación.

Transformar valores desde un esquema a otro.

- Construcción de documentos XML.

Lenguajes de consulta para XML: Algunos ejemplos

XPath:

- Lenguaje más popular para navegar, seleccionar y extraer valores desde documentos XML.
- Parte de lenguajes más complejos como XQuery y XSLT.

XQuery:

- Genera documentos XML como respuesta.
- Incluye join y agregación.

XSLT:

- Lenguaje de patrones.
- Puede generar como respuesta documentos XML, HTML, texto u otros formatos.

Lenguajes de consulta para XML: Algunos ejemplos

XPath:

- Lenguaje más popular para navegar, seleccionar y extraer valores desde documentos XML.
- Parte de lenguajes más complejos como XQuery y XSLT.

XQuery:

- Genera documentos XML como respuesta.
- Incluye join y agregación.

XSLT:

- Lenguaje de patrones.
- Puede generar como respuesta documentos XML, HTML, texto u otros formatos.

XML Path Language (XPath)

Estándar de la W3C: `http://www.w3.org/TR/xpath`.

Lenguaje para navegar, seleccionar nodos y extraer valores.

Algunas implementaciones:

XALAN	:	Apache Foundation (XSLT)
XT	:	James Clark (XSLT)
SAXON	:	Michael Kay (XSLT y XQuery)

XML Path Language (XPath)

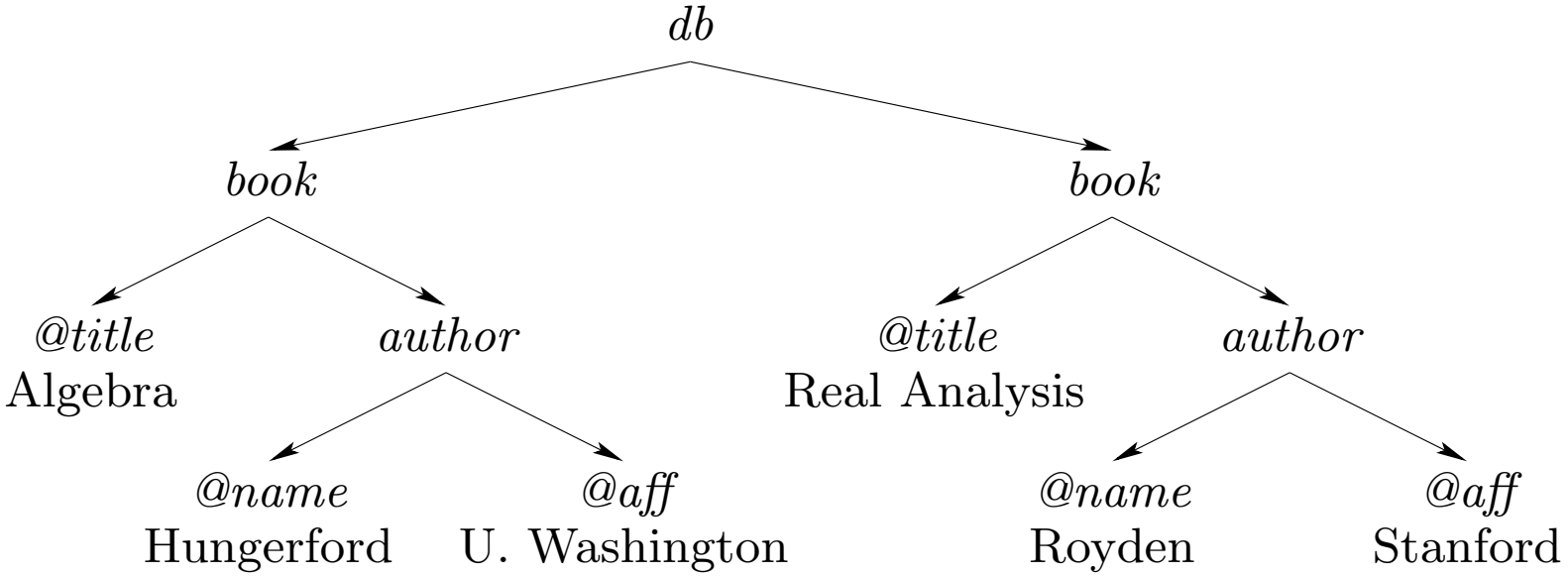
Estándar de la W3C: `http://www.w3.org/TR/xpath`.

Lenguaje para navegar, seleccionar nodos y extraer valores.

Algunas implementaciones:

XALAN	:	Apache Foundation (XSLT)
XT	:	James Clark (XSLT)
SAXON	:	Michael Kay (XSLT y XQuery)

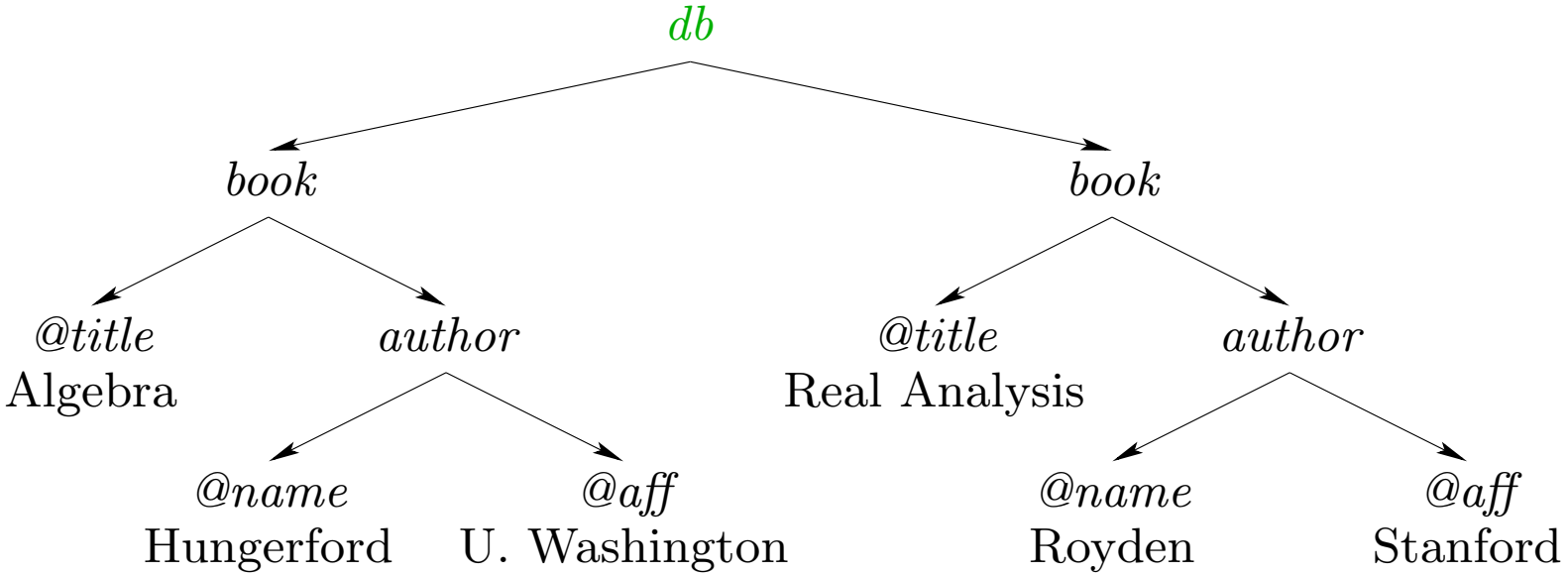
Core XPath: Primer ejemplo



Consulta XPath: `child`/?*book*

Respuesta:

