VLDB 2007

Vienna, Austria

# Matching Twigs in Probabilistic XML

*Benny Kimelfeld* & *Yehoshua Sagiv*

The Selim and Rachel Benin School of Engineering and Computer Science

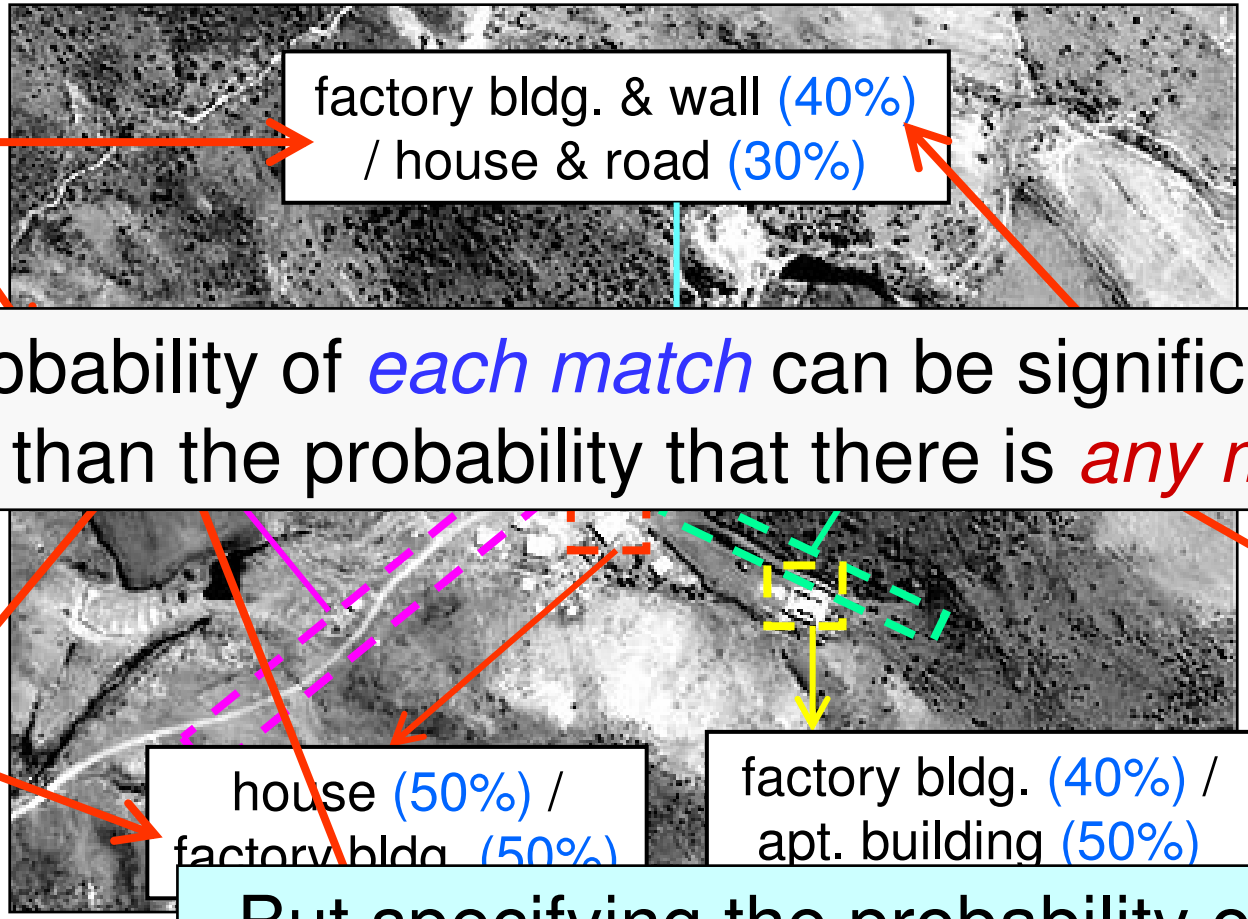
האוניברסיטה העברית בירושלים
The Hebrew University of Jerusalem

# *Example: Scanning Aerial Photography*

Find regions that include a factory building and a road

… with a high probability

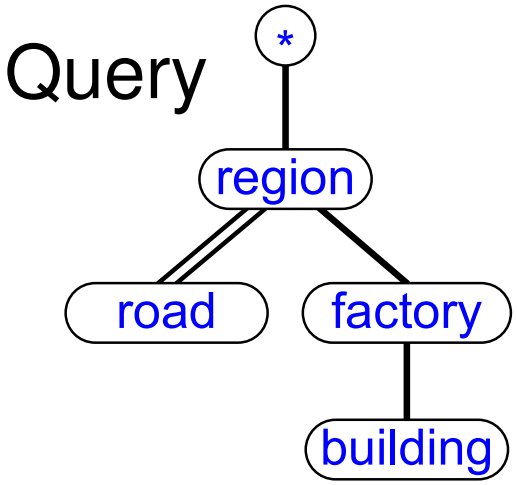
What is the probability that this region is an answer (i.e., includes a factory building and a road)?




The probability of *each match* can be significantly smaller than the probability that there is *any match*

But specifying the probability of each match **does not** answer the question!

# A Database Point of View

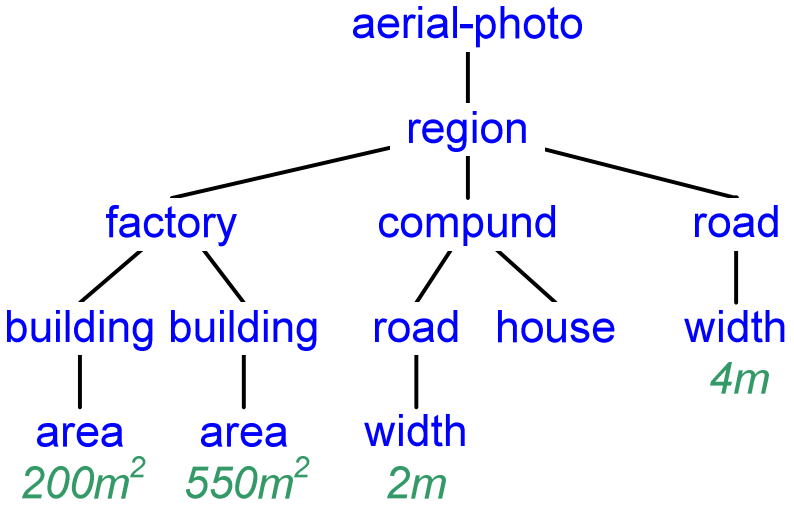



**Querying probabilistic data:**

Each answer has an amount of *certainty*:
The probability of being obtained when querying a random database

