

Representing and Querying XML with Incomplete Information

Serge Abiteboul

INRIA

Joint work with Victor Vianu, UCSD and Luc Segoufin, INRIA

Organization

- Incomplete databases
- XML
- Motivations
- Setting: documents, types, queries
- Example
- What can be done: PTIME
- Limits: exponential blow-up

Incomplete databases

Incomplete databases

- Moshe Vardi from Rice is presenting a paper, but I don't know the title
 - Is Moshe from U. Wisconsin? *No*
 - Is Moshe presenting a paper? *Yes*
 - Is Moshe speaking *on the usual effectiveness of logic in computer science?* *Maybe*
- My knowledge of the world is incomplete

Incomplete databases

- Moshe is from Rice and he is presenting a paper
 - Is Moshe from U. Wisconsin? *No*
 - Is Moshe presenting a paper? *Yes*
 - Is Moshe speaking *on the unusual effectiveness of logic in computer science?* *Maybe*
- My knowledge of the world is incomplete
- Natural for logicians – an early discovery for database people

Incomplete databases

- An old story
- Main idea: database is the set of all possible worlds – incompleteness comes from the fact that we do not know which one it is
- Sure answers: true in all worlds
- Possible answers: possible in some worlds

Database people look at simple models

| A | B | C | D |
|---|-----|---|---|
| 4 | 5 | @ | 6 |
| 7 | \$2 | 5 | 8 |
| 7 | \$2 | 6 | @ |

Incomplete databases

- From the very first papers of Codd
- Tables = simple descriptions of the possible words
 - Landmark paper: Imielinski and Lipski – notion of representation system
- Logical approach
 - knowledge of the world = a set of logical sentences
 - Landmark paper: Vardi's *Querying Logical Databases* and the complexity of nulls
- Results on *meaning* of answers and complexity

Representation system

- A system to represent the information we have – **T**
- Need a representation of (possible) answers – **q(T)**

$$\begin{array}{ccc} \text{rep}(\mathbf{T}) & \xrightarrow{\mathbf{q}} & \mathbf{q}(\text{rep}(\mathbf{T})) = \text{rep}(\mathbf{q}(\mathbf{T})) \\ \uparrow \text{rep} & & \uparrow \text{rep} \\ \mathbf{T} & \xrightarrow{\mathbf{q}} & \mathbf{q}(\mathbf{T}) \end{array}$$

- This is a (weak) **representation system**
- You may think of it as any limited logic with the property of the diagram

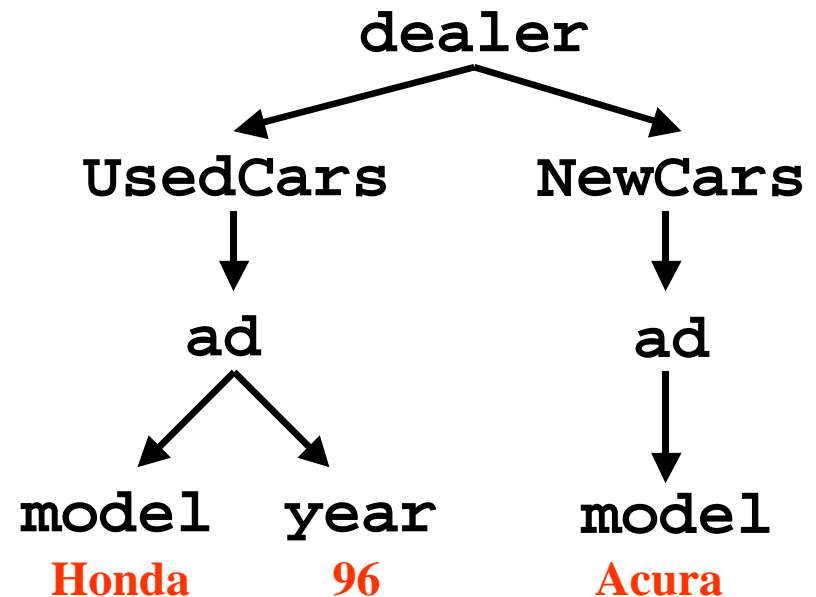
XML

XML

- New fashion: standard for the web to replace HTML
- Data exchange model: semistructured data
- Bottom line: trees with tagged vertices
- Query languages: mix of information retrieval and relational query languages

Review of XML and DTDs

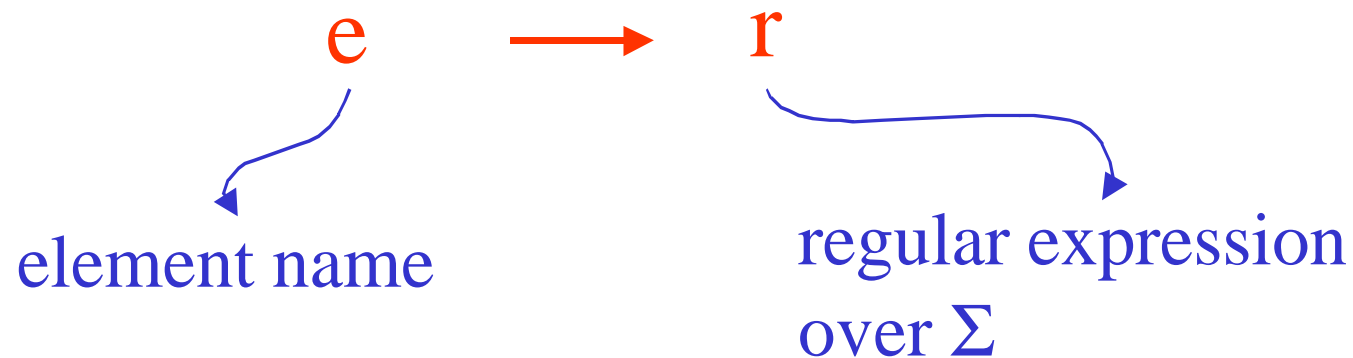
```
<dealer>
  <UsedCars>
    <ad>
      <model>Honda</model>
      <year>96</year>
    </ad>
  </UsedCars>
  <NewCars>
    <ad>
      <model>Acura</model>
    </ad>
  </NewCars>
</dealer>
```



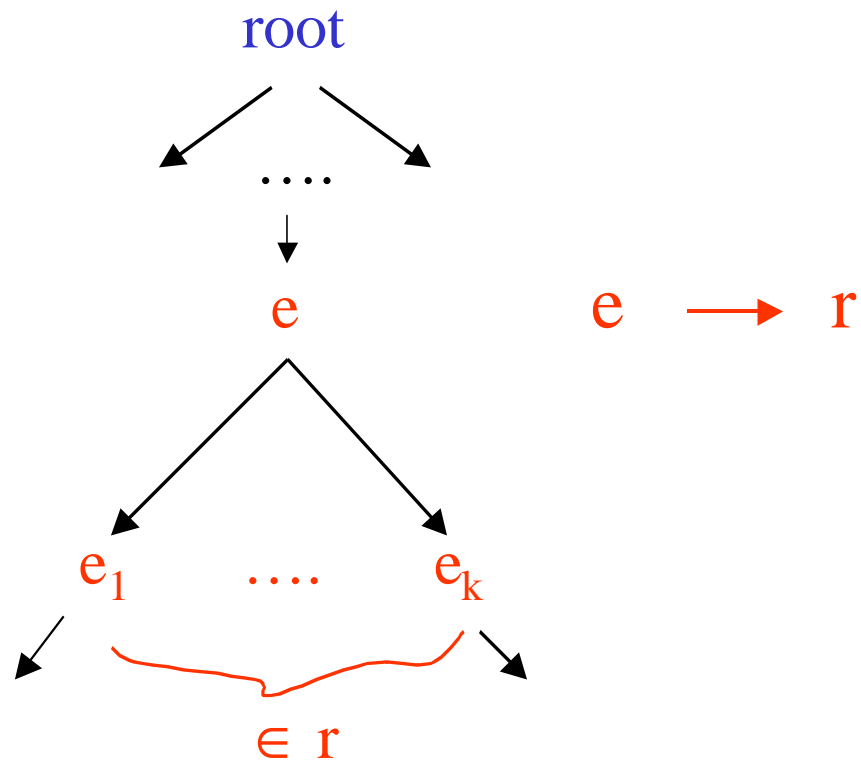
Typing XML: Data Type Definition

Σ : alphabet of element names, $\text{root} \in \Sigma$

set of rules:



Documents satisfying a DTD



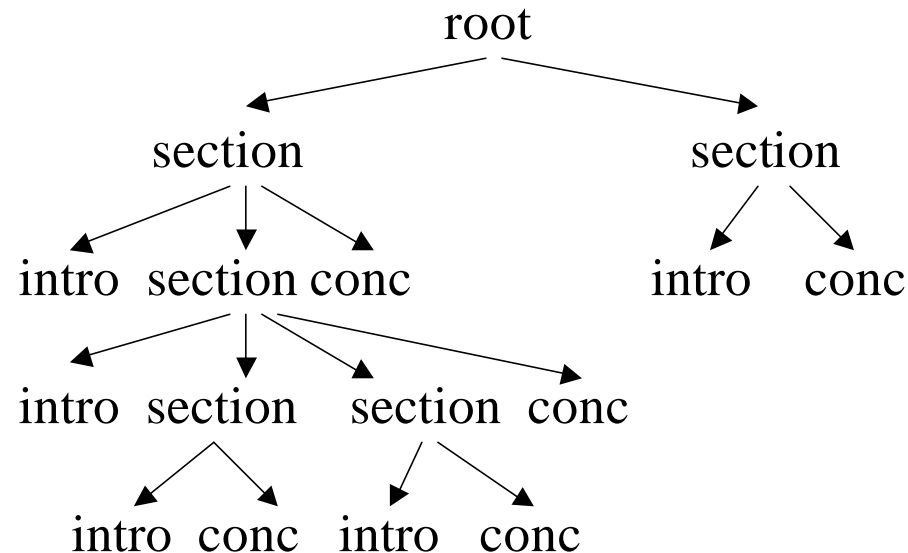
Set of trees satisfying
DTD d : **Tree(d)**

Example

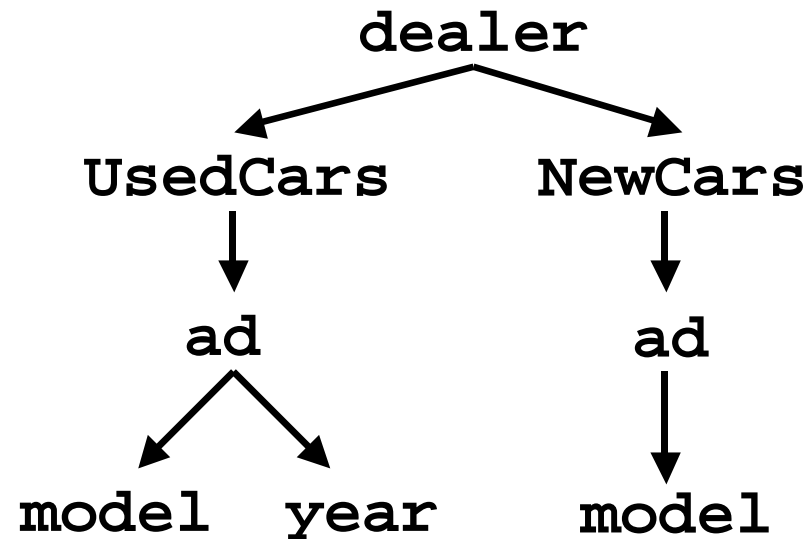
A DTD and a tree satisfying it:



`root` → `section*`;
`section` → `intro, section*, conclusions;`



Shortcoming of DTDs: context-free definition



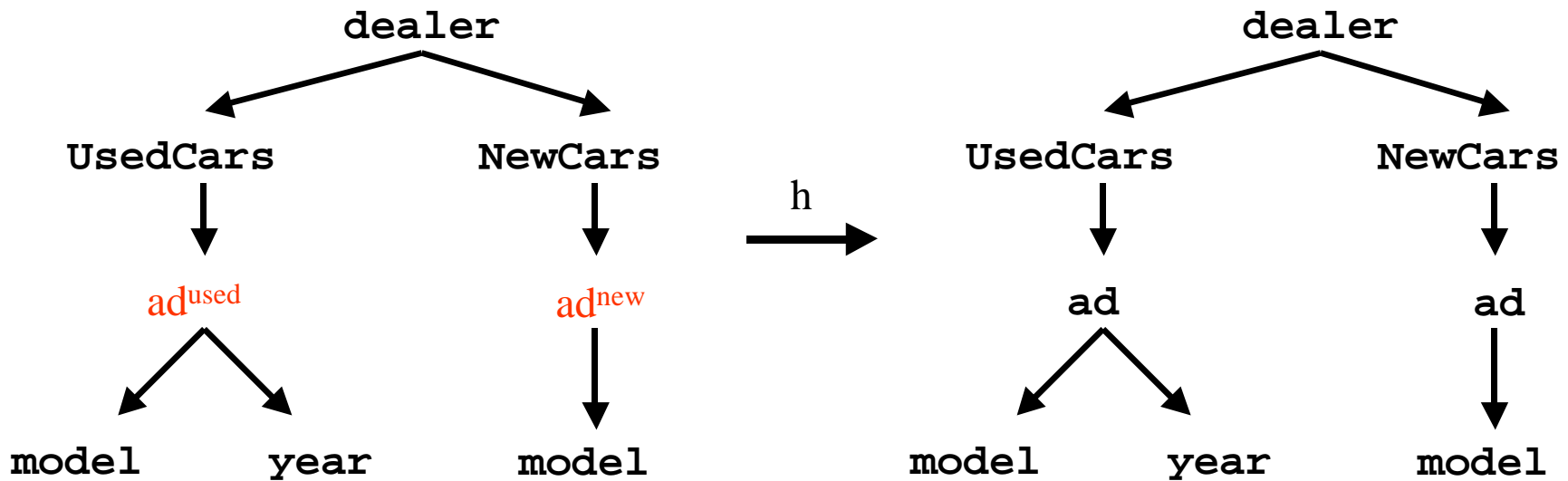
Cannot write a DTD describing this single document:
ad has different structure in different contexts

Solution: specialization (decoupled tags)

specialize **ad**: ad^{used} and ad^{new}

Dealer \longrightarrow UsedCars NewCars
UsedCars \longrightarrow ad^{used}
NewCars \longrightarrow ad^{new}
 ad^{used} \longrightarrow model year
 ad^{new} \longrightarrow model

$h(ad^{new}) = ad$
 $h(ad^{used}) = ad$



- What sets of trees can specialized DTDs define?

Exactly the **regular tree languages**
of unranked trees!

[Bruggemann-Klein+Murata+Wood]

- **Consequences:**

same **closure properties** (e.g. intersection, union, complement) and complexities of manipulations

Motivations

Massive repository of XML documents

- Motivated by the web and the **Xyleme** project at INRIA
- Objectives:
 - store massive volumes of XML documents
 - provide queries and other services (monitoring, integration, classification, web crawling...)
 - emphasis on change management

Why incomplete information?

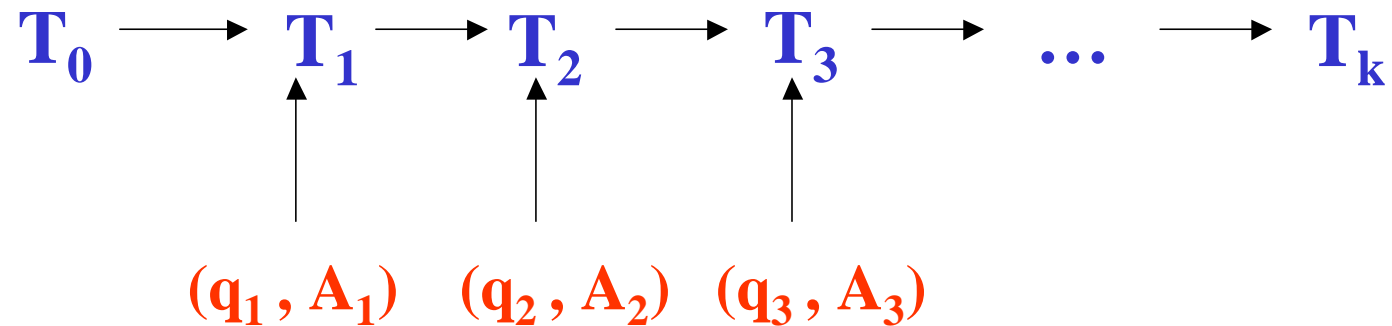
- Information in such a warehouse is seldom complete:
 - limited storage capacity
 - data change
 - expired data
 - unavailable data (server down, not proper access rights),
etc
- This work: simple, practically appealing approach to managing incomplete information in this context

Incomplete Information Scenario:

- Information **continuously enriched** by successive queries to XML sources
- Need to:
 - represent incomplete information
 - intelligently answer queries
 - using incomplete information

Wish list:

- Need some form of representation system
- Efficient incremental maintenance through consecutive queries



$$\text{rep}(T_k) = \text{rep}(T_0) \zeta q_1^{-1}(A_1) \zeta \dots \zeta q_k^{-1}(A_k)$$

Useful also for queries

- Compute a representation of the answer

$$\begin{array}{ccc} \mathbf{rep(T)} & \xrightarrow{\mathbf{q}} & \mathbf{q(rep(T)) = rep(q(T))} \\ \uparrow \text{rep} & & \uparrow \text{rep} \\ \mathbf{T} & \xrightarrow{\mathbf{q}} & \mathbf{q(T)} \end{array}$$

Need: strong representation system
wrt query language

Answer queries – use what you have

- Decide if available info is enough to fully answer the query
 - similar to answering queries using views
- Answering queries using views: heavily studied problem
 - $V_1/Q_1, V_2/Q_2, \dots, V_n/Q_n$
 - Comes a query Q
 - Can it be answered fully, partial answer...