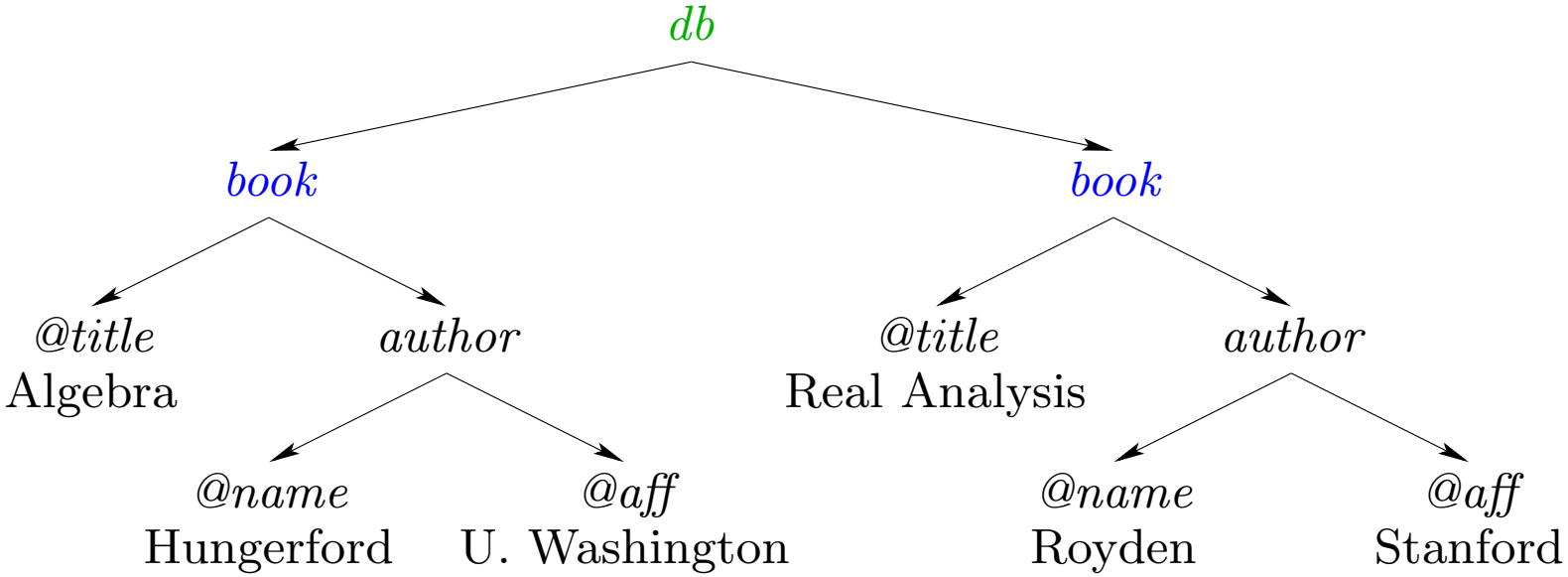
Core XPath: Primer ejemplo



Consulta XPath: `child`/?*book*

Respuesta:

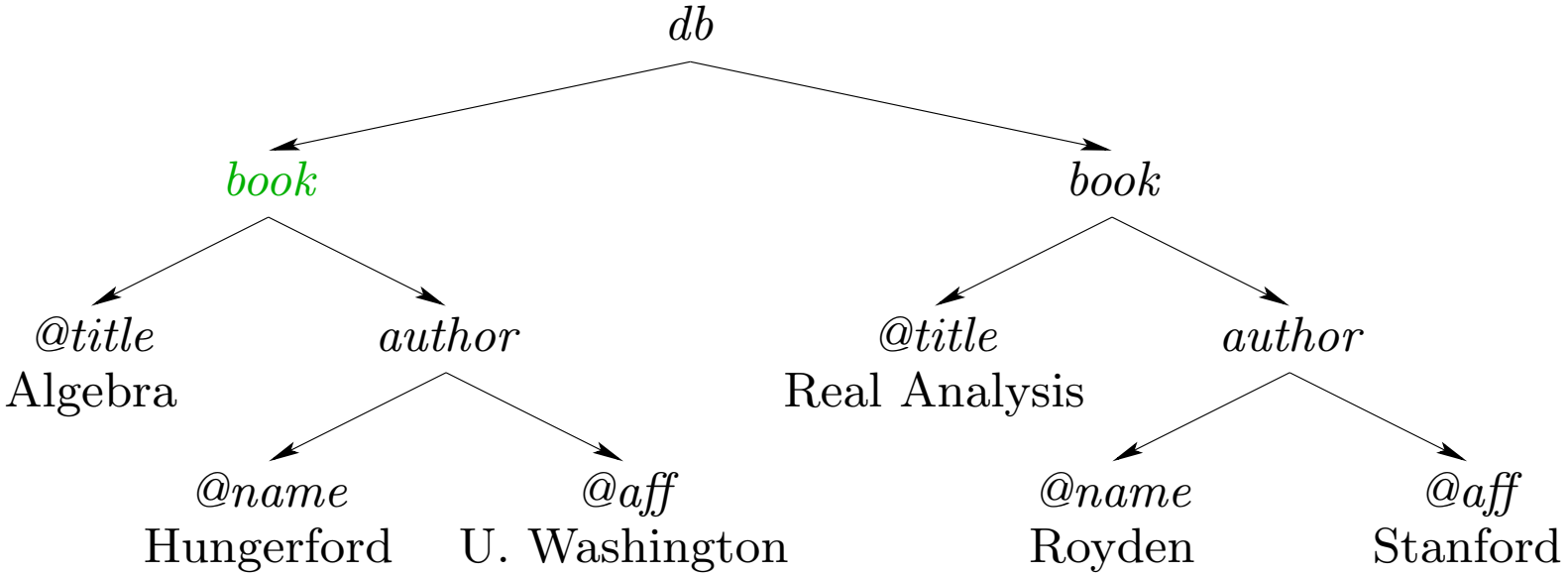
Core XPath: Primer ejemplo



Consulta XPath: `child`/?*book*

Respuesta: nodos azules

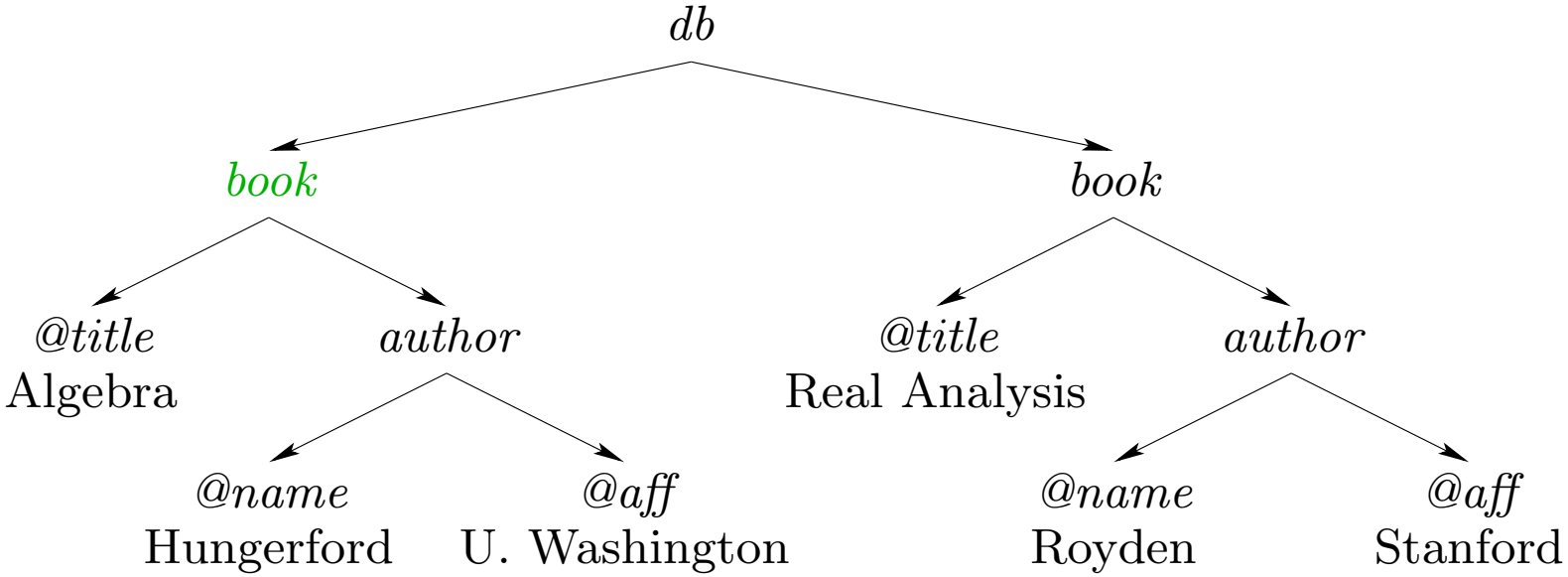
Core XPath: Segundo ejemplo



Consulta XPath: `child`/?*book*

Respuesta:

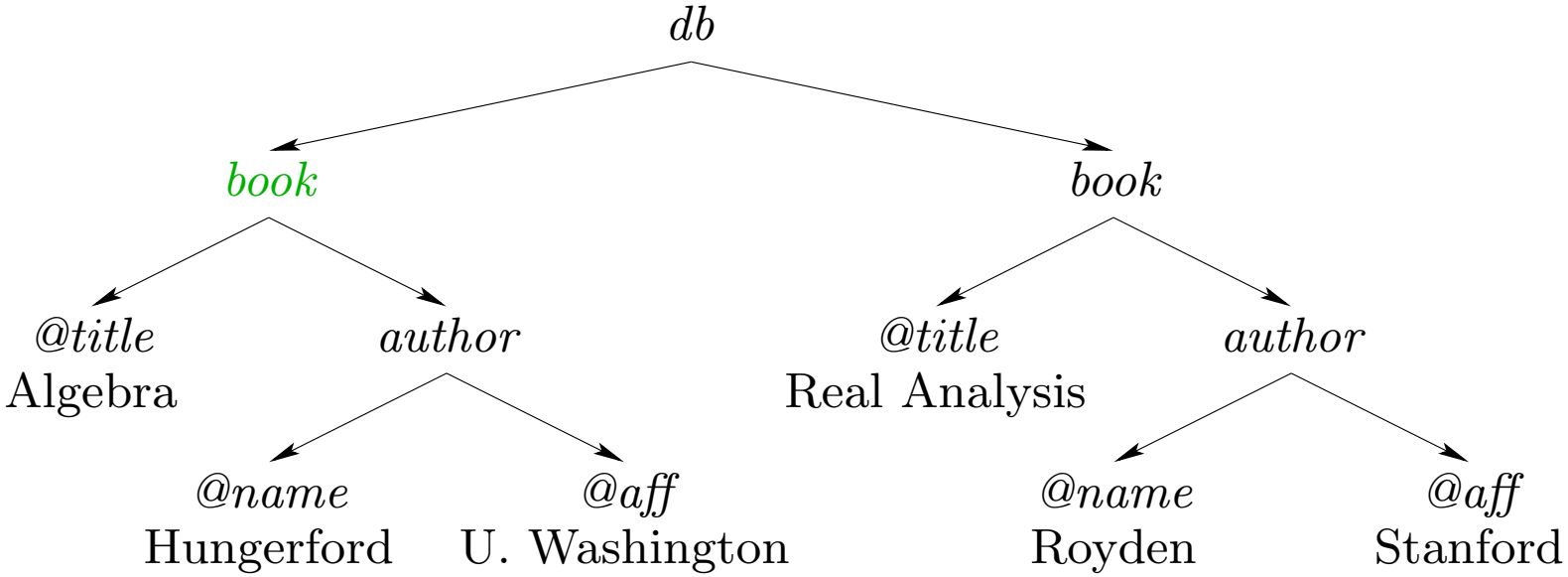
Core XPath: Segundo ejemplo



Consulta XPath: `child`/?*book*

Respuesta: $\emptyset$

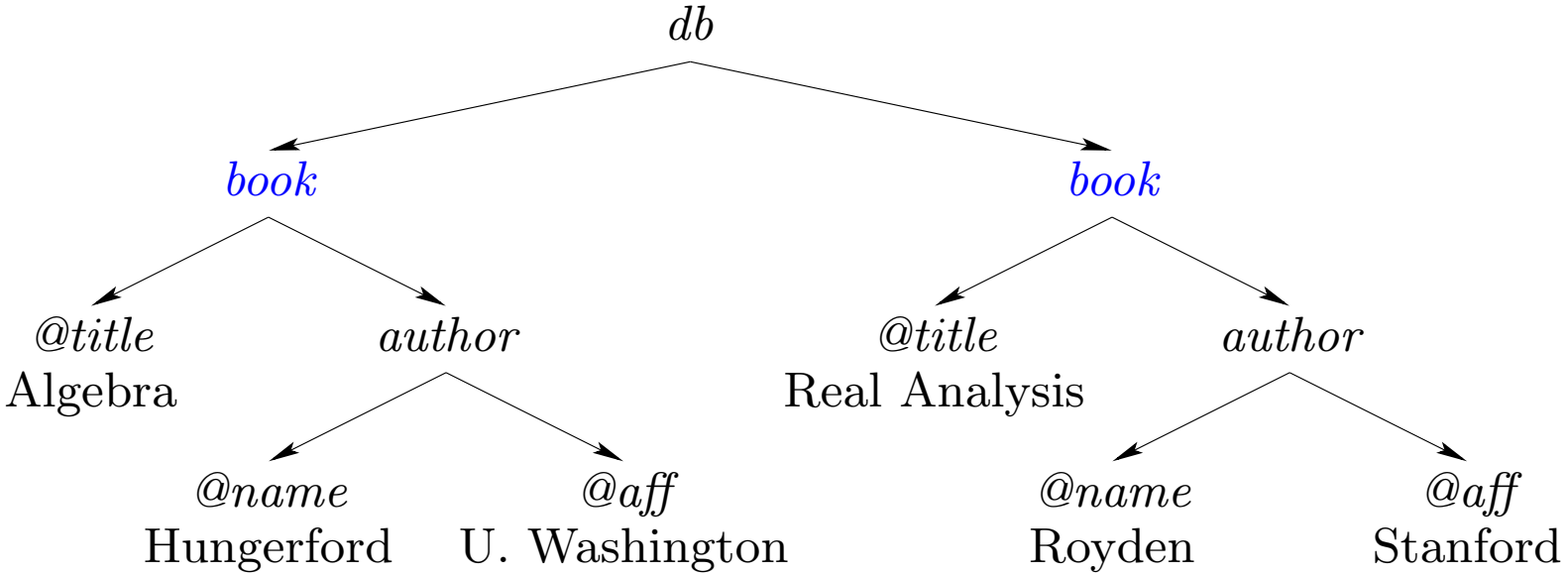
Core XPath: Tercer ejemplo



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

Respuesta:

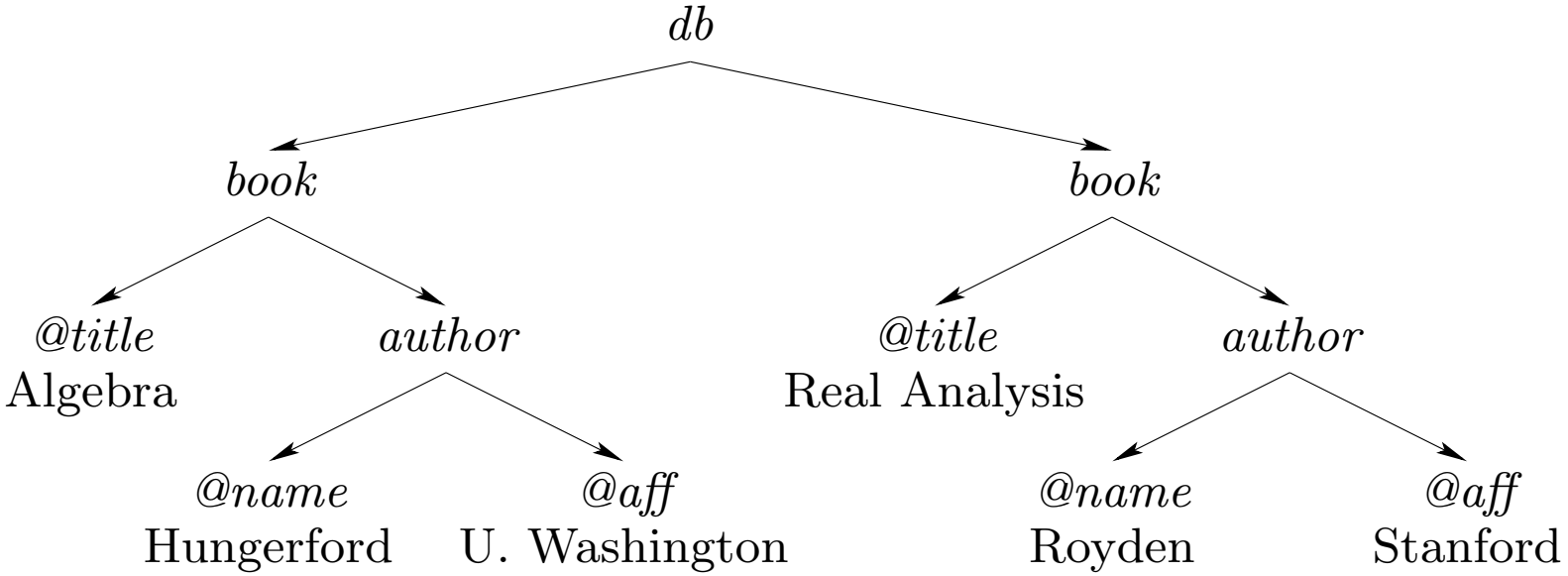
Core XPath: Tercer ejemplo



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

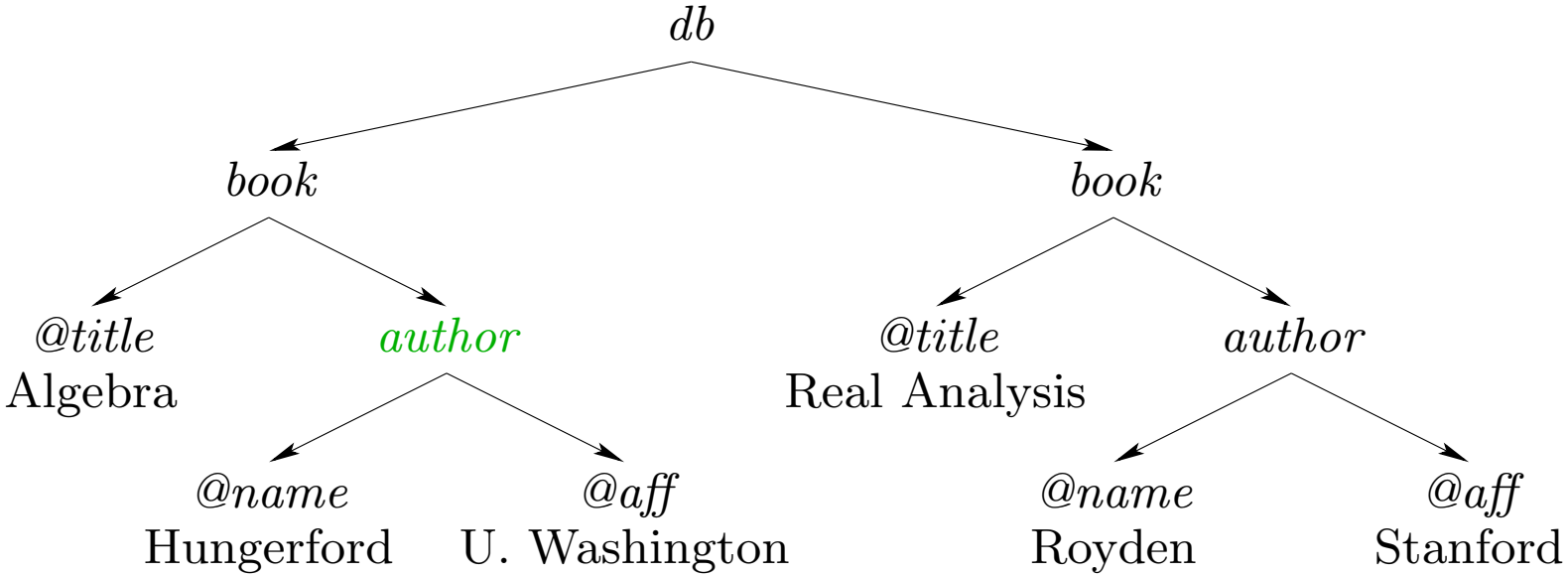
Respuesta: nodos azules

Core XPath: Ultimo ejemplo



Consulta XPath: $\texttt{parent}^*/\texttt{child}^*/?book$

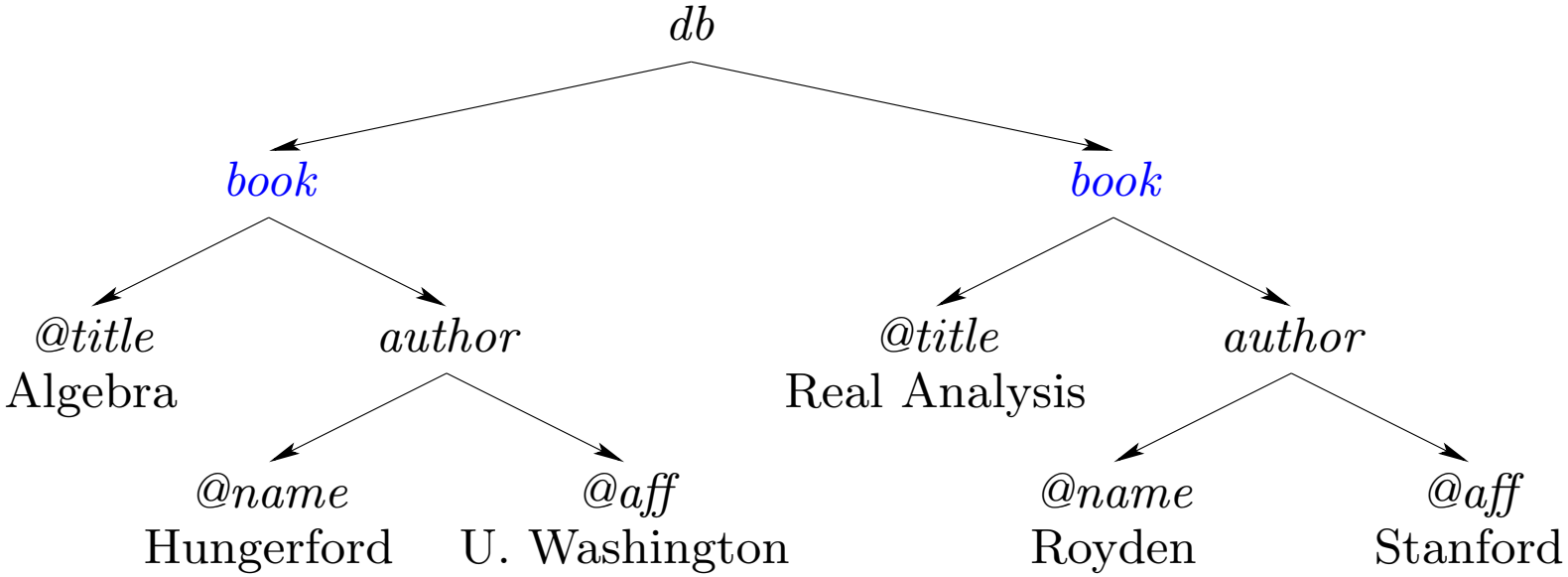
Core XPath: Ultimo ejemplo



Consulta XPath: $\texttt{parent}^*/\texttt{child}^*/?book$

Respuesta:

Core XPath: Ultimo ejemplo



Consulta XPath: $\texttt{parent}^*/\texttt{child}^*/?book$

Respuesta: nodos azules

Core XPath: Sintaxis

Caminos básicos:

$$\texttt{paso} \quad ::= \quad \texttt{child} \mid \texttt{parent} \mid \texttt{right} \mid \texttt{left}$$

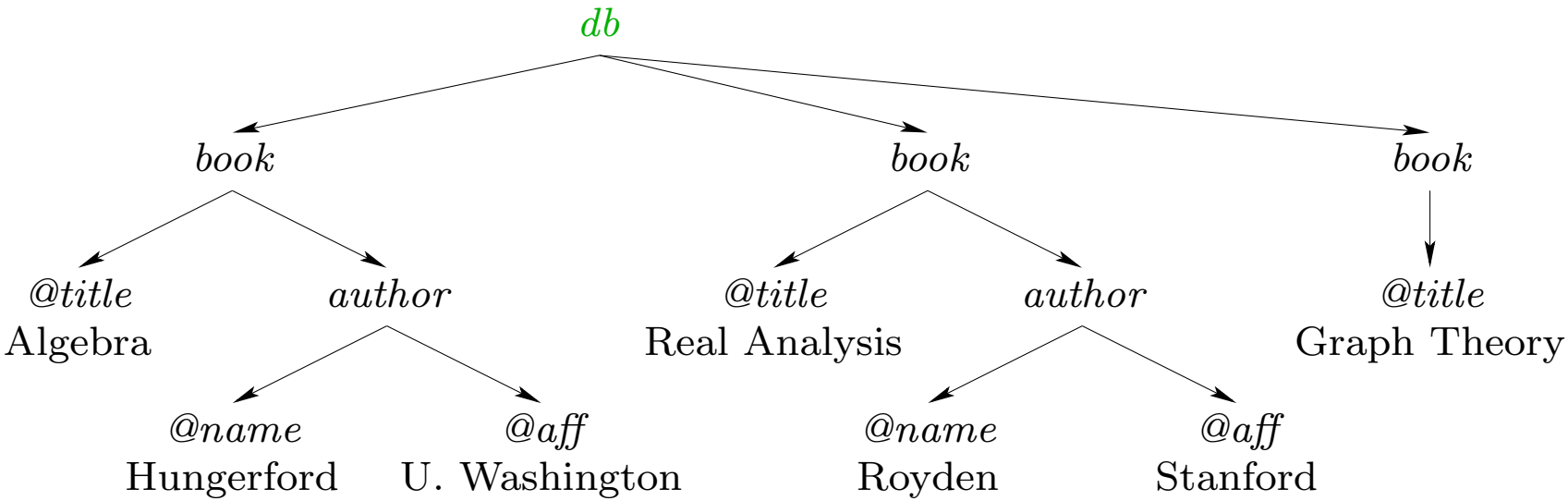
Expresiones para caminos:

$$\begin{aligned} \texttt{camino} \quad ::= \quad & \texttt{paso} \mid \texttt{paso}^* \mid \texttt{camino/camino} \mid \\ & \texttt{camino} \cup \texttt{camino} \mid ?\texttt{test} \end{aligned}$$

Filtros:

$$\texttt{test} \quad ::= \quad \textit{nombre} \mid \langle \texttt{camino} \rangle \mid \neg\texttt{test} \mid \texttt{test} \wedge \texttt{test}$$

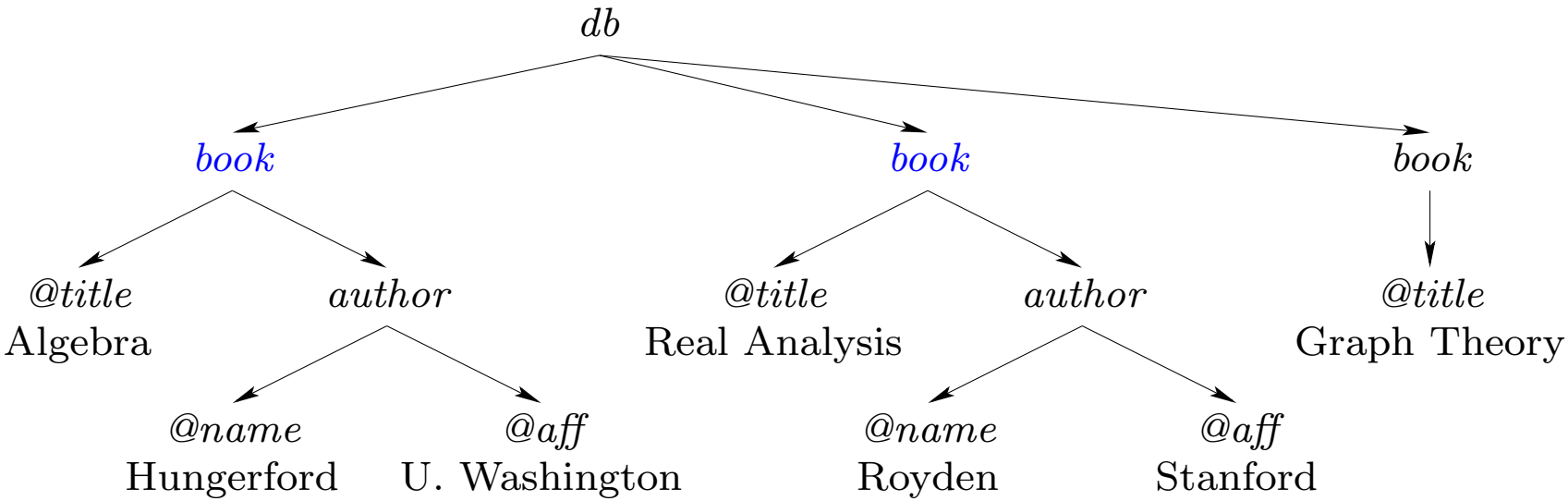
Core XPath: Otro ejemplo



Consulta XPath: $\texttt{child}/?book/?\langle\texttt{child}/?author\rangle$

Respuesta:

Core XPath: Otro ejemplo



Consulta XPath: $\texttt{child}/?book/?\langle\texttt{child}/?author\rangle$

Respuesta: nodos azules

Core XPath: Semántica para caminos básicos

Dado: Árbol $T = (D, \lambda)$.

$$
\begin{aligned}
[\![\texttt{child}]\!]_T &= \{(n_1, n_2) \mid n_2 \text{ es hijo de } n_1 \text{ en } T\} \\
[\![\texttt{parent}]\!]_T &= \{(n_1, n_2) \mid n_2 \text{ es el padre de } n_1 \text{ en } T\} \\
[\![\texttt{right}]\!]_T &= \{(n_1, n_2) \mid \text{ existe nodo } n \text{ en } T \text{ tal que } n_1 \text{ es el } i\text{-ésimo hijo} \\
&\quad \text{de } n \text{ y } n_2 \text{ es el } (i+1)\text{-ésimo hijo de } n \text{ en } T\} \\
[\![\texttt{left}]\!]_T &= \{(n_1, n_2) \mid \text{ existe nodo } n \text{ en } T \text{ tal que } n_1 \text{ es el} \\
&\quad (i+1)\text{-ésimo hijo de } n \text{ y } n_2 \text{ es el } i\text{-ésimo hijo de } n \text{ en } T\}
\end{aligned}
$$