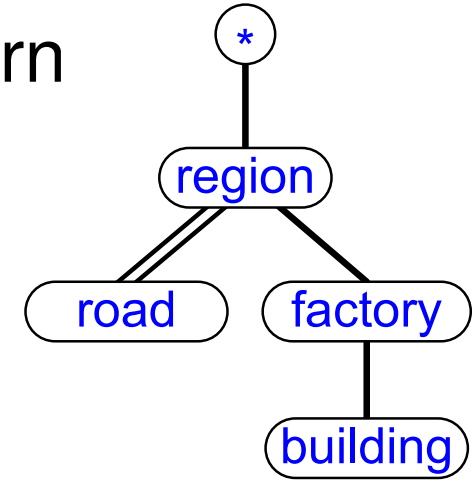
Probabilistic Data

A prob. process for generating random data



# What Query Should We Pose?

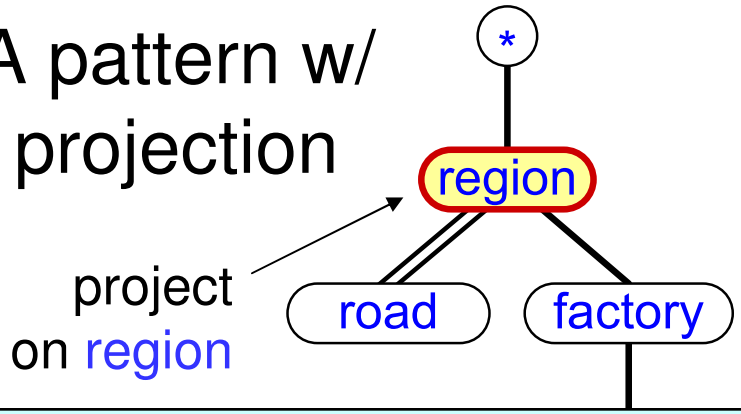
A pattern




- An answer is a match
- What is the probability of each *specific match*?
- What is the probability of each pair of road & factory building?

---

A pattern w/ projection




**This is what we need!**

- An answer is a projection of one or more matches
- What is the prob. of each answer **after the projection**?
- For each region, what is the prob. that it has *some* pair of road & factory building?

# Another Example

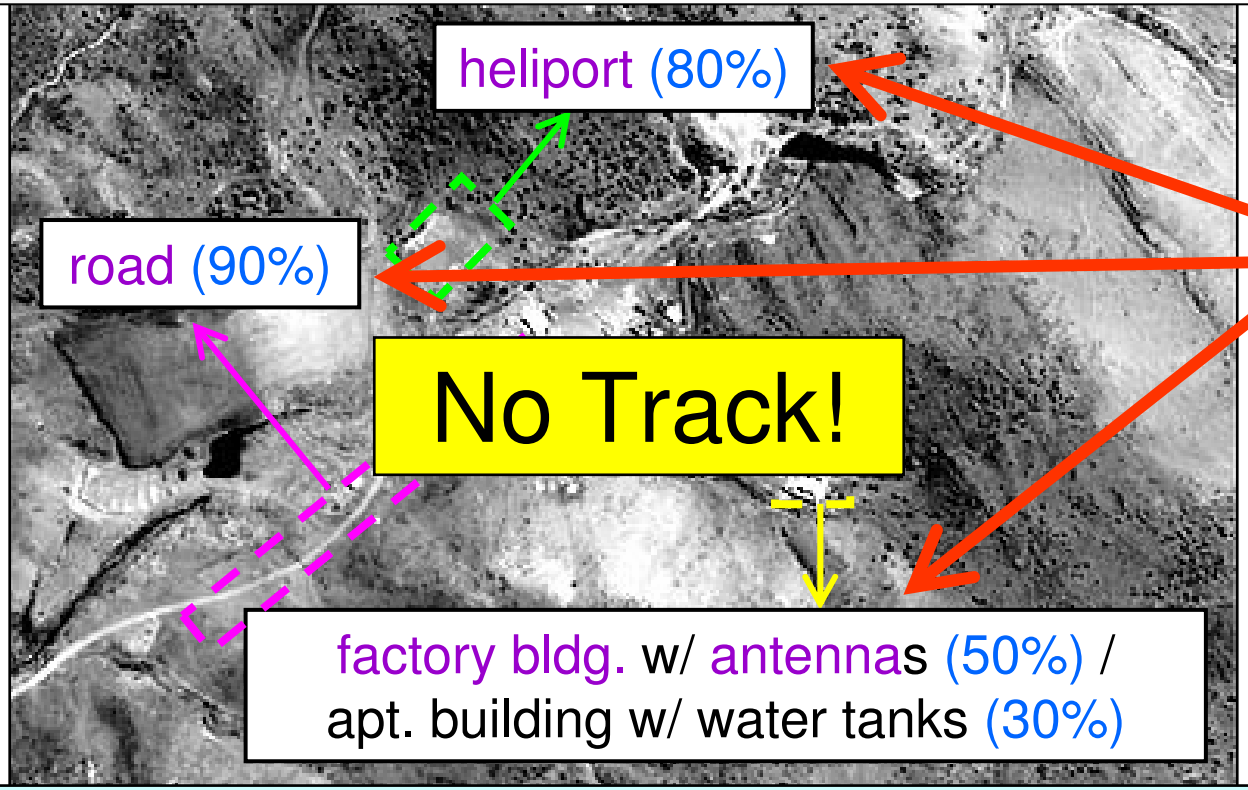
Find the following objects in one region:

A factory building, a road, an antenna, a heliport, a track

# *Finding a Partial Match*

Find the following objects in one region:

A factory building, a road, an antenna, a heliport, a track



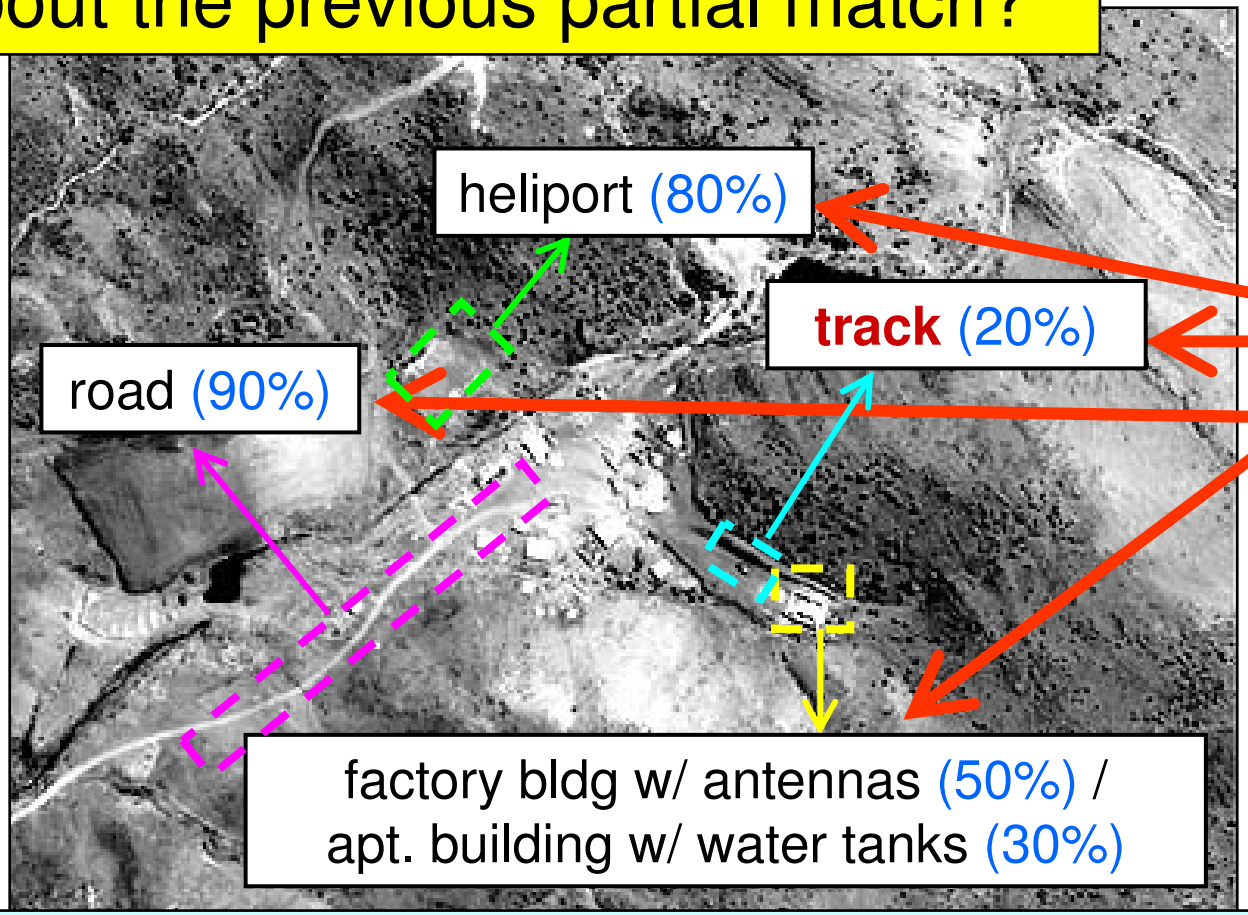

partial match (36%)

For many applications, that's good enough …

Should we just filter out the whole match?

**Does not make sense!**

What about the previous partial match?

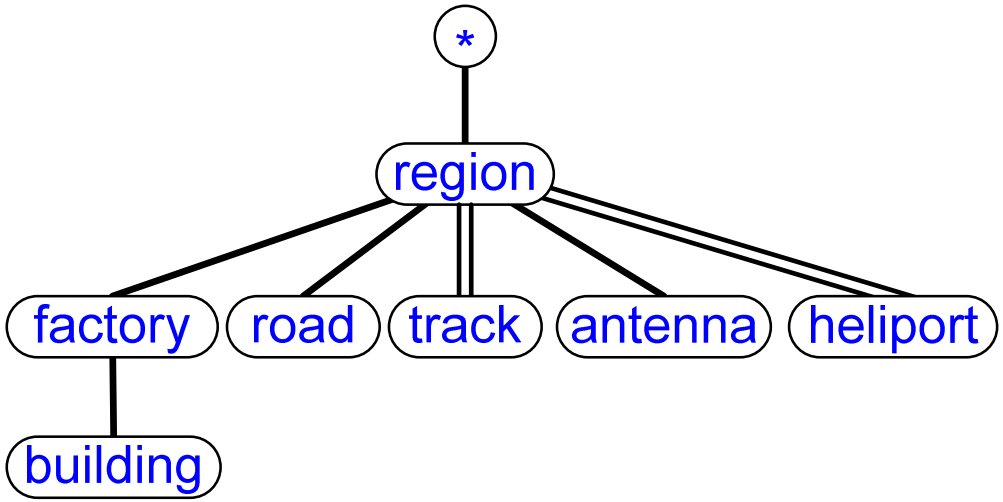



match (7.2%)

The probability may be too low to be of any interest!

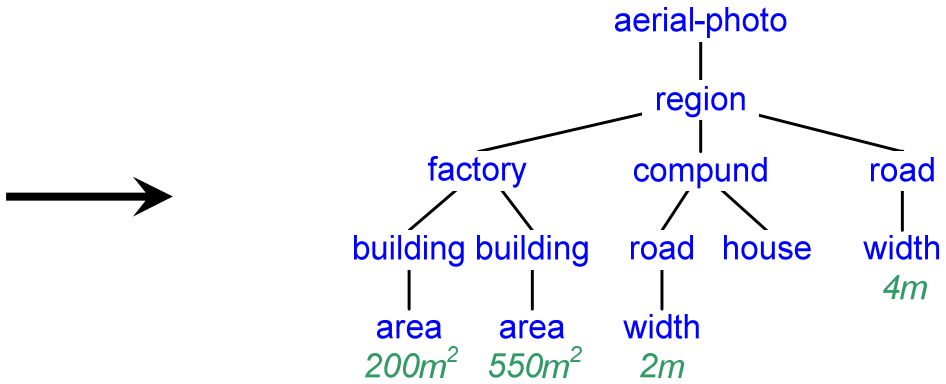
# *Finding Maximal Matches*

A pattern




The goal is to find the maximal among the partial matches with a *sufficient probability*

Probabilistic Data



# Querying Prob. Data: Earlier Work

- Projection and incomplete semantics were explored for relational models
  - Projection: Very simple queries can be highly intractable (data complexity) [Dalvi & Suciu, VLDB 04]
  - Maximally joining relations: Tractable under data complexity, generally intractable under query-and-data complexity [Kimelfeld & Sagiv, PODS 07]
    - Yet tractable for important classes of schemas
- **None** of these paradigms studied in the context of prob. XML (only complete matches w/o projection)

