Generating Relations from XML Documents

Sara Cohen

Yaron Kanza

Yehoshua Sagiv

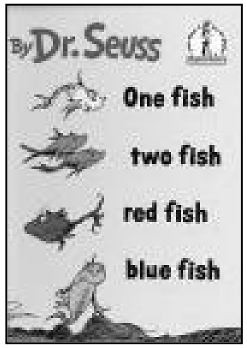
The Hebrew University of Jerusalem

The Problem

A Typical Web Page

Buy our Classic Children's books.

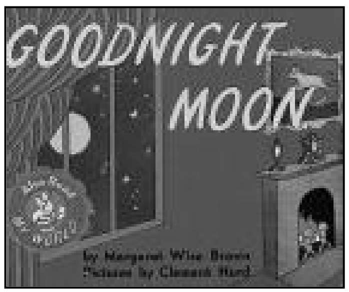
Now at a special discount!



One Fish Two Fish by Dr. Seuss

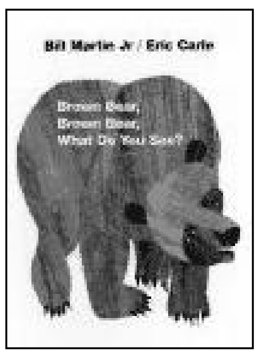
Our Discount: 10%

Costs Only: $7.95



Goodnight Moon by Margaret Brown

Costs Only: $10.55



Brown Bear by Bill Martin Jr.

Our Discount: 15%

Costs Only: $6.00

amazing.com

An Inside Look

<bookinfo>

<book><title>One Fish Two Fish</title>

<aname>Dr. Seuss</aname>

<discount>10</discount>

<price>7.95</price></book>

<book><title>Goodnight Moon</title>

<aname>Margaret Brown</aname>

<price>10.55</price></book> ....

</bookinfo>

A Query

- Find titles and discounts of books by Dr. Seuss that cost less than $10.

How??

Attempt 1: Search Engine

This won't work!

- No "<" operator defined
- Can't specify that Dr. Seuss is the author and $10 is the price
- Can't specify that the price belongs to the book
- Can't specify desired output (i.e., titles, discounts)

Attempt 2: XQuery

```
FOR $b IN document("bixml")//book
WHERE $b/price<10 AND $b/name='Dr. Seuss'
RETURN
  <result>
    <title> $b/title </title>
    <discount> $b/discount </discount>
  </result>
```

This will work, but:

- Difficult for naive users
- Requires knowledge of document structure
- Dependent on document structure

Attempt 3: Select-Project

Here is what we would like:

```
SELECT title, discount
  FROM  "bib.xml"
WHERE aname = 'Dr. Seuss' and
          price < 10
```

This is possible if the relation

Book(title, aname, price, discount)

can be generated from the document

Our Goal – Extract Relations from XML

- Simple language to define **relation generators**
- Relation generators should work correctly even if **the structure of the document changes**
- Missing information should be handled gracefully, i.e., create relations with **null values**

Syntax of Relation Generators

The Elements of the Syntax

- Essentially, a relation generator is a list of tags, e.g., **Book(title, aname, price, discount)**
- More generally, we can use **XPath expressions**, instead of tags
- Any fragment of XPath can be used, provided that there is a PTime test for checking whether a given node satisfies a given path expression
- We may also want to specify that some of the tags should not get null values