Core XPath: Semántica para caminos complejos

$$
\begin{aligned}
[\![\texttt{camino}_1/\texttt{camino}_2]\!]_T &= \{(n_1, n_2) \mid \text{existe } n_3 \text{ tal que} \\
&\quad (n_1, n_3) \in [\![\texttt{camino}_1]\!]_T \text{ y } (n_3, n_2) \in [\![\texttt{camino}_2]\!]_T\} \\
[\![\texttt{camino}_1 \cup \texttt{camino}_2]\!]_T &= [\![\texttt{camino}_1]\!]_T \cup [\![\texttt{camino}_2]\!]_T \\
[\![\texttt{paso}^*]\!]_T &= \{(n, n) \mid n \in D\} \cup [\![\texttt{paso}]\!]_T \cup [\![\texttt{paso}/\texttt{paso}]\!]_T \cup \cdots \\
&\quad [\![\texttt{paso}/\texttt{paso}/\texttt{paso}]\!]_T \cup \cdots \\
[\![?\texttt{test}]\!]_T &= \{(n, n) \mid n \in [\![\texttt{test}]\!]_T\}
\end{aligned}
$$

Core XPath: Semántica para filtros

$$
\begin{aligned}
[\![nombre]\!]_T &= \{n \text{ en } T \mid \lambda(n) = nombre\} \\
[\![\langle \texttt{camino} \rangle]\!]_T &= \{n \text{ en } T \mid \text{ existe } n' \text{ en } T \text{ tal que } (n, n') \in [\![\texttt{camino}]\!]_T\} \\
[\![\neg \texttt{test}]\!]_T &= \{n \text{ en } T \mid n \notin [\![\texttt{test}]\!]_T\} \\
[\![\texttt{test}_1 \wedge \texttt{test}_2]\!]_T &= [\![\texttt{test}_1]\!]_T \cap [\![\texttt{test}_2]\!]_T
\end{aligned}
$$

¿Cuándo es *bueno* un lenguaje de consulta?

Criterios esenciales:

- Expresividad.
- Complejidad.

¡Estos objetivos se contraponen!

- Vamos a ver que pasa en el caso de Core XPath.

Core XPath: Complejidad

Enfoque ingenuo para evaluar una consulta: Usar un algoritmo recursivo que procesa secuencialmente la consulta.

- Usado en la mayoría de las implementación de XPath.

En XALAN y XT: Si $Q = p_1/p_2/\cdots/p_k$ $(k \geq 1)$, donde cada p_i es un paso o un test, entonces Q es procesada de la siguiente forma.

procesar(Q: consulta, T: árbol, n: nodo)
  $N := \{n' \mid (n, n') \in [\![p_1]\!]_T\}$
  if $k = 1$ **then return** N
  else
    $R := \emptyset$
    for each $n' \in N$ **do** $R := R \cup$ **procesar**$(p_2/\cdots/p_k, T, n')$
    return R

Core XPath: Complejidad

¡Enfoque ingenuo es exponencial!

- Incluso si consideramos documentos de tamaño fijo.

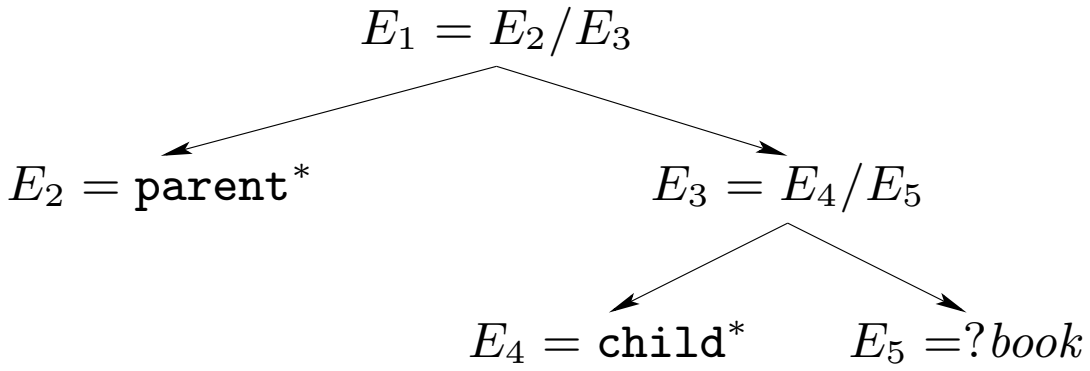
Ejemplo [GKP05]: Considere el documento usado antes y la siguiente secuencia de consultas.

$$
\begin{aligned}
Q_1 &= ?db/\texttt{child}/?book \\
Q_2 &= ?db/\texttt{child}/?book/\texttt{parent}/?db/\texttt{child}/?book \\
Q_3 &= ?db/\texttt{child}/?book/\texttt{parent}/?db/\texttt{child}/?book/\texttt{parent}/?db/\texttt{child}/?book \\
&\ldots
\end{aligned}
$$

Core XPath: Evaluación bottom-up

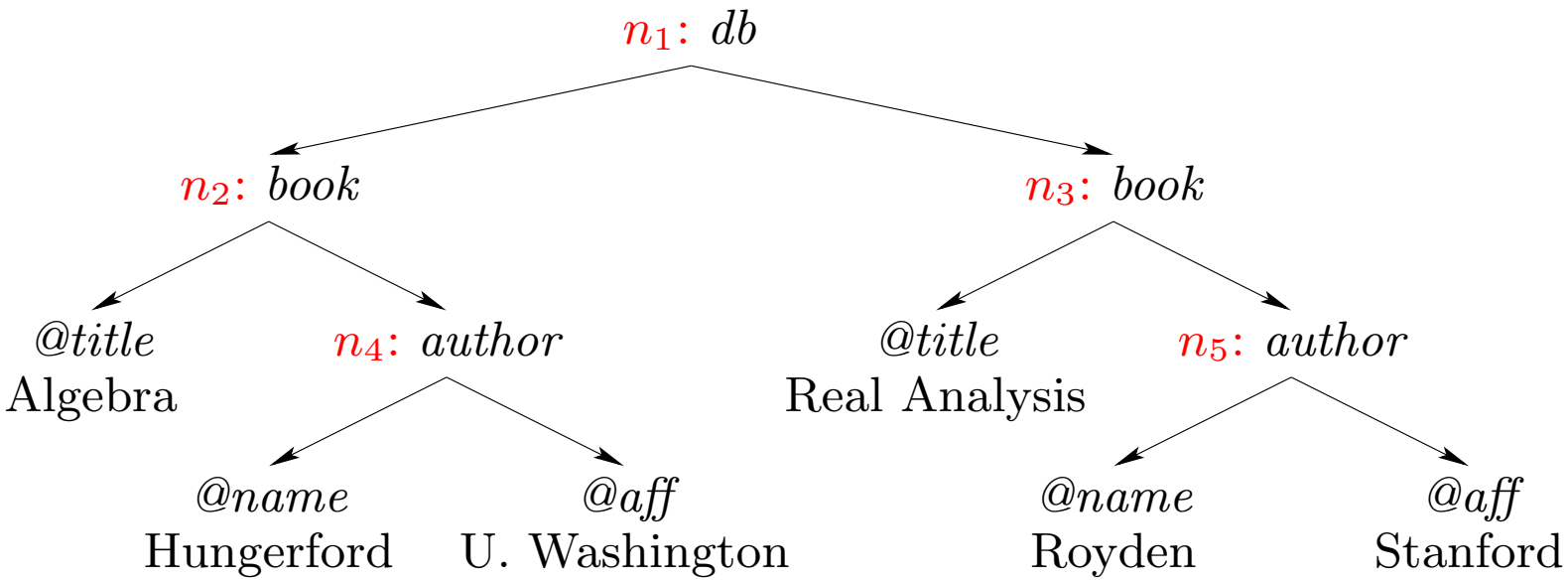
Algoritmo Bottom-up [GKP05]: $\texttt{parent}^*/\texttt{child}^*/?book$

Primer paso: Construir el árbol de parsing de la consulta.



Core XPath: Evaluación bottom-up

Segundo paso: Evaluar las sub-consultas de manera bottom-up.