But they are more relevant to prob. XML since, as the paper shows, they become tractable

In the paper, we also have some preliminary results on the *combination of maximal matches and projection*

# Query evaluation over probabilistic XML

Efficient algorithms and complexity analysis for various paradigms of querying

- Evaluating twig queries with projection
- Evaluating Boolean twig queries
- Finding maximal matches of twigs

In the paper, we explain in detail why our results **do not follow** from previous results on XML/relational models

# *Talk Overview*

1. Introduction

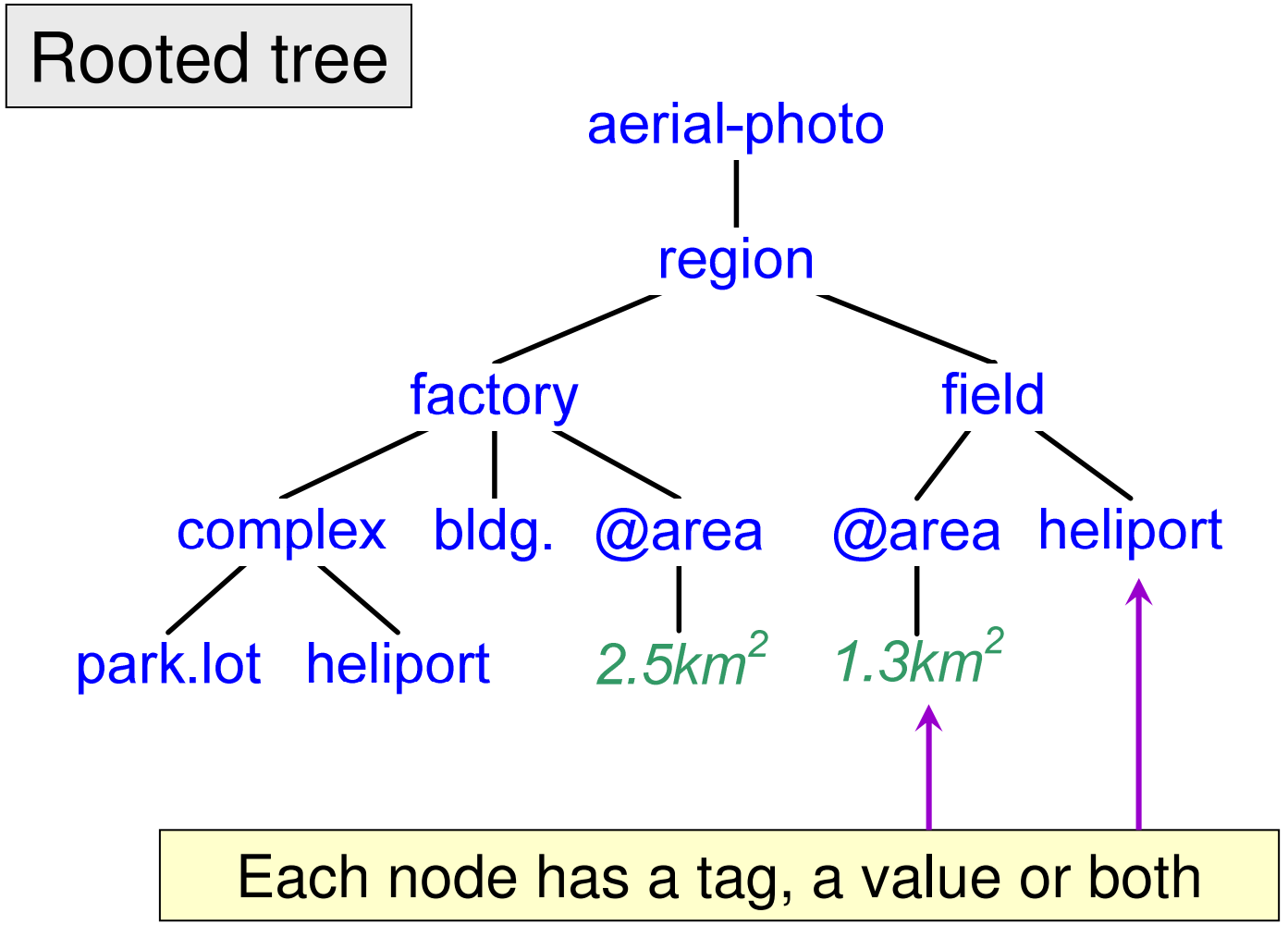
2. Twig Queries over Probabilistic XML
   - XML and Twig Queries
   - Probabilistic XML
   - Querying Probabilistic XML (Complete Semantics)