The Formal Syntax

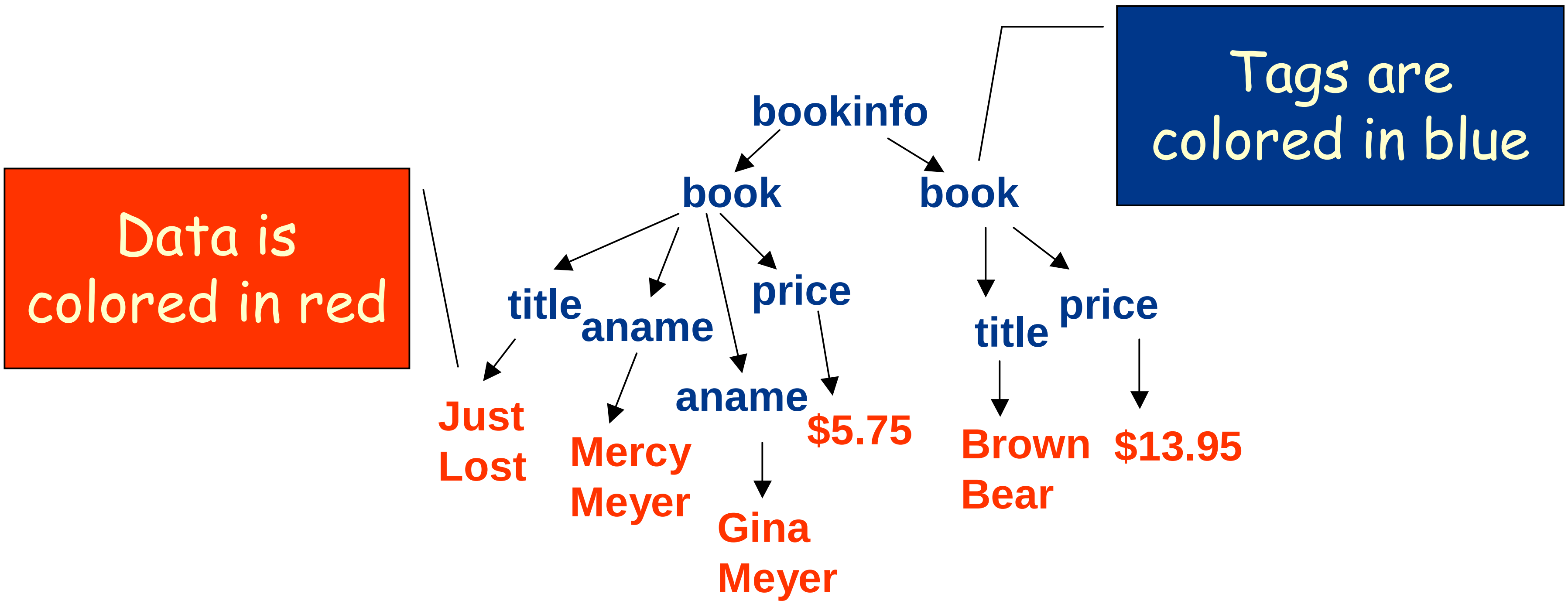
- Relation generators are built up from **XPath expressions**, denoted by p, p_1, p_2, etc.
- A **relation generator** is a pair $\Delta=(P,k)$, where
  - P is an m-tuple of XPath expressions
  - $k \leq m$
    - k means that the first k tags should not get null values

The Semantics (Intuitively)

- The **result** of applying a relation generator $((p_1,\ldots,p_m), k)$ to a document is a set of m-tuples $(n_1,\ldots,n_m)$ of **nodes and null values**, such that
  - n_i **satisfies** p_i if n_i is not the nul lvalue, $i \leq m$
  - n_i is **not the null value**, for $i \leq k$
  - the nodes in $(n_1,\ldots,n_m)$ are **meaningfully related**