Core XPath: Evaluación bottom-up

$[\![E_4]\!]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_1	n_2
n_1	n_3
n_1	n_4
n_1	n_5
n_2	n_4
n_3	n_5

Core XPath: Evaluación bottom-up

$[\![E_4]\!]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_1	n_2
n_1	n_3
n_1	n_4
n_1	n_5
n_2	n_4
n_3	n_5

$[\![E_5]\!]_T$	
n_2	n_2
n_3	n_3

Core XPath: Evaluación bottom-up

$[\![E_4]\!]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_1	n_2
n_1	n_3
n_1	n_4
n_1	n_5
n_2	n_4
n_3	n_5

$[\![E_5]\!]_T$	
n_2	n_2
n_3	n_3

$[\![E_3]\!]_T$	
n_2	n_2
n_3	n_3
n_1	n_2
n_1	n_3

Core XPath: Evaluación bottom-up

$[\![E_2]\!]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_2	n_1
n_4	n_2
n_4	n_1
n_3	n_1
n_5	n_3
n_5	n_1

Core XPath: Evaluación bottom-up

$[\![E_2]\!]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_2	n_1
n_4	n_2
n_4	n_1
n_3	n_1
n_5	n_3
n_5	n_1

$[\![E_3]\!]_T$	
n_2	n_2
n_3	n_3
n_1	n_2
n_1	n_3

Core XPath: Evaluación bottom-up

$[\![E_2]\!]_T$	
n_1	n_1
n_2	n_2
n_3	n_3
n_4	n_4
n_5	n_5
n_2	n_1
n_4	n_2
n_4	n_1
n_3	n_1
n_5	n_3
n_5	n_1

$[\![E_3]\!]_T$	
n_2	n_2
n_3	n_3
n_1	n_2
n_1	n_3

$[\![E_1]\!]_T$	
n_1	n_2
n_1	n_3
n_2	n_2
n_2	n_3
n_3	n_2
n_3	n_3
n_4	n_2
n_4	n_3
n_5	n_2
n_5	n_3

Core XPath: Evaluación bottom-up

Teorema [GKP05]: Una Consulta Q en Core XPath puede ser evaluada en tiempo $O(|T| \cdot |Q|)$.

Nota: Dado un nodo n, es posible calcular $\{n' \mid (n, n') \in [\![Q]\!]_T\}$ en tiempo $O(|T| \cdot |Q|)$.

Sabemos que en términos de complejidad Core XPath es un buen lenguaje. ¿Es también bueno en términos de expresividad?

¿Qué tan expresivo es Core XPath?

La respuesta es relativa.

- Es *más*, *tan* o *menos* expresivo que ...

¿Con qué lenguaje podemos comparar?

- Un poco de historia: Bases de datos relacionales.

Bases de datos relacionales: Lógica de primer orden

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.

Bases de datos relacionales: Algebra relacional

Un segundo lenguaje de consulta: Algebra Relacional.

- Combinación de operaciones algebraicas: selección, proyección, join, unión, diferencia, ...

Ventaja: Fácil de implementar y optimizar.

- Una de las razones para el éxito de las bases de datos relacionales.

Pero ¿cuál es la expresividad del álgebra relacional?

Teorema [Cod72]: Algebra relacional = FO.

FO y XML

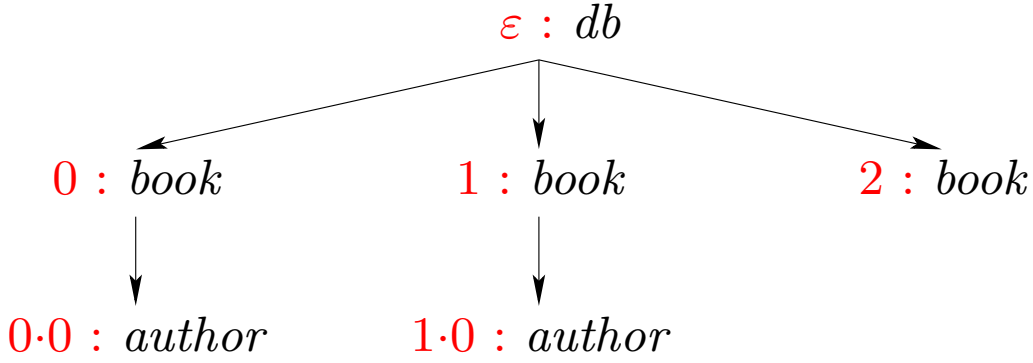
¿Podemos utilizar lógica de primer orden como un lenguaje de consulta para XML?

- ¿Es fácil expresar una consulta?

¿Cuál es la relación entre Core XPath y FO?

FO sobre árboles

Dado árbol T:



Representamos T como una estructura relacional:

$$I_T \quad : \quad \{\texttt{child},\ \texttt{desc},\ \texttt{ns},\ \texttt{sb},\ P_{db},\ P_{book},\ P_{author}\}$$

FO sobre árboles

Donde:

$$
\begin{aligned}
D &= \{\varepsilon, 0, 1, 2, 0{\cdot}0, 1{\cdot}0\} \\
\texttt{child} &= \{(\varepsilon, 0), (\varepsilon, 1), (\varepsilon, 2), (0, 0{\cdot}0), (1, 1{\cdot}0)\} \\
\texttt{desc} &= \{(\varepsilon, \varepsilon), (\varepsilon, 0), (\varepsilon, 1), (\varepsilon, 2), (\varepsilon, 0{\cdot}0), (\varepsilon, 1{\cdot}0), (0, 0), \\
&\quad (0, 0{\cdot}0), (1, 1), (1, 1{\cdot}0), (2, 2), (0{\cdot}0, 0{\cdot}0), (1{\cdot}0, 1{\cdot}0)\} \\
\texttt{ns} &= \{(0, 1), (1, 2)\} \\
\texttt{sb} &= \{(\varepsilon, \varepsilon), (0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 2), \\
&\quad (0{\cdot}0, 0{\cdot}0), (1{\cdot}0, 1{\cdot}0)\} \\
P_{db} &= \{\varepsilon\} \\
P_{book} &= \{0, 1, 2\} \\
P_{author} &= \{0{\cdot}0, 1{\cdot}0\}
\end{aligned}
$$

FO sobre árboles: Ejemplos

Veamos como escribir las consultas que formulamos en Core XPath:

$\texttt{child}/?book$:

$$Q_1(x,y) \quad = \quad \texttt{child}(x,y) \wedge P_{book}(y).$$

$\texttt{parent}/\texttt{child}/?book$:

$$Q_2(x,y) \quad = \quad \exists z\,(\texttt{child}(z,x) \wedge \texttt{child}(z,y) \wedge P_{book}(y)).$$

$\texttt{parent}^*/\texttt{child}^*/?book$:

$$Q_3(x,y) \quad = \quad \exists z\,(\texttt{desc}(z,x) \wedge \texttt{desc}(z,y) \wedge P_{book}(y)).$$

$\texttt{child}/?book/?\langle\texttt{child}/?author\rangle$:

$$Q_4(x,y) \quad = \quad \texttt{child}(x,y) \wedge P_{book}(y) \wedge \exists z\,(\texttt{child}(y,z) \wedge P_{author}(z)).$$

Core XPath y FO

Core XPath $\subseteq$ FO: Para cada consulta Q en Core XPath existe una fórmula $\varphi(x, y)$ en FO tal que

$$(n, n') \in [\![Q]\!]_T \qquad \text{si y sólo si} \qquad I_T \models \varphi(n, n').$$

Ejercicio: Haga la demostración.

Pero: FO $\not\subseteq$ Core XPath.

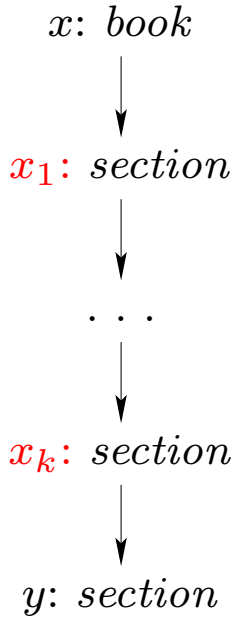
- No podemos expresar algunas consultas naturales.