Answer queries – Ask for more

- If it is not sufficient, seek additional information
 - **mediator** problem: find “minimal” set of additional queries to sources needed to fully answer query
 - use representation of incomplete info to **guide mediator**

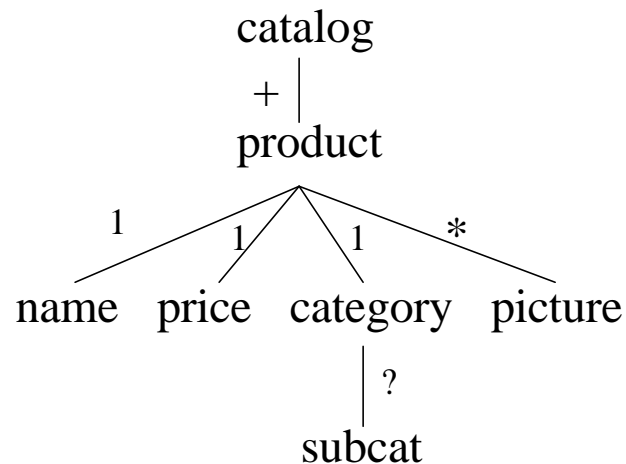
Challenge: balance expressiveness and tractability!

- Choice of: XML document types (DTDs)
Query language
- This work: one proposal
 - simple, practically appealing, many limitations
- Justification:
 - extra features lead to serious problems!

Setting

Documents and Types

- XML abstraction: unranked trees with labels and values
- simplified DTD: unordered, simple cardinality constraints

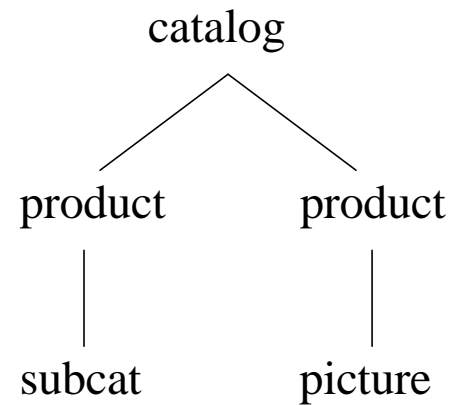
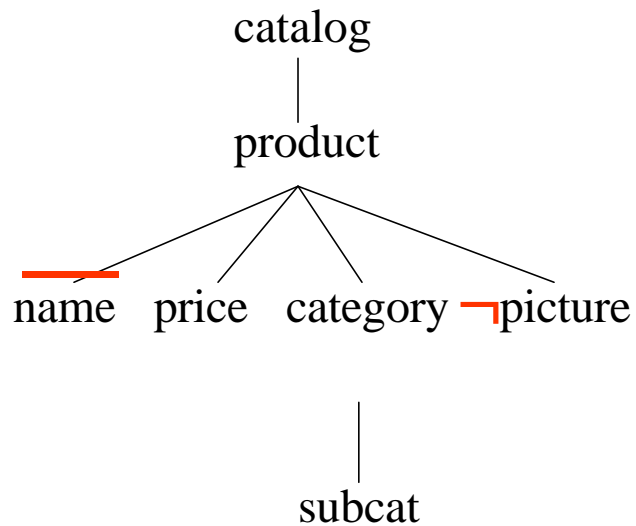


1: exactly one
+: at least one
*: unrestricted
?: zero or one

Query Language

- **prefix-selection queries** (ps-queries)

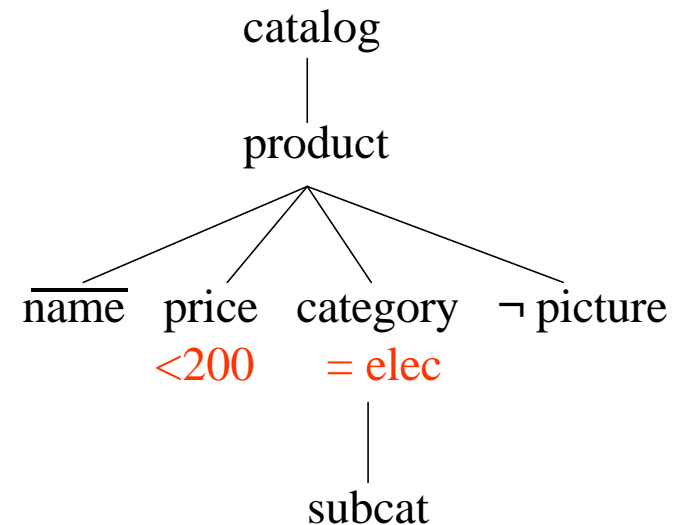
No branching: siblings with same label



Prefix-selection Queries

Find electronics products with
price < 200 and without
pictures

- display all info about their name,
the price, the category and
subcategory)



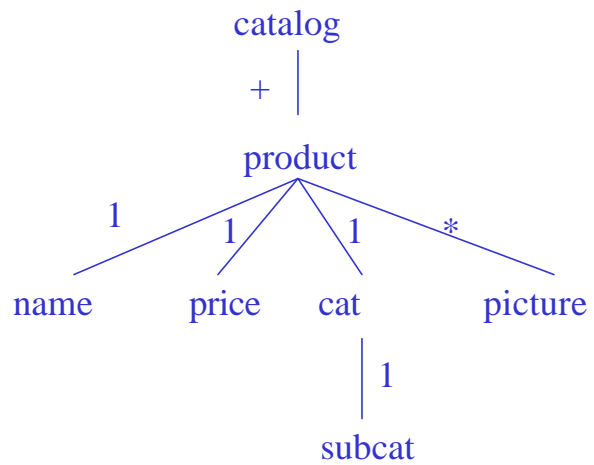
Important assumption:

persistent node ids!

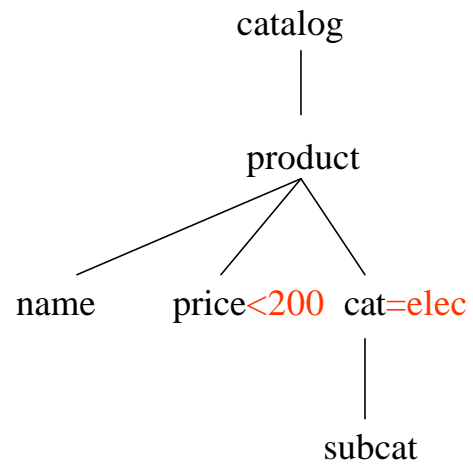
- queries return actual nodes from
the input: can “join” answers
from consecutive queries