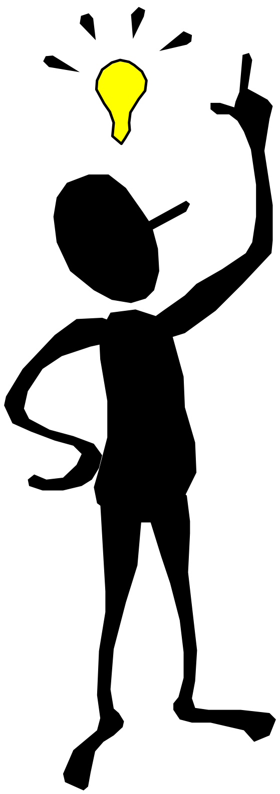
Semantics: The Intuition

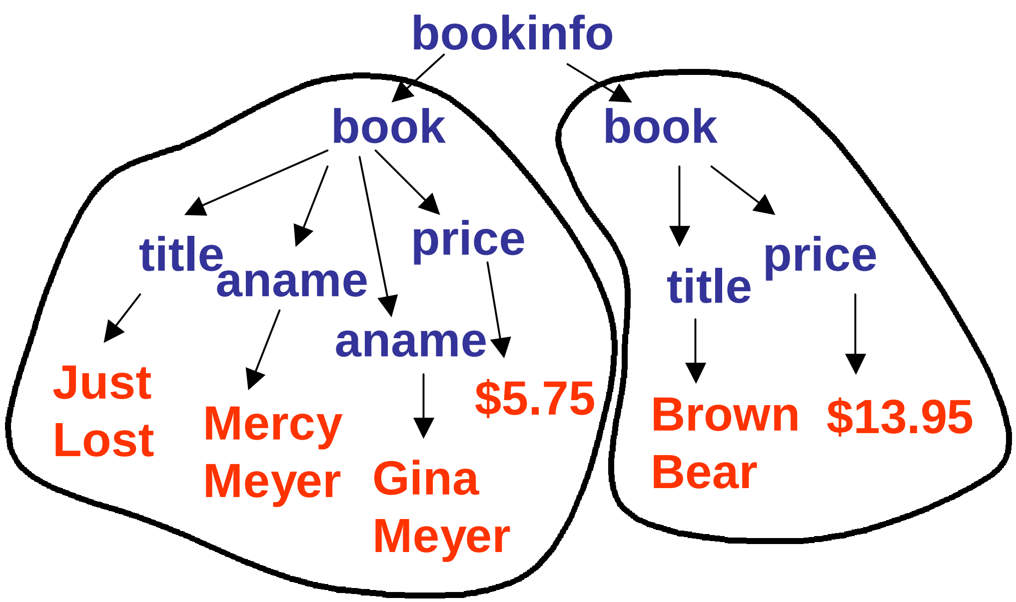
An Example Document, as a Tree



Document with information about two books

Applying a Relation Generator with Human Intervention

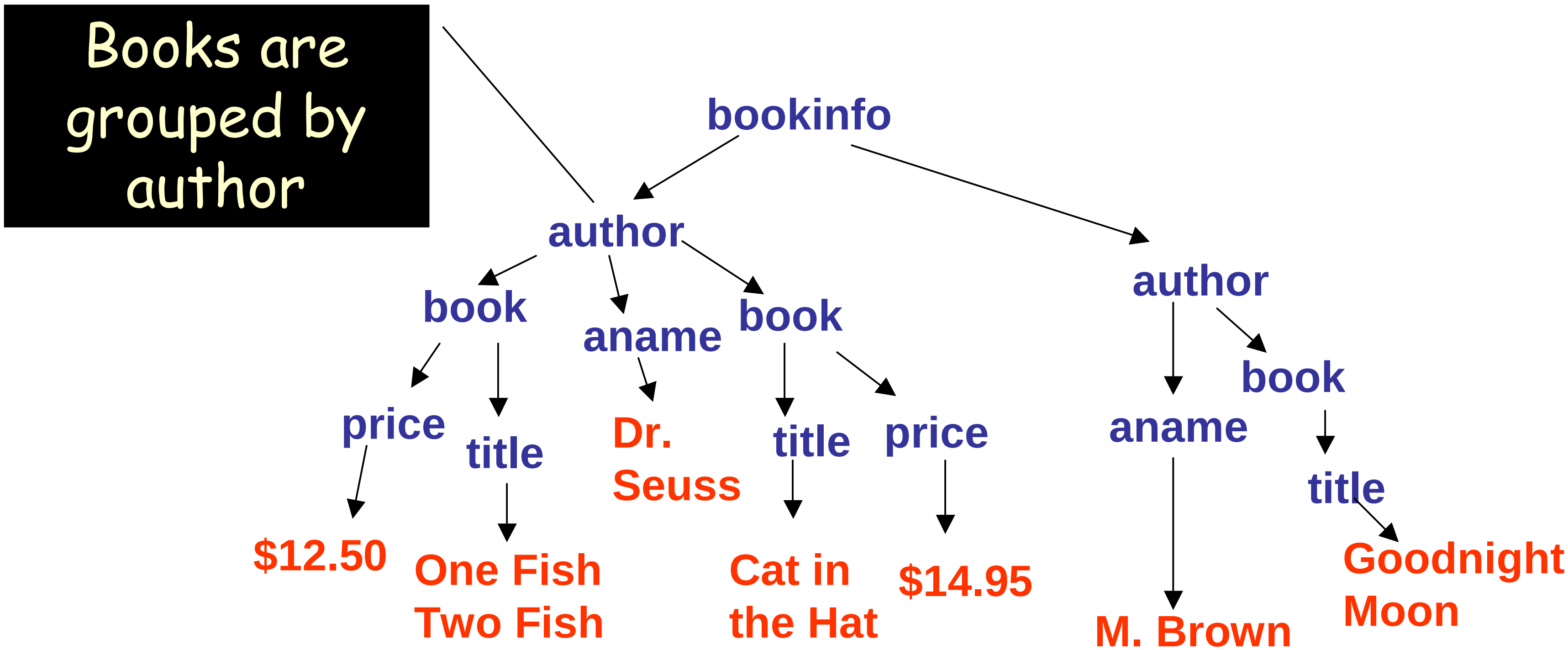




Result:

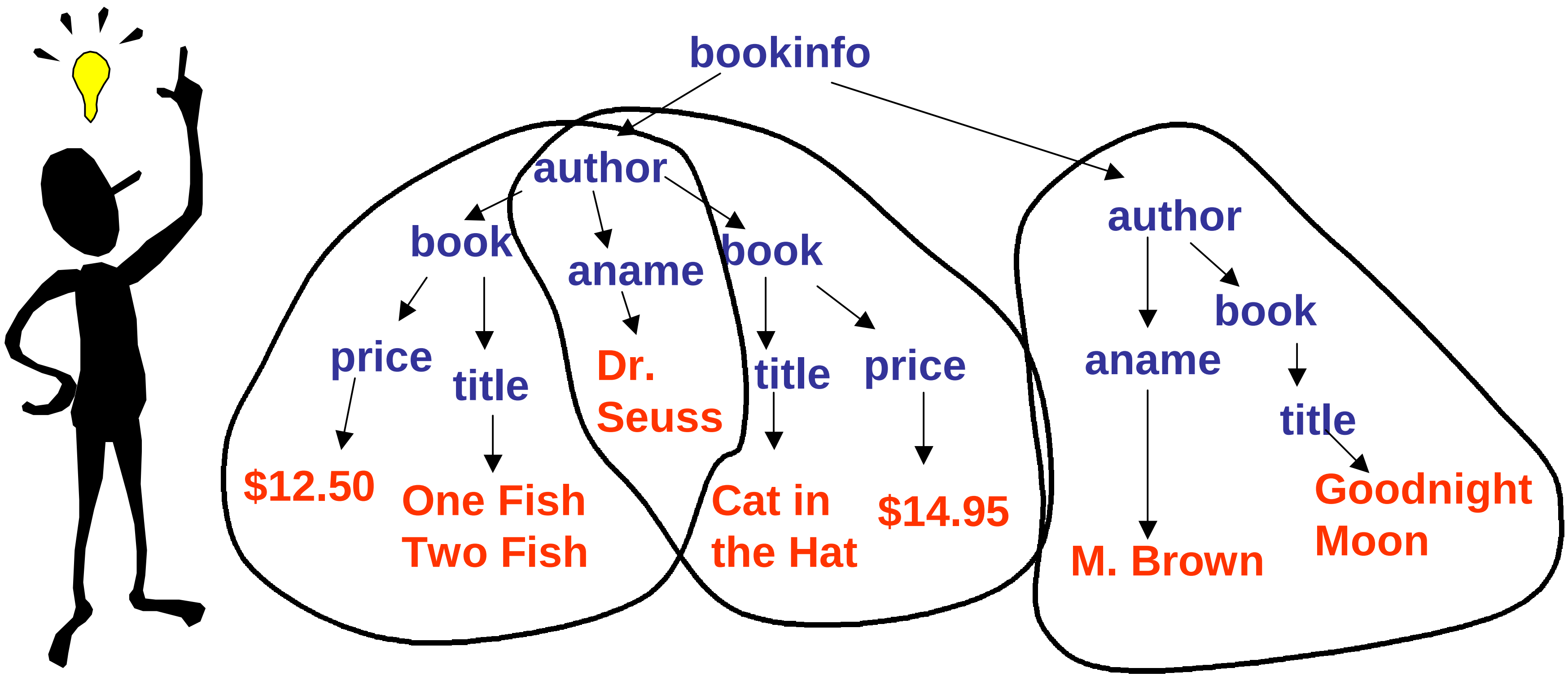
- Just Lost, $5.75
- Brown Bear, $13.95

A Different Document



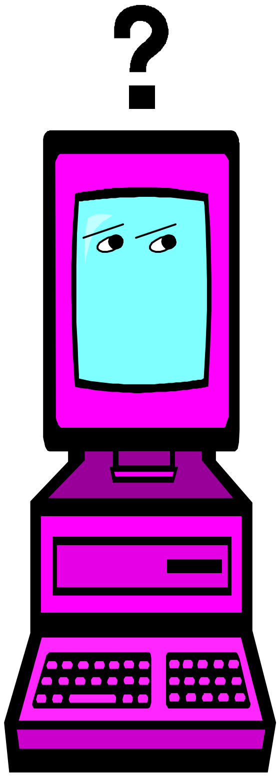
Similar document, but the hierarchical structure is different from that of the previous document