3. Query Evaluation (Complete Semantics)

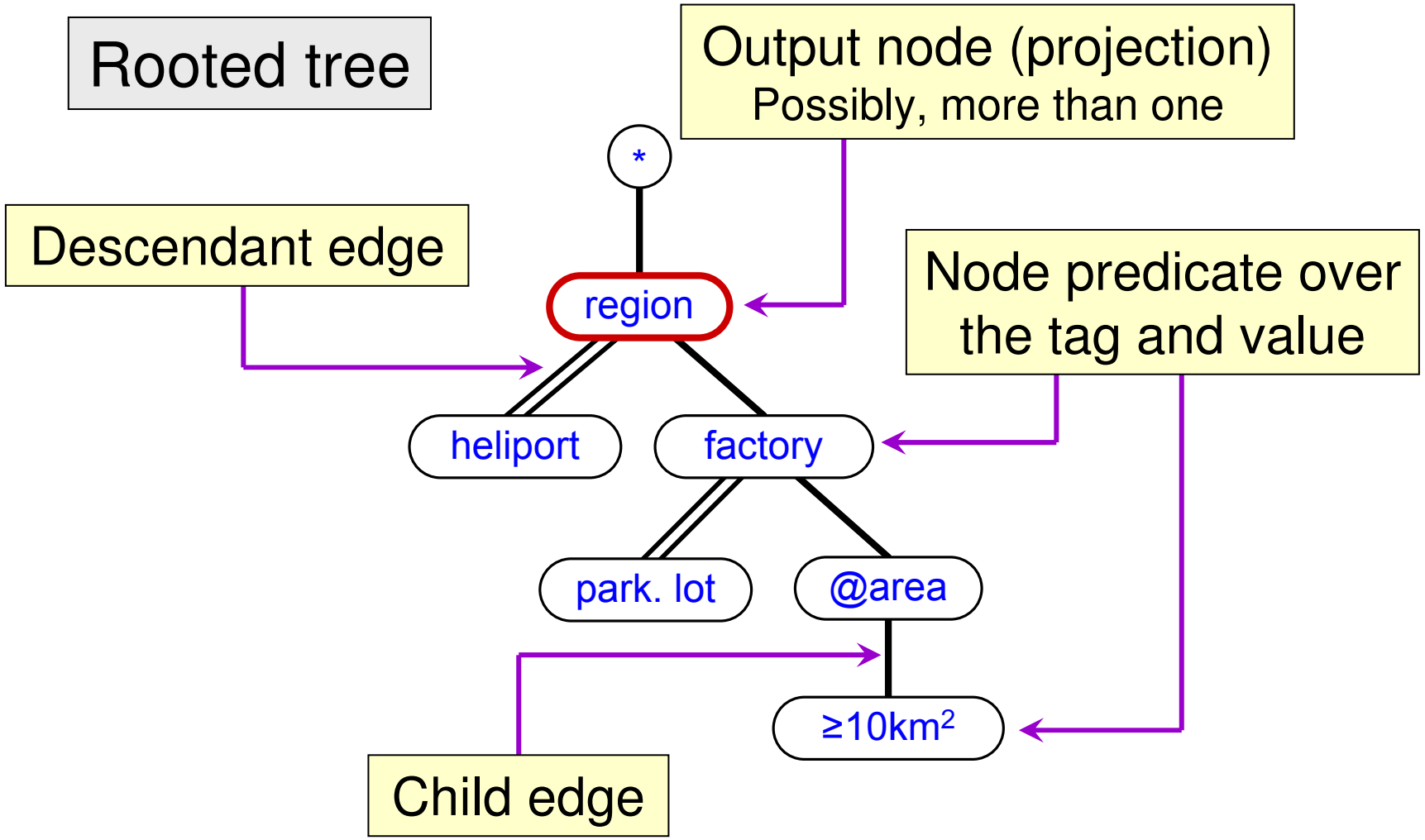
4. Finding Maximal Matches

5. Conclusion, Related and Future Work

# (Ordinary) XML Documents



# Twig Queries



# Matches and Answers

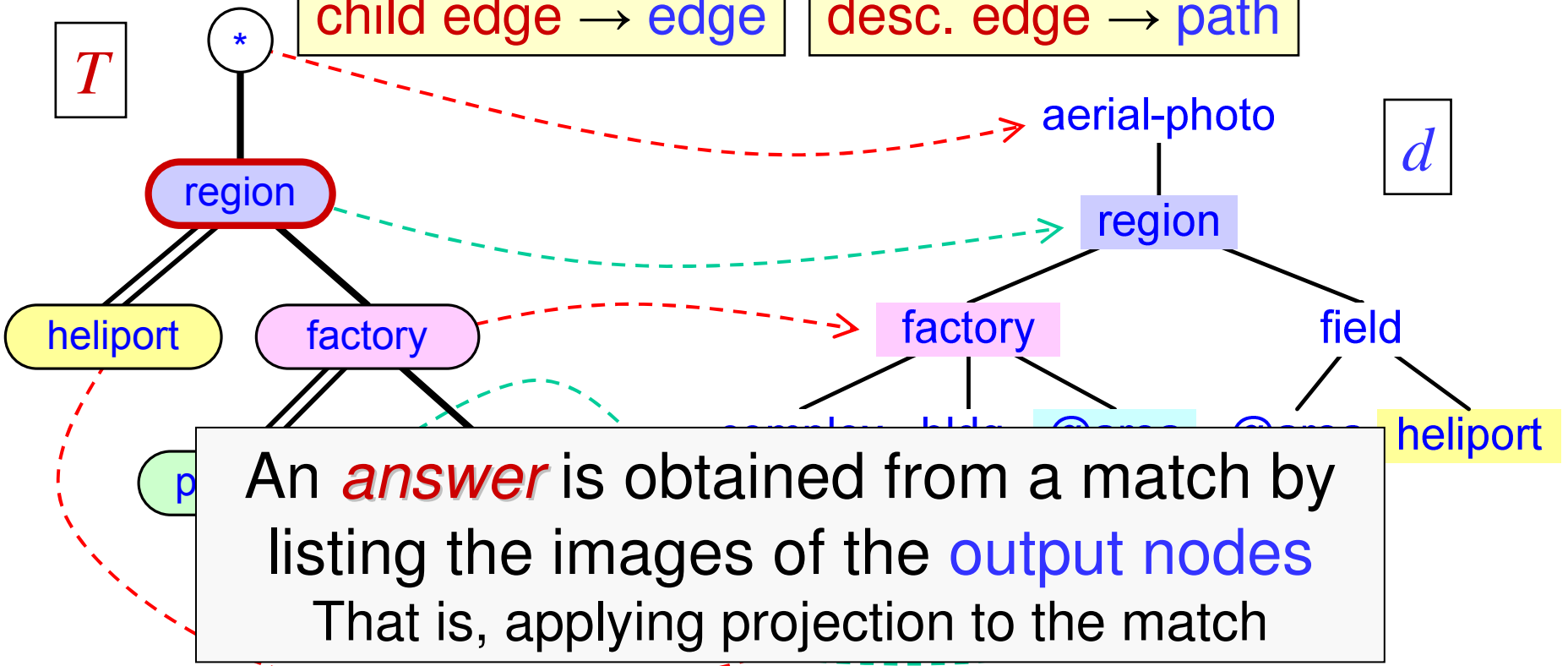
A *match* of a twig $T$ in a document $d$ is a mapping from the nodes of $T$ to those of $d$

root($T$) → root($d$)