Example

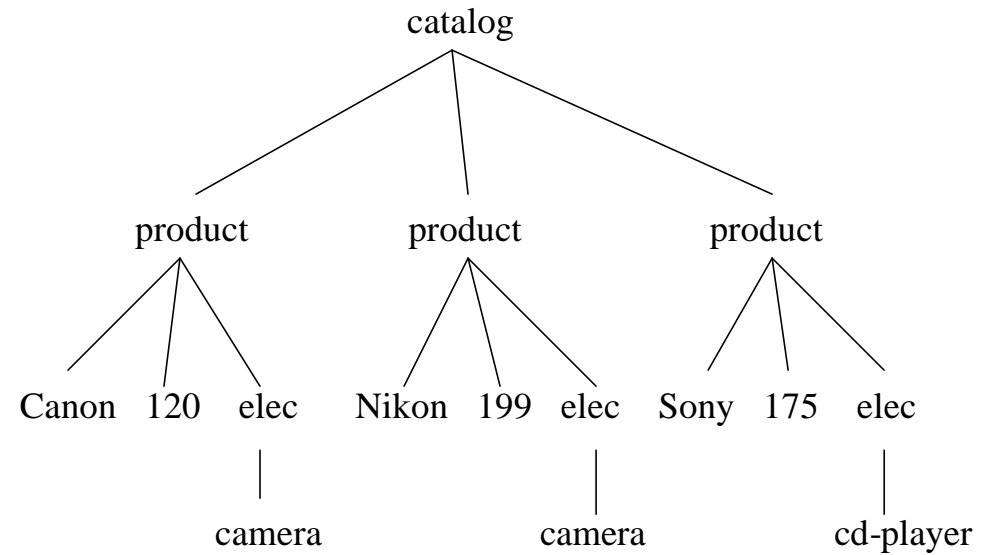
Source DTD (tree type)