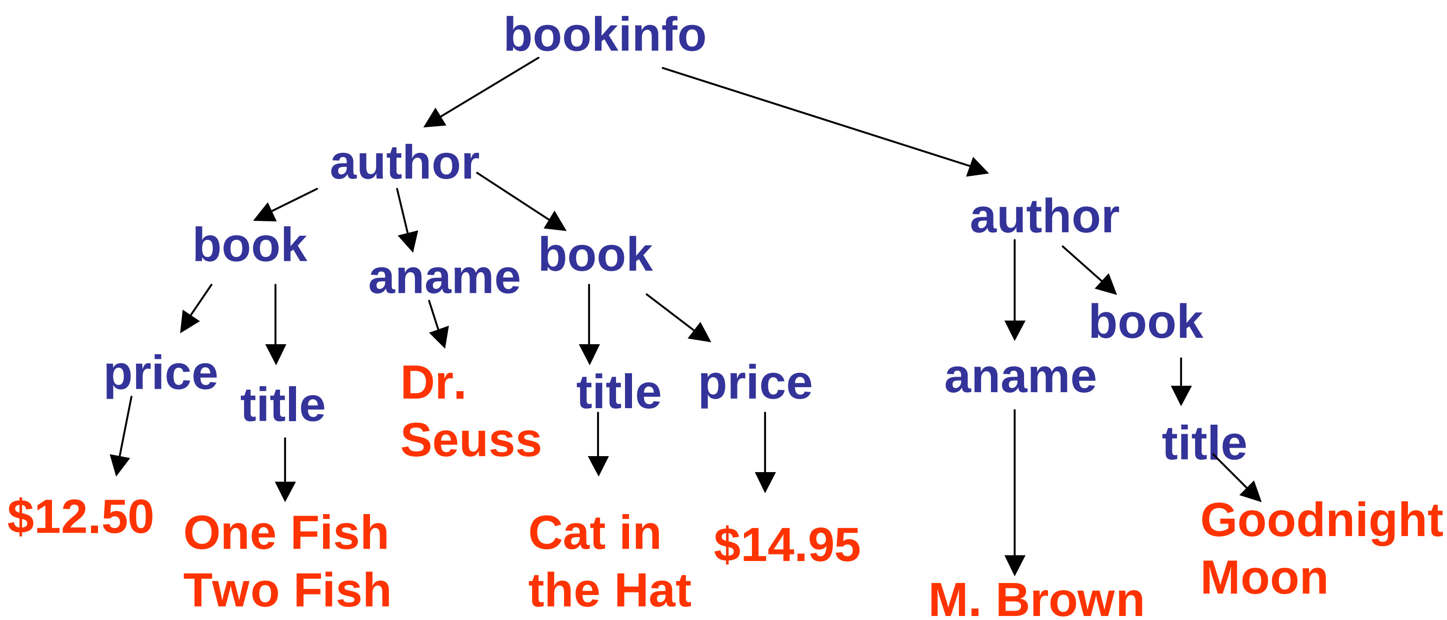
Applying a Relation Generator with Human Intervention (2)



We find the tuples, even with this hierarchy

Answering a Query *without* Human Intervention





We need to find pairs of related title and price nodes. How??

Formal Semantics

Finding Related Nodes

- The **relationship tree** of n_1 and n_2 is the subtree T of the document D, such that
  - T is rooted at the **lowest common ancestor** (lca) of n_1 and n_2, and
  - T consists of the two paths from the lca to n_1 and n_2
- We say that n_1 and n_2 are **interconnected** if the relationship tree of n_1 and n_2 either
  - does not contain two nodes with the same label, or
  - the only two distinct nodes with the same label are n_1 and n_2