node predicates are satisfied

child edge → edge

desc. edge → path



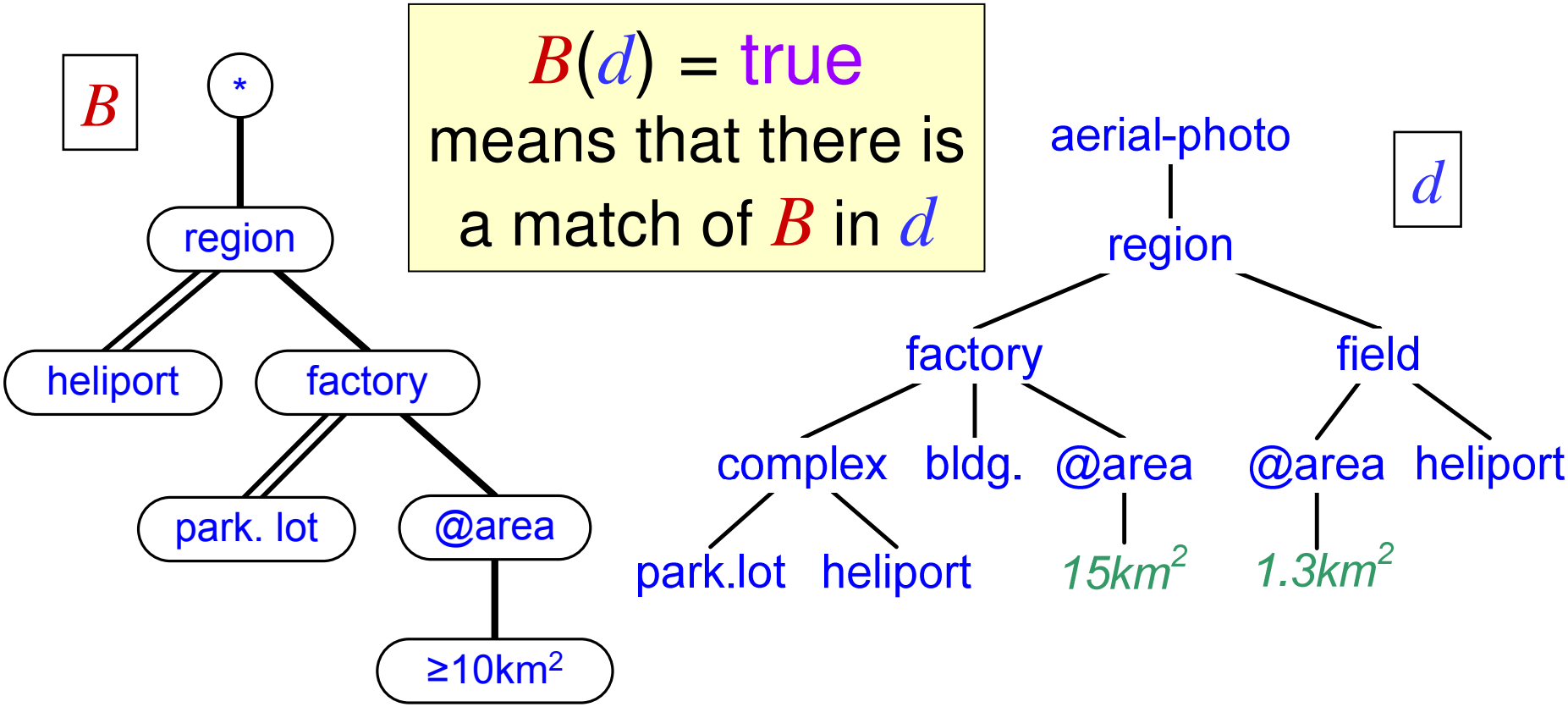

An *answer* is obtained from a match by listing the images of the output nodes

That is, applying projection to the match

# Boolean Queries

A twig without output nodes is a *Boolean* twig

The answer is either *true* or *false*



# *Talk Overview*

1. Introduction

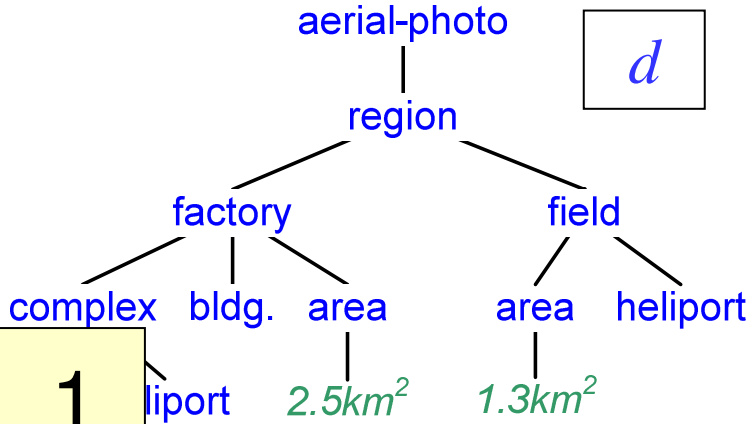
2. Twig Queries over Probabilistic XML
   - XML and Twig Queries
   - Probabilistic XML
   - Querying Probabilistic XML (Complete Semantics)