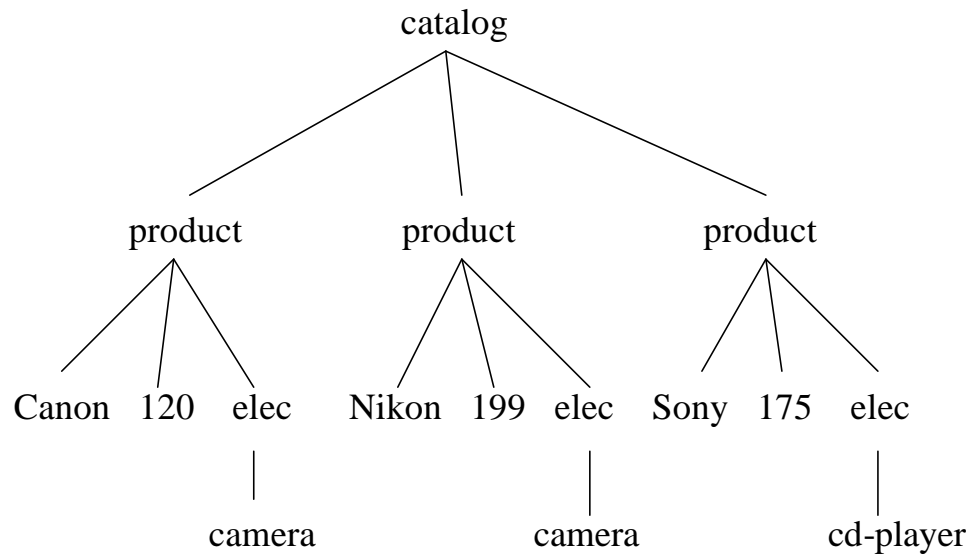
Query 1



Answer to Query 1

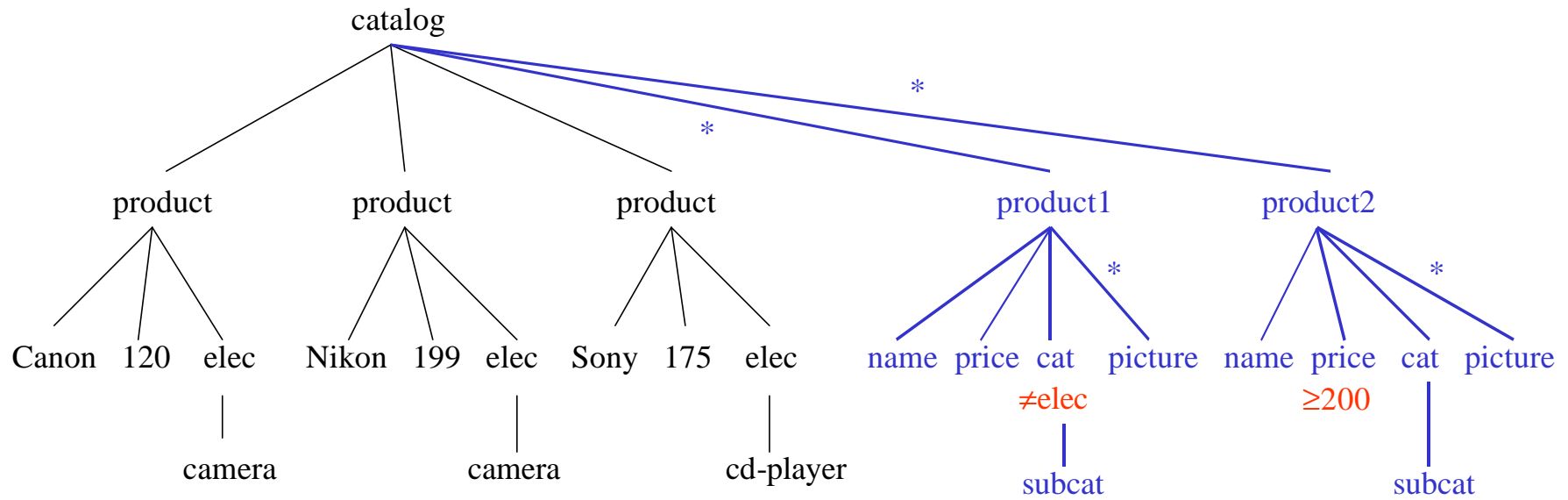


Incomplete information after Query 1



known information
prefix data tree

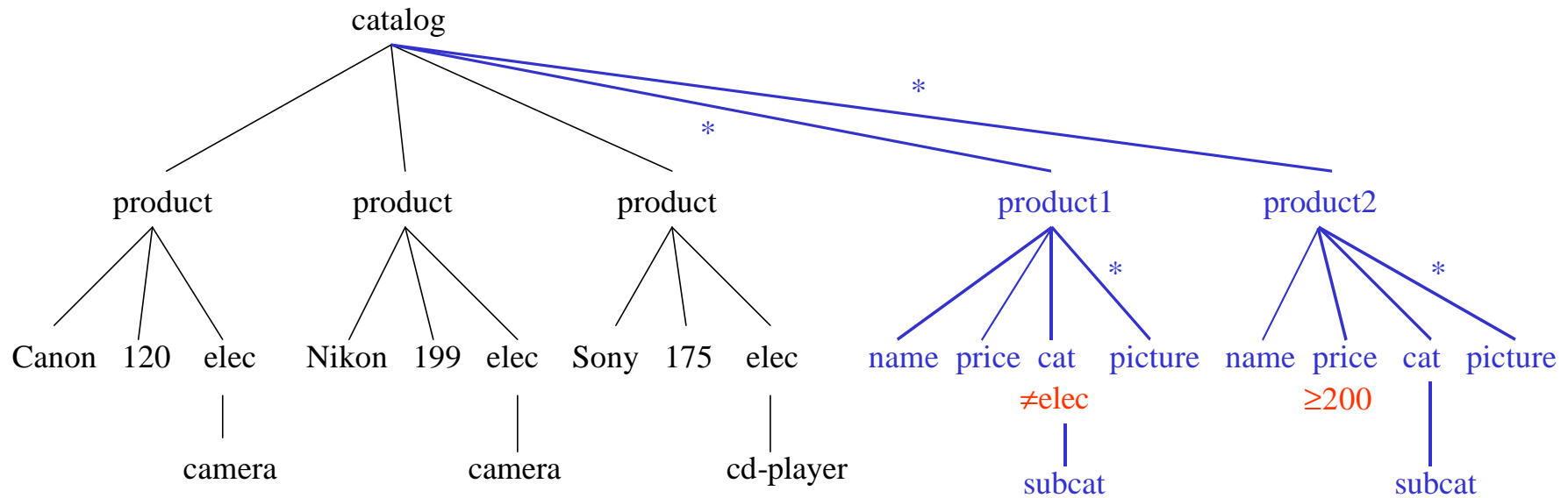
Incomplete information after Query 1



known information
prefix data tree

missing information

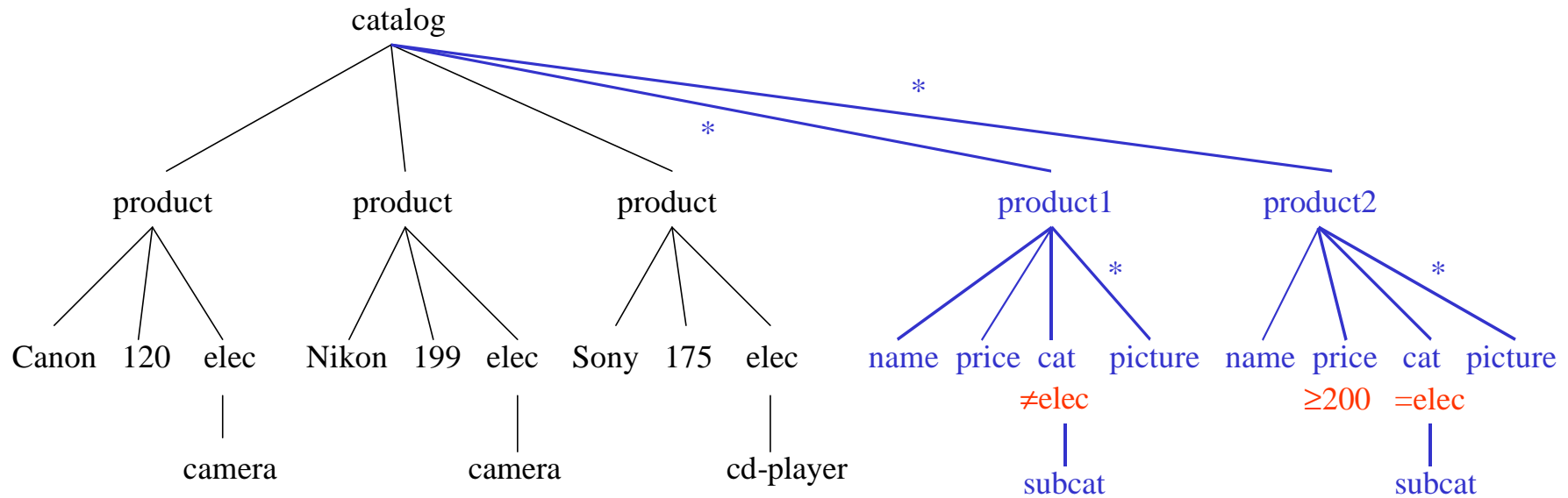
Incomplete information after Query 1



known information
prefix data tree

missing information
extended tree type:
--conditions on data values
--specialization, disjunction

Incomplete information after Query 1



known information
prefix data tree

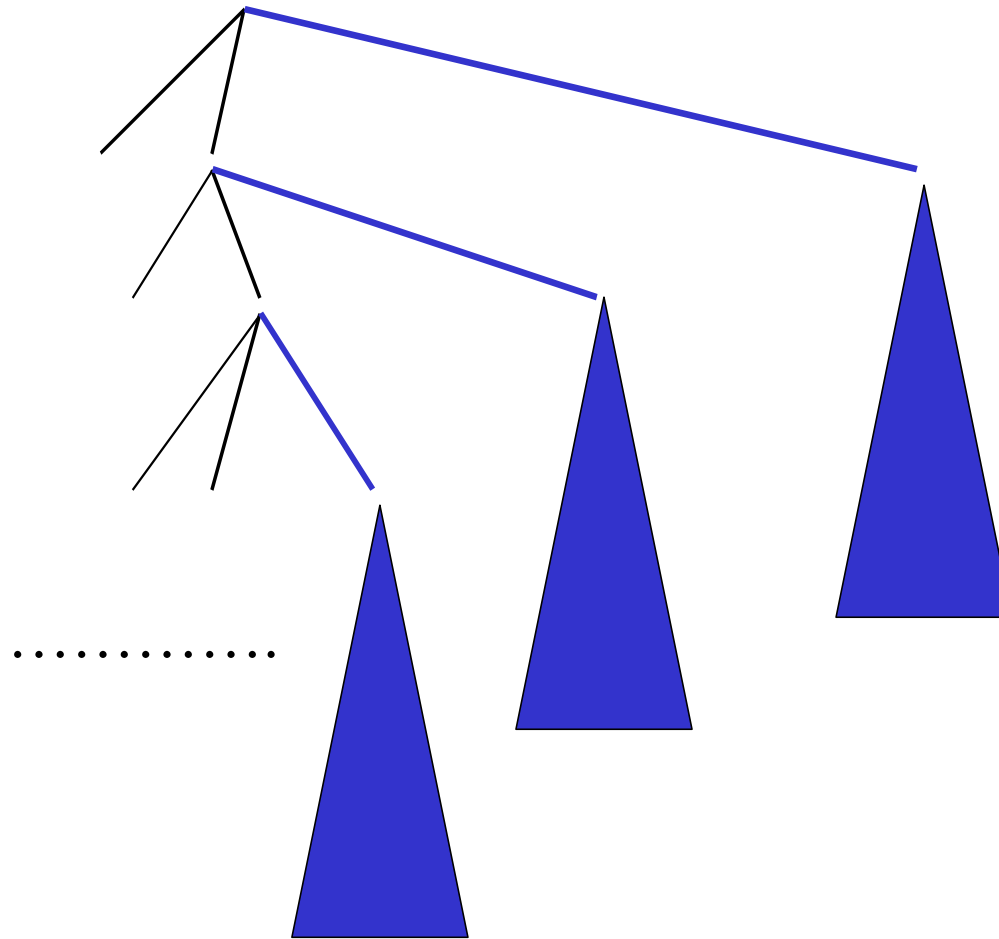
+

Incomplete tree

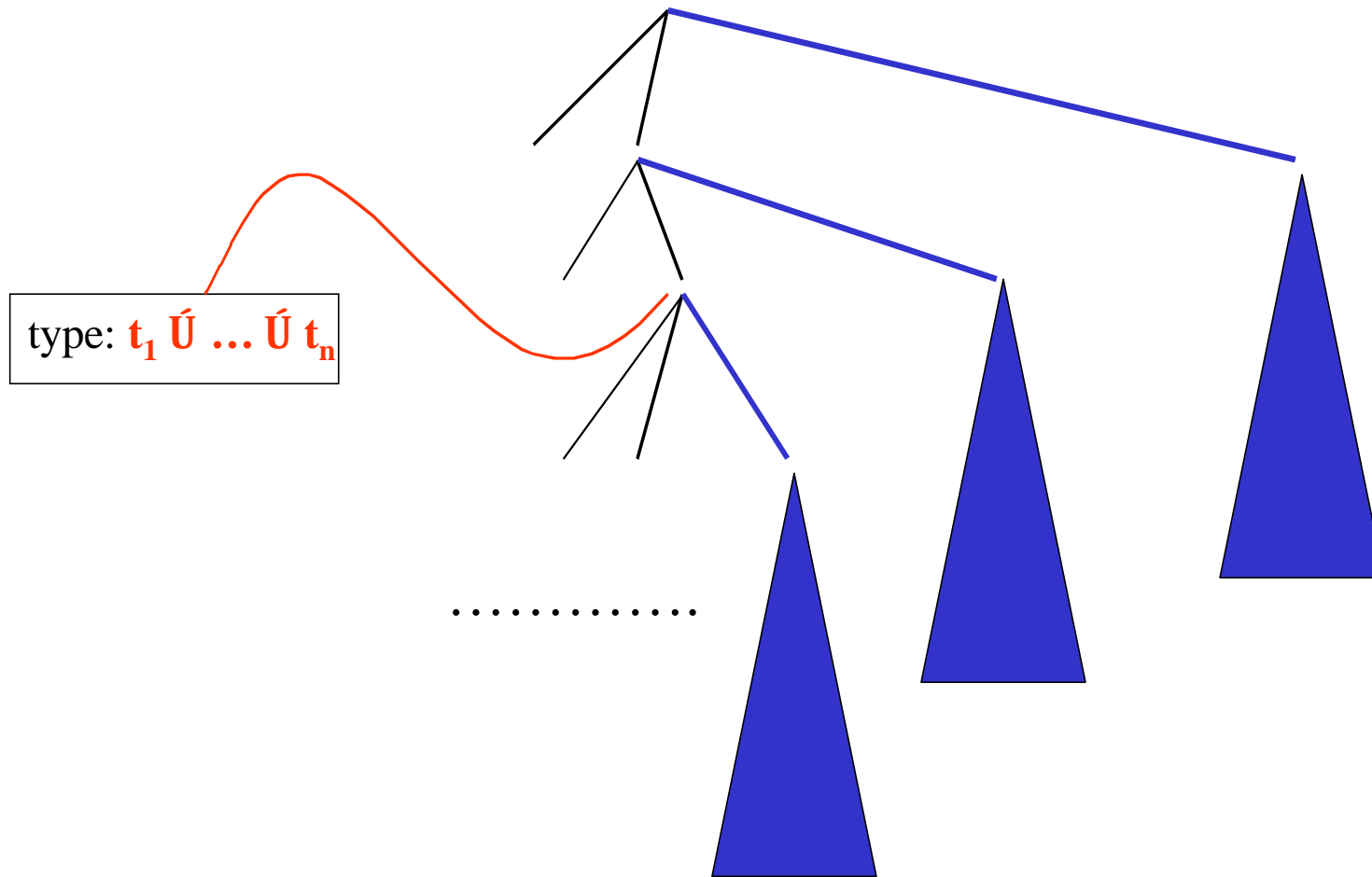
missing information
extended tree type:

- conditions on data values
- specialization, disjunction

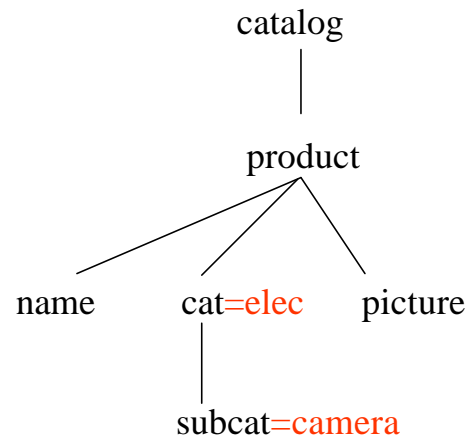
Incomplete tree



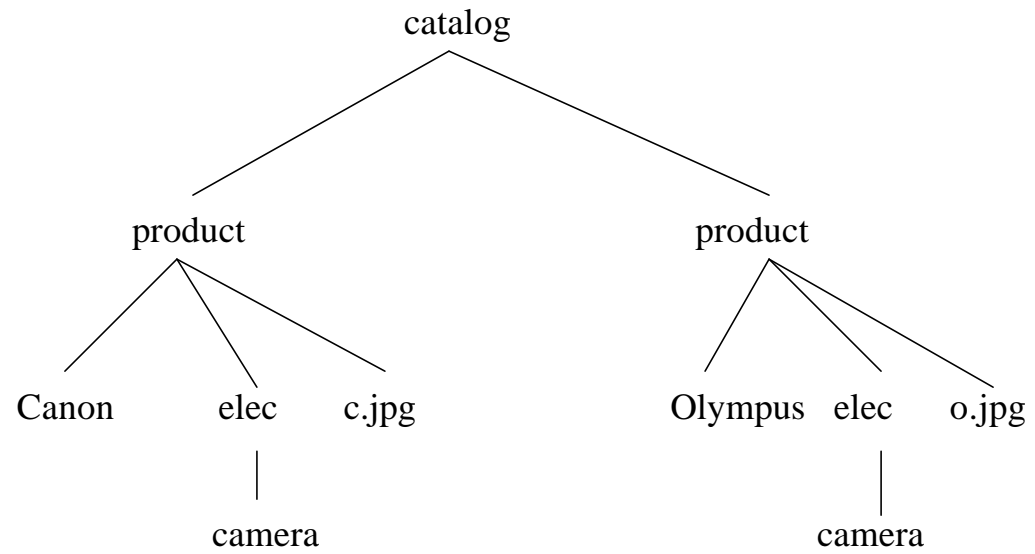
Incomplete tree



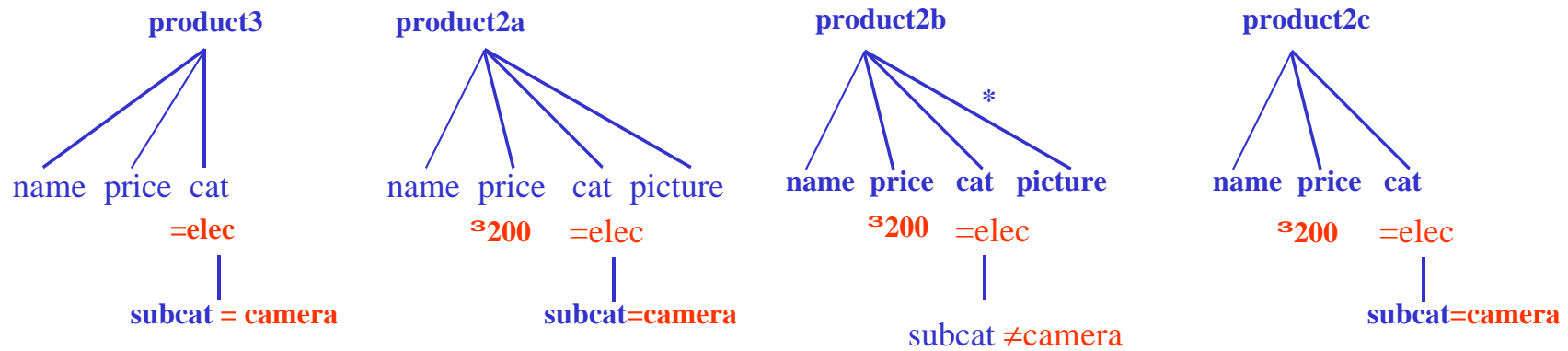
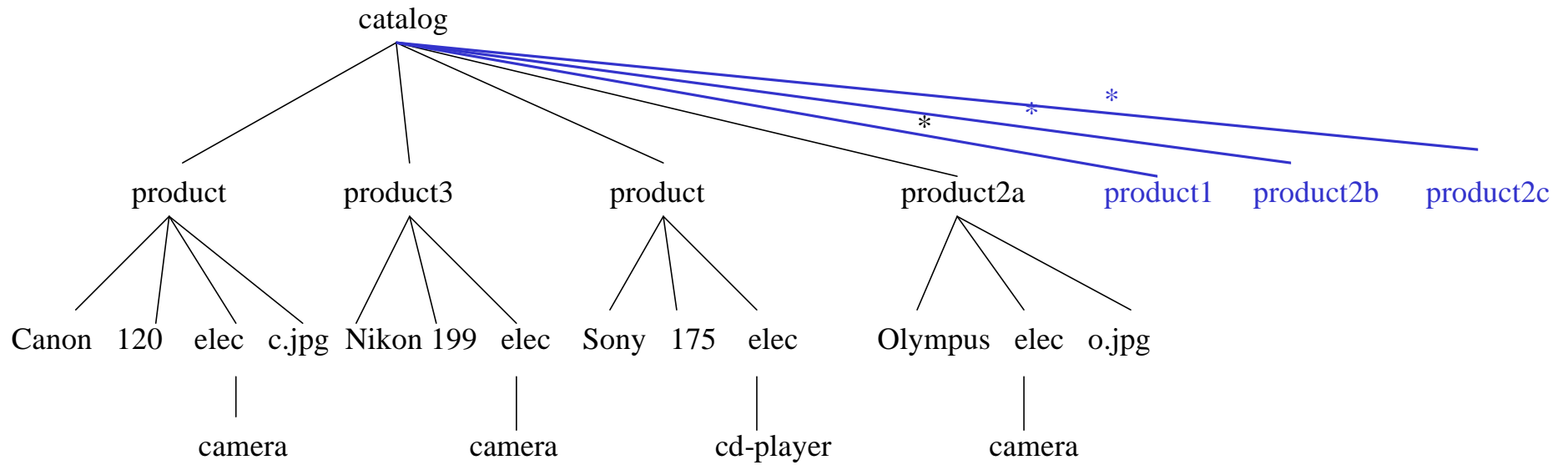
Query 2



Answer to Query 2

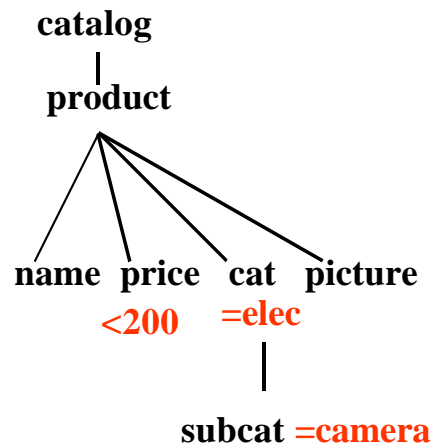


Incomplete tree after Query 2



Suppose next query is:

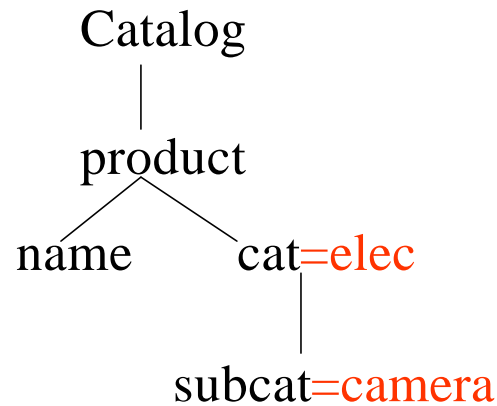
Query 3: find the name, price and pictures of all cameras costing less than \$200 and having at least one picture



Can be fully answered using available information

Note: need tests of the form $q(\text{rep}(T)) = F$

Query 4: find all cameras

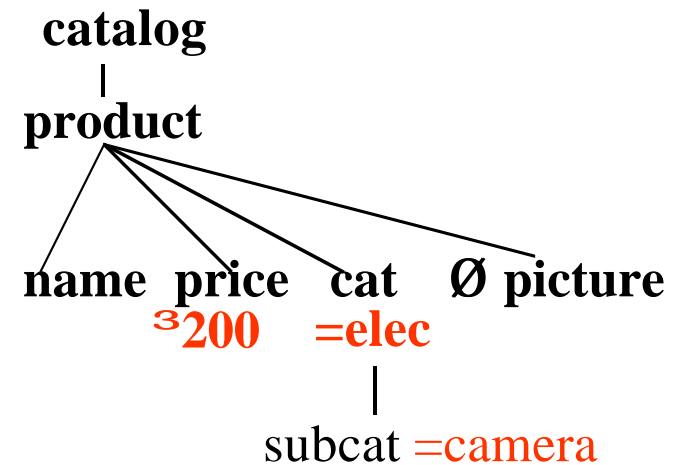


Using available information can:

- provide the complete list of cameras that are less than \$200 or have a picture;
- tell the user that there may be more cameras (that are expensive and have no pictures).

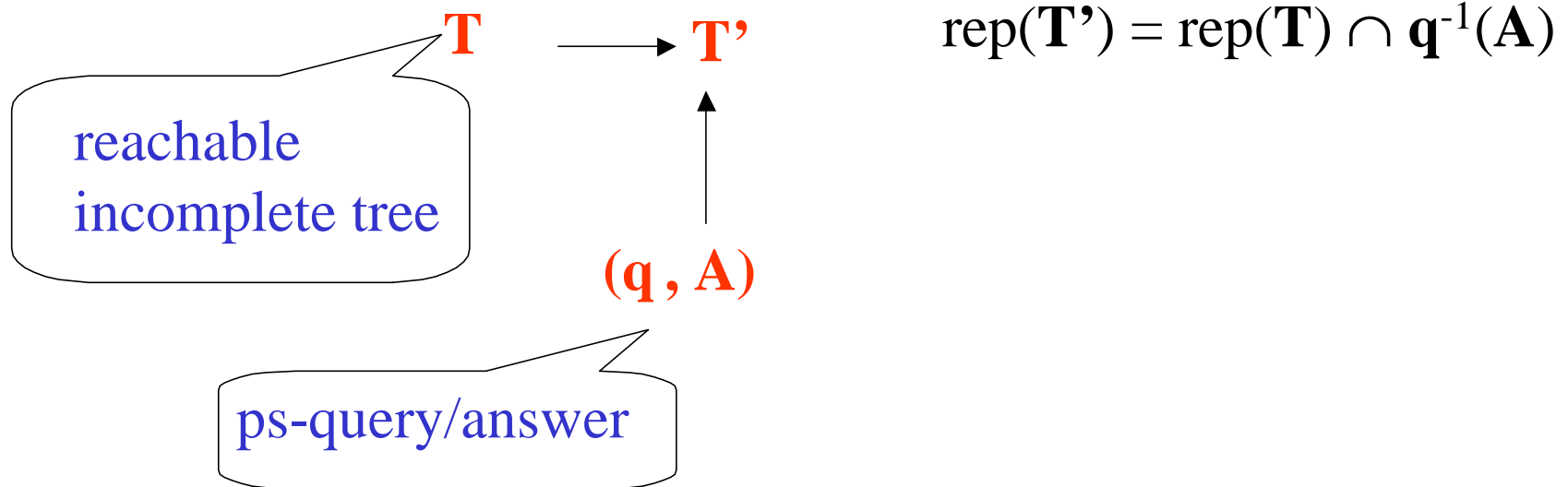
Can fully answer query by asking the **extra query**:

Query 5: find the cameras that cost at least \$200 and have no picture.

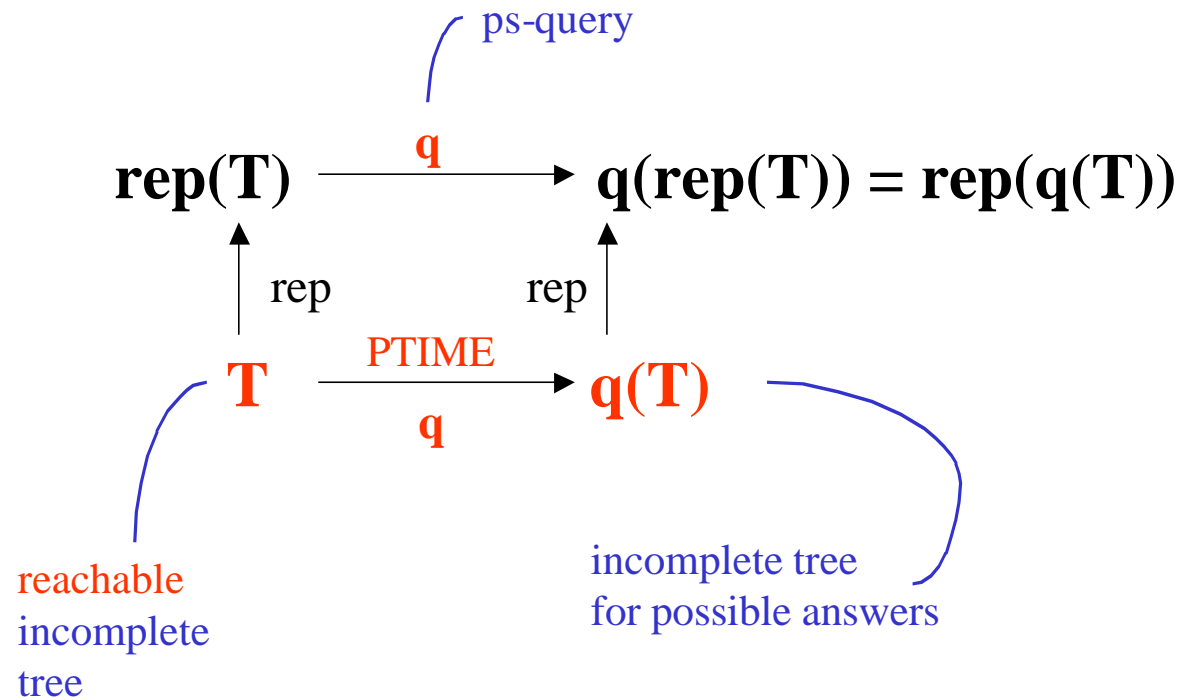


Measuring up to the wish list

- Incomplete trees can be
incrementally computed in PTIME



Strong representation system for ps-queries



- Can check in PTIME if a ps-query can be fully answered with incomplete information described by incomplete tree

Note: in terms of answering queries using views:

can q be answered using views provided by q_1, \dots, q_k

- Can complete answer: given q and T , generate in PTIME a non-redundant set of queries sufficient to fully answer q

Non-redundant:

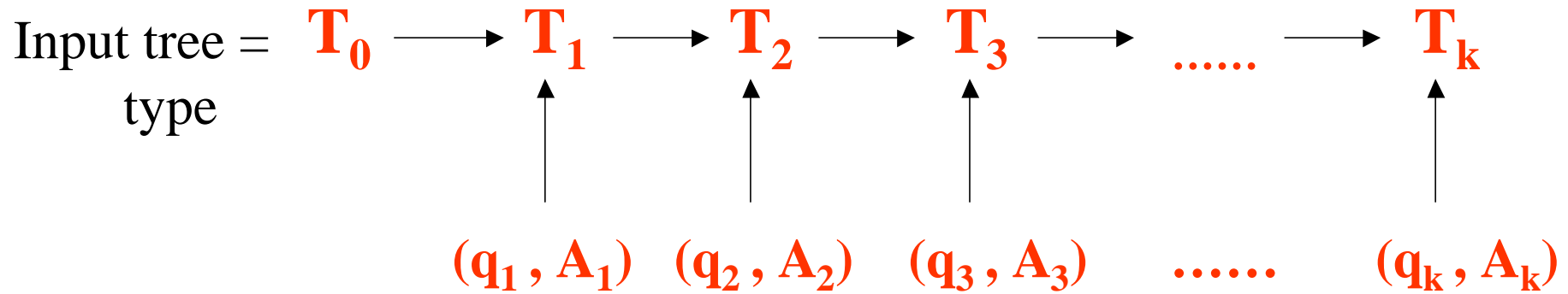
-- no existing nodes are retrieved again