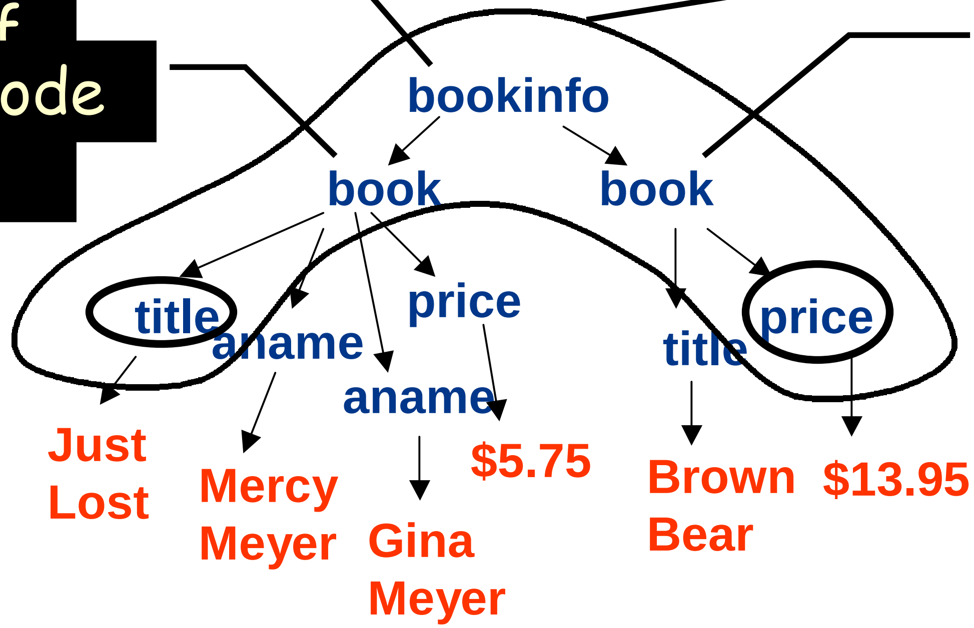
Example 1

The lowest comm on ancestor of

A book node nodes

A book node

relationship tree of the circled nodes

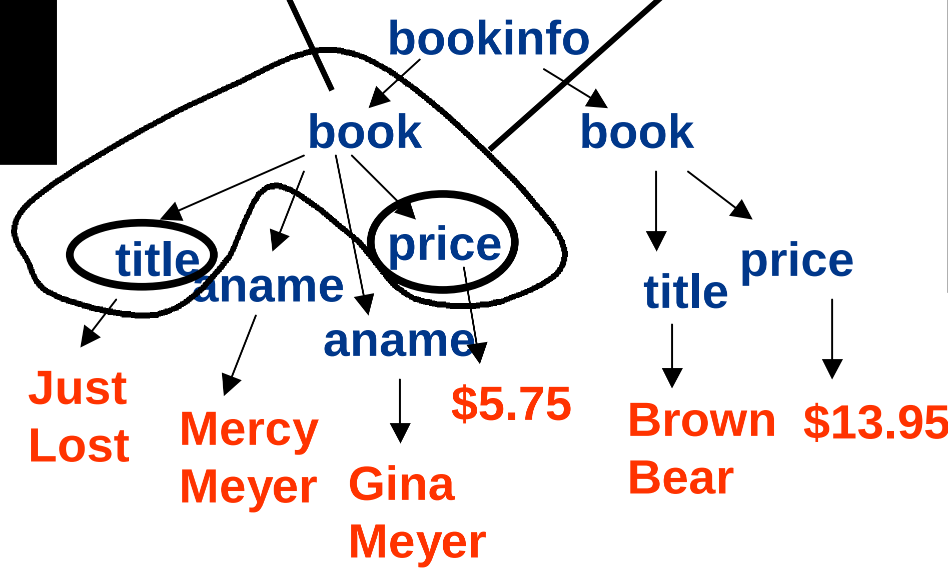


Intuition: The nodes belong to ***different*** book entities

Example 2

The lowest comm on ancestor of the circled nodes

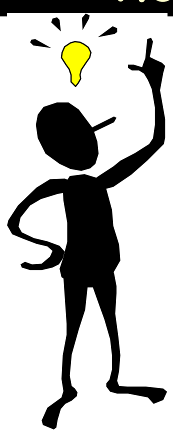
The relationship tree of the circled nodes

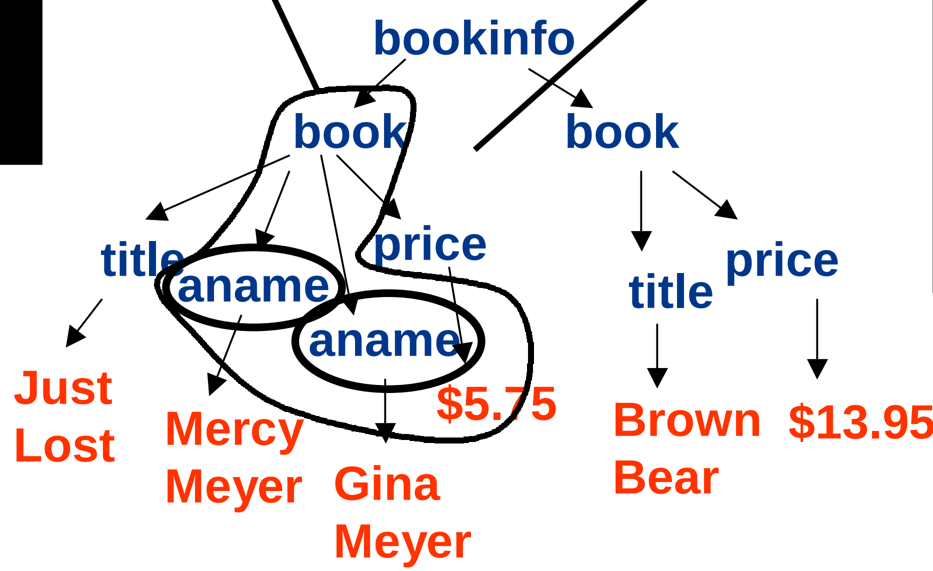


Intuition: The nodes belong to ***the same*** book entity

Example 3

The lowest common ancestor of the circled nodes





The relationship tree of the circled nodes

Intuition: Although the two nodes represent different author names, they are meaningfully related by virtue of belonging to ***the same*** book entity

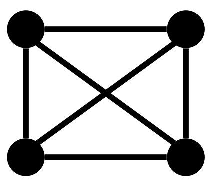
Interconnection Graphs

- The **interconnection graph** of a document *T*, denoted ***IG(T)***, consists of
  - the same nodes as in *T*, and
  - an edge between each pair of interconnected nodes
- We use ***IG(T,N)*** to denote the **induced subgraph** of *IG(T)* on the set of nodes *N*

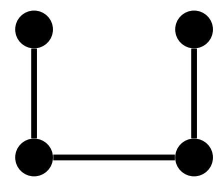
Graph Properties

- We wil lbe interested in 3 different types of graphs:

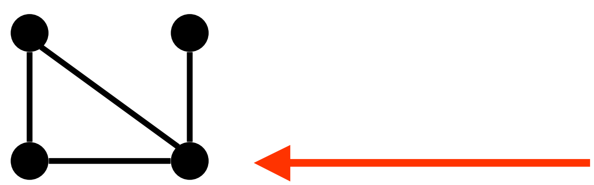
Complete Graphs



Connected Graphs



Star Graphs



center

Matchings

- $p_1, \ldots, p_m$ are path expressions
- S is the set of nodes in the document tree
- A function

$$\mu: \{ p_1, \ldots, p_m \} \rightarrow S \cup \{\text{null}\}$$

is a **matching** if for all i,

- $\mu(p_i)$ satisfies p_i *or*
- $\mu(p_i)$ = null

Types of Matchings

- Let μ be a matching
- Let N be the the set of nodes in the image of μ
- μ is a **complete matching** if $IG(T,N)$ is a complete graph
- μ is a **reachable matching** if $IG(T,N)$ is a connected graph
- μ is a **star matching** if $IG(T,N)$ is a star graph

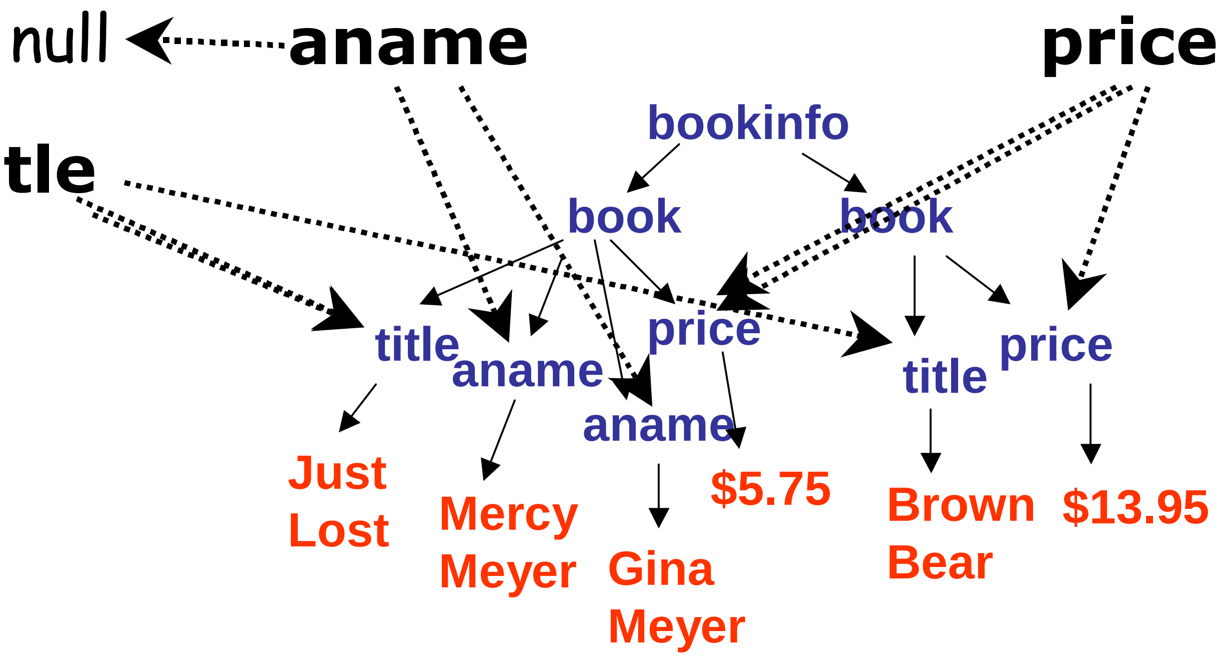
Maximal Matchings

- A matching μ **subsumes** $\mu`$ if $\mu$ and $\mu`$ are equal on all non-null images of $\mu`$, i.e., for all *p*, either
  - $\mu(p) = \mu`(p)$ or
  - $\mu(p)$ = nul l
- A matching is **maximal** if it is maximal with respect to subsumption