3. Query Evaluation (Complete Semantics)

4. Finding Maximal Matches

5. Conclusion, Related and Future Work

# *Probabilistic XML*




$$\sum_{d} \Pr(d) = 1$$

## Probabilistic XML document

A probabilistic process of generating ordinary XML documents

## Random Instance

An ordinary XML document $d$, generated with probability $\Pr(d)$

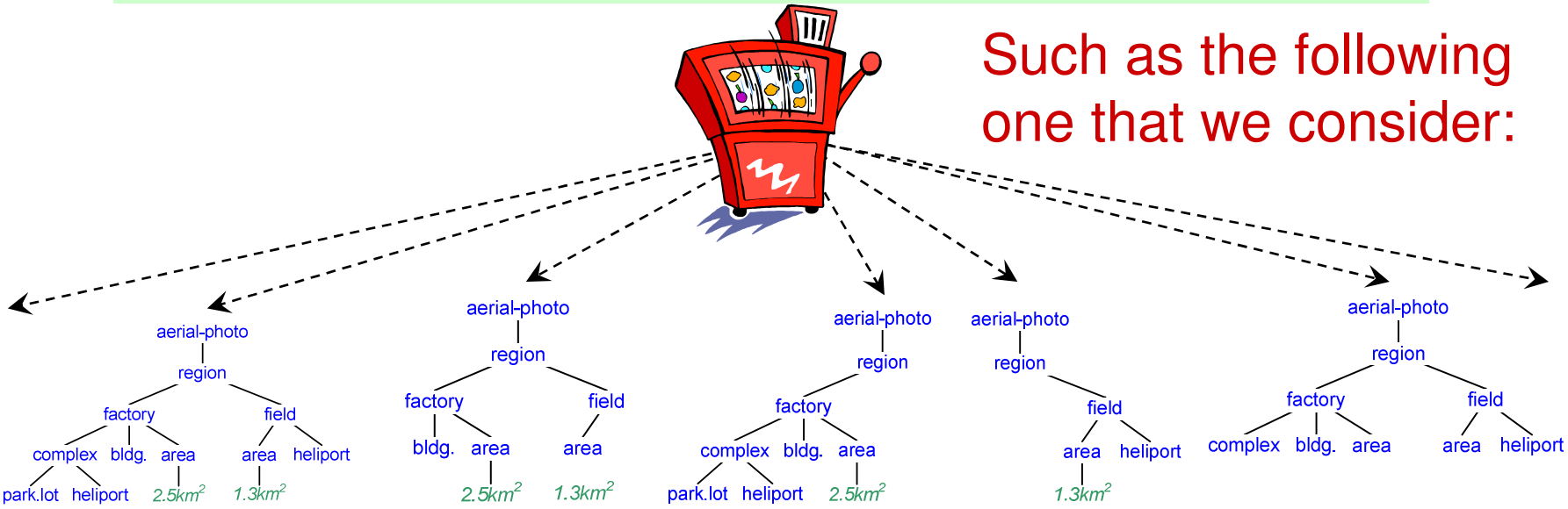
# *Implicit Representations*

In practice, the probability space may be huge

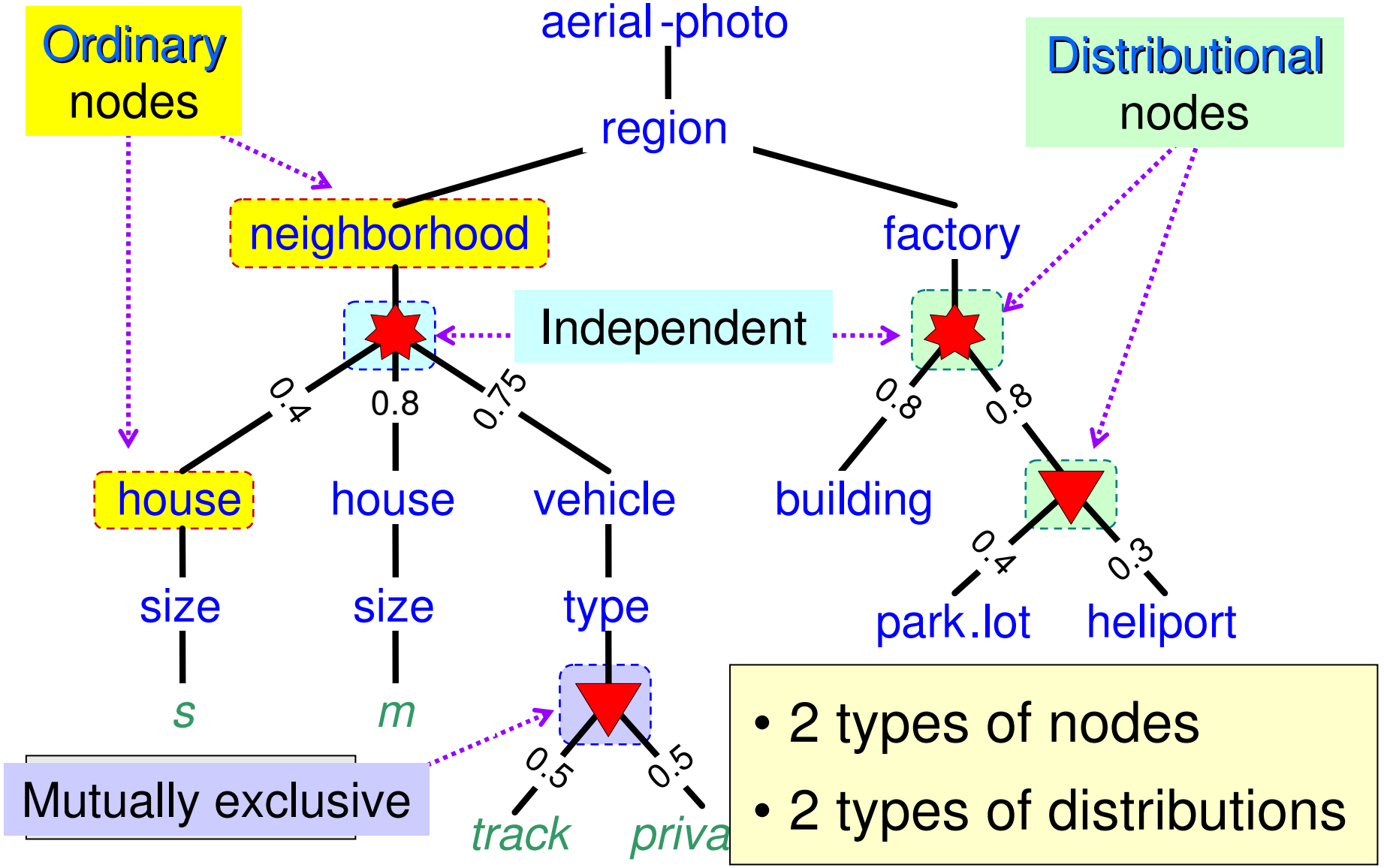
E.g., uncertainty is many small pieces of data

It is unrealistic to represent the probabilistic document by explicitly specifying the entire space