-- no queries are asked that always have empty answer on the possible inputs

- Key fact: can check in PTIME if $q(\text{rep}(T)) = \Phi$

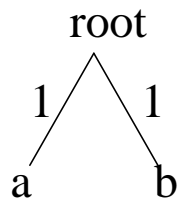
Exponential blow-up

Drawback: incomplete tree can become exponential wrt overall query/answer sequence

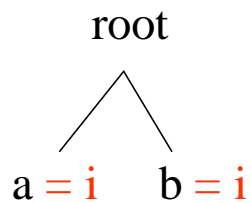


Example

input type:



q_i



$1 \leq i \leq n$

empty answers

Basic trade-off: conciseness vs. efficiency of manipulations

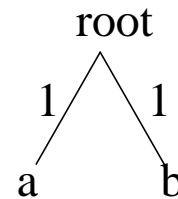
NP-complete whether t is possible prefix for inputs compatible with (q_i, A_i) and T_0

Dealing with the exponential blowup

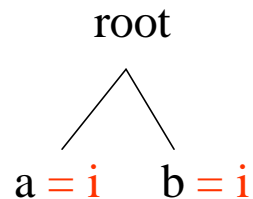
- **augment incomplete trees**: conjunctions of disjunctions of types

Example

input type:



Q_i



empty answers

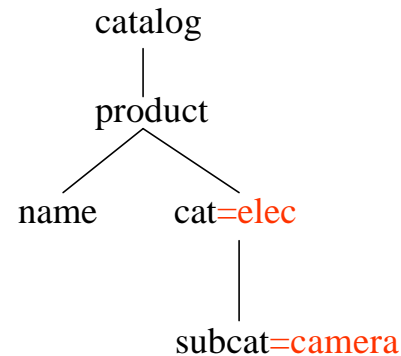
root: $(a_1 \dot{\cup} b_1) \dot{\cup} \dots \dot{\cup} (a_n \dot{\cup} b_n)$
 where $a_i : a \neq i$ $b_i : b \neq i$

--size of incomplete tree stays polynomial

--increase in complexity of manipulations

Dealing with the exponential blowup

- **restrict tree types and ps-queries**
 - non-recursive tree types
 - ps-queries testing data values only along one path



- size of incomplete tree stays polynomial
- no increase in complexity** of manipulations

Dealing with the exponential blowup

- **heuristics** to deal with large incomplete trees
 - ask linear set of **additional queries** (always possible)
can guarantee overall polynomial size
 - gracefully loose** some of the information
to shrink the incomplete tree (e.g., “un-specialize” types)

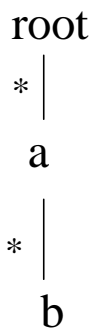
Extensions of query language yield problems:

- No user-friendly representation of incomplete info
- No strong representation system
- High complexity for basic manipulations
- Undecidability of basic questions

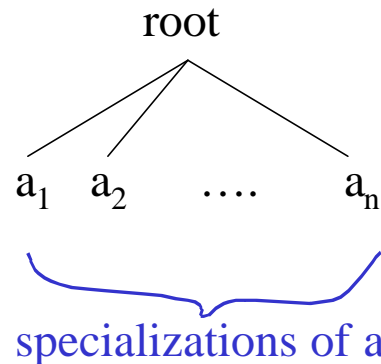
Branching

- Incomplete trees remain strong representation system
- Can be maintained incrementally in PTIME
- But: $q(\mathbf{T})$ exponential in \mathbf{T}

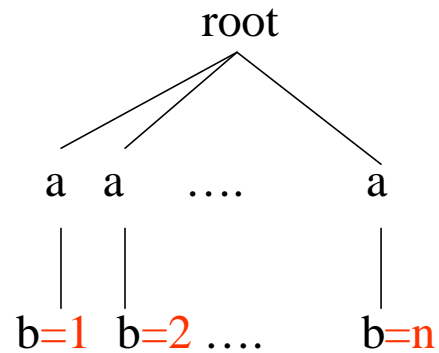
input type



incomplete tree \mathbf{T}



query q

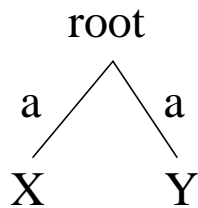


$n!$ possibilities...

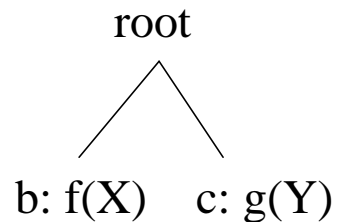
Branching + constructed answers

- no (known) strong representation system

where



construct

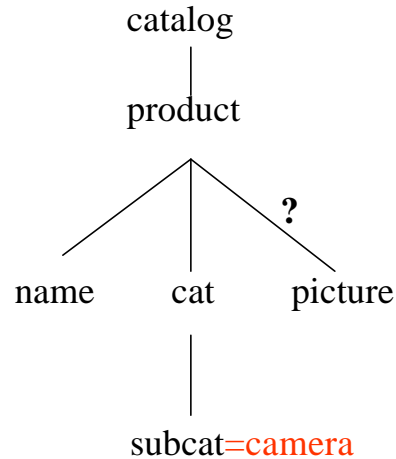


output: $b^n c^n$

Note: connection to **type inference**

Branching + optional subtrees

Find all cameras and their pictures (if any)



Complexity: given tree type \mathbf{T} , query/answer $\langle \mathbf{q}, \mathbf{A} \rangle$, new query \mathbf{q}'

\mathbf{t} is a sure prefix for $\mathbf{q}'(\text{rep}(\mathbf{T}) \subseteq \mathbf{q}^{-1}(\mathbf{A}))$ is **co-NP-complete**

There can be no strong representation system for which incremental maintenance, computing $\mathbf{q}(\mathbf{T})$, and testing that \mathbf{t} is a sure prefix of $\text{rep}(\mathbf{T})$ are all in PTIME

Very powerful transformers: k-pebble transducers with data value selections

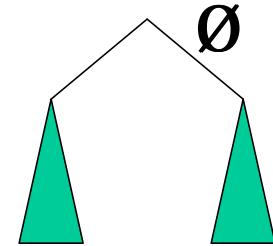
[Milo,Suciu,V.]

- subsume tree manipulation core of XML-QL and XSL
- **k-pebble automata** provide representation system,
can be maintained in PTIME wrt overall sequence
- no (known) strong representation system
- testing $\text{rep}(\mathbf{T}) = \Phi$ is **non-elementary** !

Branching + join on data values + negation

Comparisons of data values, negative sub-trees

Basic questions are **undecidable**:



Given: input tree type **T**

query/answers $\langle q_1, A_1 \rangle \dots \langle q_n, A_n \rangle$

query **q**

Test whether **q** is **always empty** on compatible inputs

There cannot exist an effective strong representation system for which such questions are decidable

Similar:

Branching + join on data values
+ optional sub-trees + construction

Trade-off between negation and optional sub-trees + construction

Recursive path expressions and join on data values

Proofs: reduce implication of FDs + INCDs,
emptiness of intersection of two CFGs

Other issues

- Persistent node id assumption
 - without it approach still works, but weaker
 - cannot enrich info about nodes
- Order: input tree, input DTD, used by queries

New problems:

q_1 : list all a elements

q_2 : list all b elements

q_3 : list all elements -- can answer using q_1, q_2 ?

yes: if input DTD is $a^* b^*$

no: if input DTD is $(a+b)^*$

Conclusion

- Simple framework for acquiring, maintaining, and querying incomplete XML documents
- Answer queries as best possible given incomplete information, or generate additional non-redundant queries to provide full answers
- Limitations: simple DTDs and queries, persistent id assumption, no order
- But: even small extensions lead to serious problems