Evaluating Relation Generators

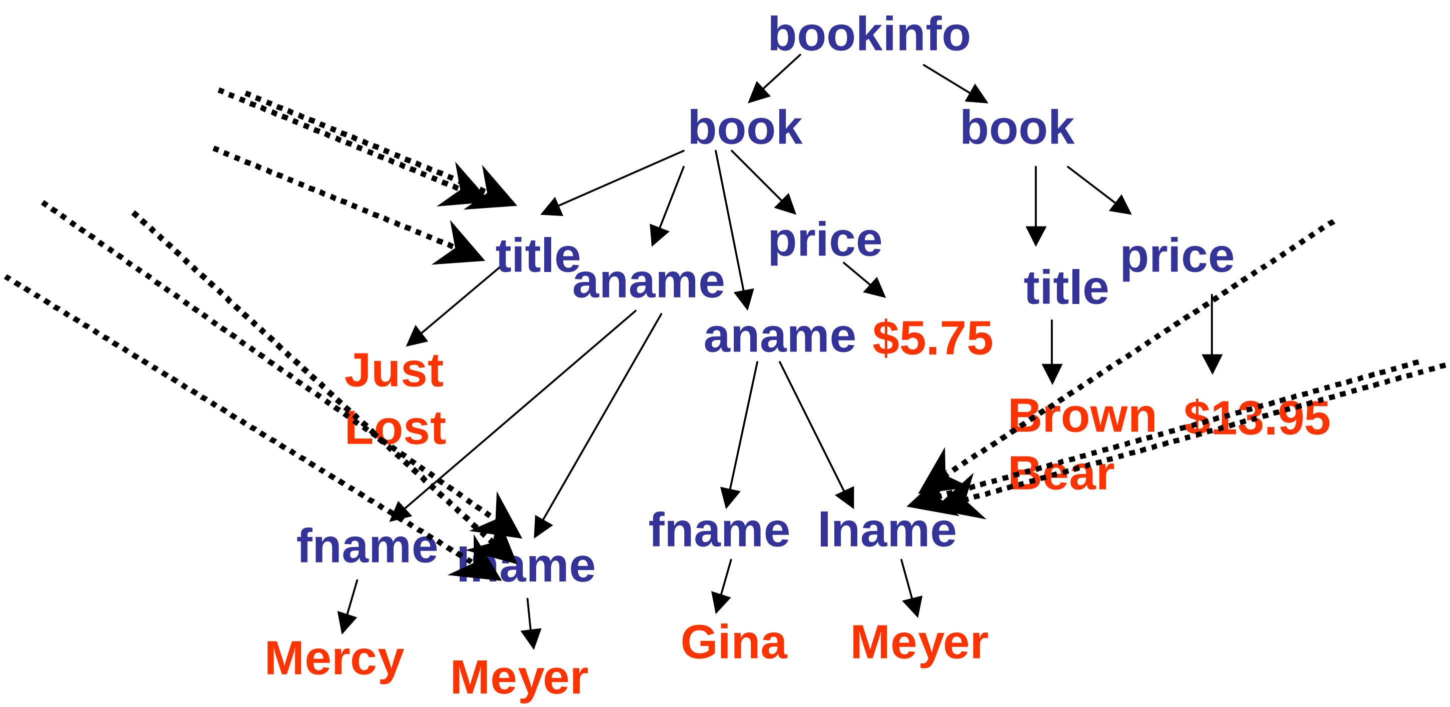
- The result of applying $((p_1,\ldots,p_m), k)$ to T under **complete semantics** is the set of images of all maximal complete matchings
- We define similarly the result under **reachable semantics** and under **star semantics**

Example Evaluation (1)



The result remains the same under either the reachable semantics or the star semantics

Example Evaluation (2)



An additional matching is derived under the reachable semantics and the star semantics

Complexity of Evaluation

Complexity Measure

- The time complexity of evaluating a relation generator is measured in terms of **the size of the input and the output**
- Subsumed matchings should be removed as soon as possible or not be created at all

Star Semantics

- **Theorem:** For the **star semantics,** the result of applying a relation generator $((p_1,\ldots,p_m), k)$ to a tree T can be computed in **polynomial time** in the size of the input and the output

Complete Semantics and Reachable Semantics: The General Case

- **Theorem:** For either **the complete semantics** **or** **the reachable semantics**, it is NP-Complete to check non-emptiness of the result of applying a relation generator $((p_1,\dots,p_m), k)$ to a tree T

Complete and Reachable Semantics: All Path Expressions May Have Nulls

- **Theorem:** For either the **complete semantics** or the **reachable semantics**, the result of applying a relation generator $((p_1,\ldots,p_m), O)$ to a tree T can be computed in **polynomial time** in the size of the input and the output

Complete Semantics: Another Special Case

- **Theorem:** For the **complete semantics**, the result of applying $((p_1,\ldots,p_m), k)$ to T can be computed in **polynomial time** in the size of the input and the output, provided that
  - no path from the root of T to a leaf has repeated tags,
  - $p_1,\ldots,p_m$ are **acyclic**, and
  - for all p_i and p_j $(i \neq j)$, there is no pair of nodes n_1 and n_2 with the same tag, such that n_1 and n_2 satisfy p_i and p_j, respectively

Acyclic Path Expressions

- For a given path expression p and a tree T, the relation scheme $\boldsymbol{R_{p,T}}$ consists of al l **tags** of nodes $\boldsymbol{n}$, such that n either matches p or has a descendent that matches p
- $p_1,\ldots,p_m$ are **acyclic** if the hypergraph of $\boldsymbol{R_{p1,T}},\ldots,\boldsymbol{R_{pm,T}}$ is α-acyclic