FO no está contenido en Core XPath

Un ejemplo:

$$Q(x,y) \;=\; P_{book}(x) \wedge \mathtt{desc}(x,y) \wedge P_{section}(y) \;\wedge$$
$$\forall z\,(\mathtt{desc}(x,z) \wedge x \neq z \wedge \mathtt{desc}(z,y) \wedge z \neq y \rightarrow P_{section}(z)).$$

Buscamos:



Core XPath

¿Qué tenemos que agregar a Core XPath para alcanzar la expresividad de FO?

¿Es una buena idea agregar nuevos elementos a Core XPath?

- Si lo que vamos a agregar es *natural*, *útil* e *implementable eficientemente*, entonces conviene agregarlo.

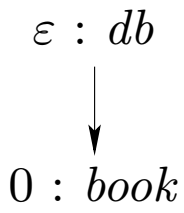
Una pregunta natural: ¿Por qué no usar FO directamente?

Tres desventajas:

- A los usuarios no les gusta trabajar con variables.
- Difícil de optimizar.
- Costoso evaluar una consulta.

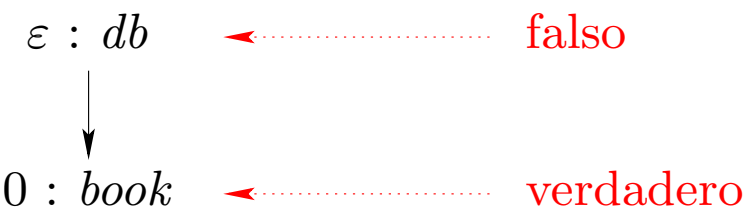
Complejidad de FO sobre árboles

Sea T:



Complejidad de FO sobre árboles

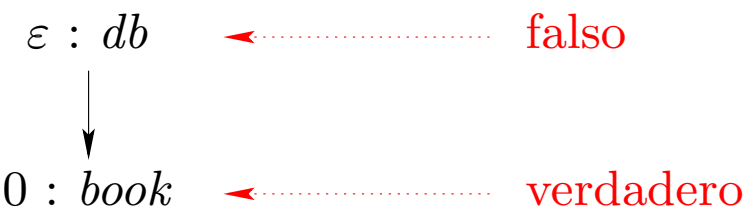
Sea T:



Podemos usar ε como falso y 0 como verdadero, y así reducir desde SAT.

Complejidad de FO sobre árboles

Sea T:



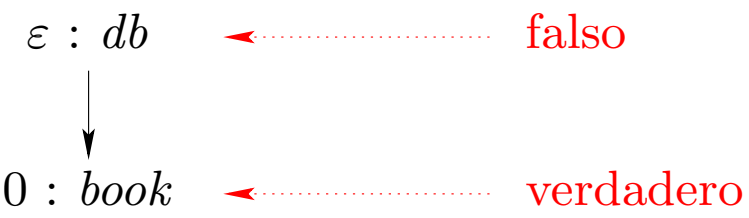
Podemos usar ε como falso y 0 como verdadero, y así reducir desde SAT.

Ejemplo: $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z)$ es representado como

$\exists x \exists y \exists z\, ((P_{book}(x) \vee P_{db}(y) \vee P_{db}(z)) \wedge (P_{db}(x) \vee P_{book}(y) \vee P_{db}(z)))$.

Complejidad de FO sobre árboles

Sea T:



Podemos usar ε como falso y 0 como verdadero, y así reducir desde SAT.

Ejemplo: $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z)$ es representado como

$\exists x \forall y \exists z \left((P_{book}(x) \vee P_{db}(y) \vee P_{db}(z)) \wedge (P_{db}(x) \vee P_{book}(y) \vee P_{db}(z)) \right)$.

Conclusión: FO sobre árboles es NP-hard.

Conditional XPath

Buena noticia: Hay que agregar poco a Core XPath para obtener FO y la complejidad no cambia.

Conditional XPath:

- Caminos básicos:

  $\texttt{paso} \quad ::= \quad \texttt{child} \mid \texttt{parent} \mid \texttt{right} \mid \texttt{left}$

- Expresiones para caminos:

  $\texttt{camino} \quad ::= \quad \texttt{paso} \mid (\texttt{paso}/?\texttt{test})^* \mid \texttt{camino}/\texttt{camino} \mid$
  $\texttt{camino} \cup \texttt{camino} \mid ?\texttt{test}$

- Filtros:

  $\texttt{test} \quad ::= \quad \mathit{nombre} \mid \langle\texttt{camino}\rangle \mid \neg\texttt{test} \mid \texttt{test} \wedge \texttt{test}$

Conditional XPath

Dado un árbol $T = (D, \lambda)$:

$$[\![(\texttt{paso}/?\texttt{test})^*]\!]_T \quad = \quad \{(n,n) \mid n \in D\} \cup [\![\texttt{paso}/?\texttt{test}]\!]_T \cup [\![\texttt{paso}/?\texttt{test}/\texttt{paso}/?\texttt{test}]\!]_T \cup \cdots$$

Nótese que Core XPath $\subseteq$ Conditional XPath:

$\texttt{paso}^*$ es equivalente a $(\texttt{paso}/?\neg(a \wedge \neg a))^*$.

Conditional XPath y FO

Teorema [Mar04a, Mar05]: Sobre árboles se tiene que Conditional XPath = FO.

Teorema [Mar04b]: Una Consulta Q en Conditional XPath puede ser evaluada en tiempo $O(|T| \cdot |Q|)$.

¡Conditional XPath es un buen lenguaje!

Otra pregunta natural

Si la complejidad de Conditional XPath es $O(|T| \cdot |Q|)$ y de FO es NP-hard, ¿Cómo puede ser que Conditional XPath = FO?

Del teorema anterior se puede concluir que no hay una traducción eficiente de FO a Conditional XPath.

Del teorema anterior no se puede inferir qué lenguaje es mejor para el usuario.

¿Necesitamos más lenguajes de consulta?

Conditional XPath es un buen lenguaje, pero ¿es suficiente?

Para navegar XML es necesario utilizar expresiones regulares arbitrarias.

- La mayoría de los lenguajes propuestos las incluyen.

Veamos un lenguaje con expresiones regulares.

Regular XPath: Sintaxis

Caminos básicos:

$$\texttt{paso} \quad ::= \quad \texttt{child} \mid \texttt{parent} \mid \texttt{right} \mid \texttt{left}$$

Expresiones para caminos:

$$\begin{aligned} \texttt{camino} \quad ::= \quad & \texttt{paso} \mid \texttt{camino}^* \mid \texttt{camino/camino} \mid \\ & \texttt{camino} \cup \texttt{camino} \mid ?\texttt{test} \end{aligned}$$

Filtros:

$$\texttt{test} \quad ::= \quad \textit{nombre} \mid \langle \texttt{camino} \rangle \mid \neg\texttt{test} \mid \texttt{test} \wedge \texttt{test}$$

Regular XPath: Semántica

Dado un árbol $T = (D, \lambda)$:

$$[\![\texttt{camino}^*]\!]_T = \{(n,n) \mid n \in D\} \cup [\![\texttt{camino}]\!]_T \cup [\![\texttt{camino/camino}]\!]_T \cup [\![\texttt{camino/camino/camino}]\!]_T \cup \cdots$$

Ejemplo: $(\texttt{child}/?\mathit{section}/\texttt{child}/?\mathit{section})^*$

La expresividad de Regular XPath

Nótese que Conditional XPath $\subseteq$ Regular XPath:

$(\texttt{paso}/?\texttt{test})^*$ es una fórmula en Regular XPath.

¿Qué tan expresivo es Regular XPath?

- ¿Es Conditional XPath = Regular XPath?
- ¿Con qué otro lenguaje lo podemos comparar?

Otro poco de historia: **Autómata y lógica.**