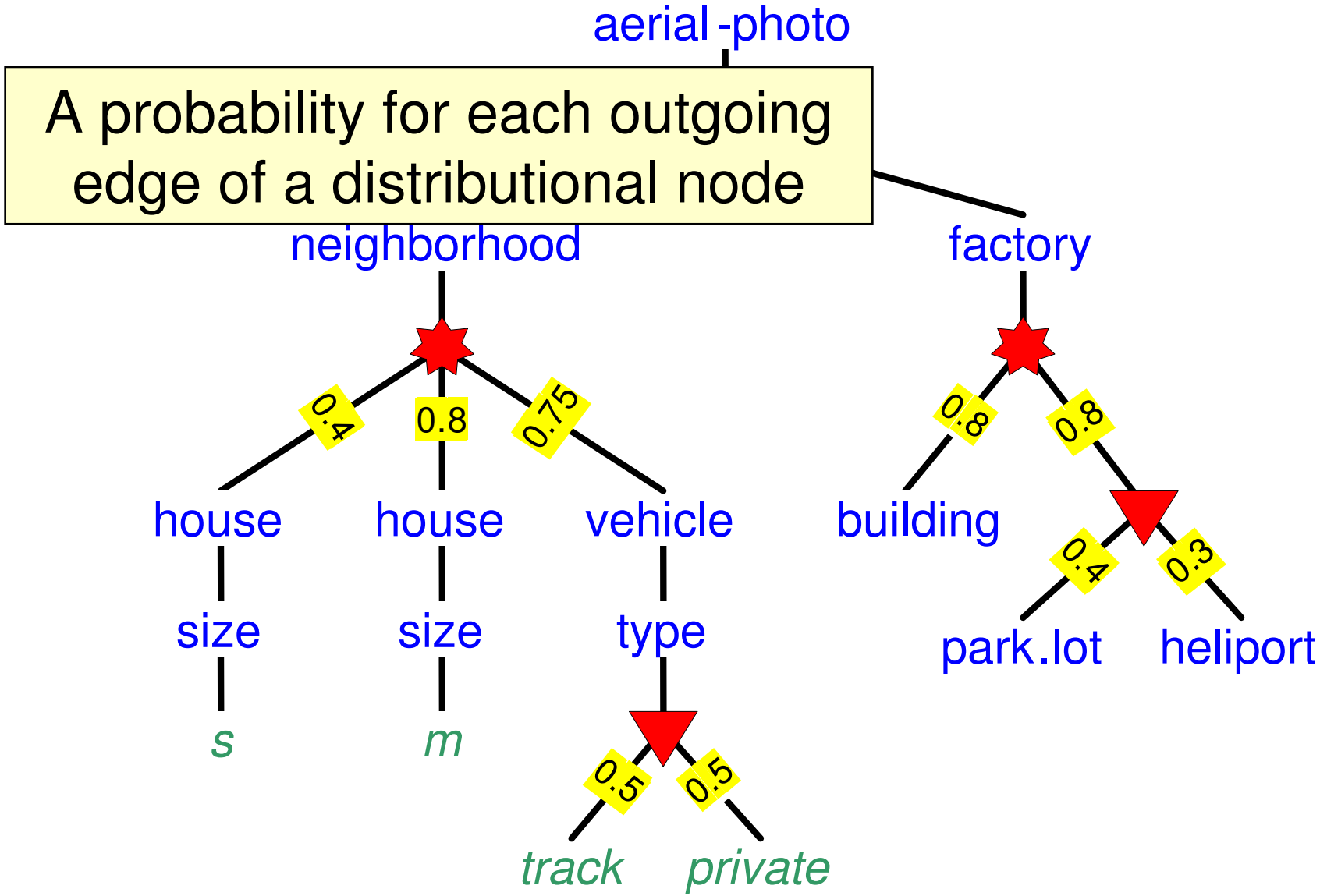
We usually explore *implicit representations*



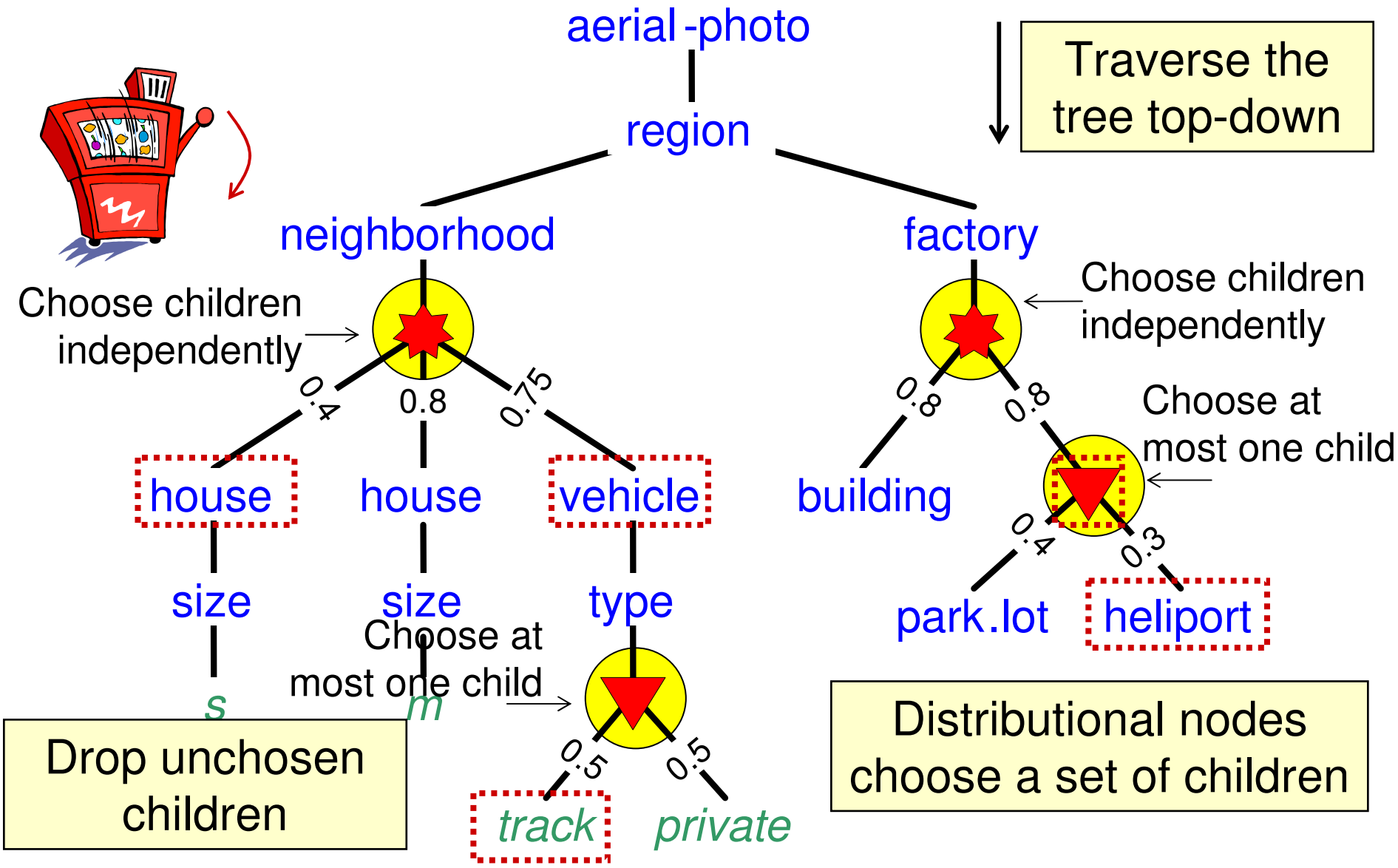
# A ProTDB Document [Nierman & Jagadish 02]



# *A ProTDB Document* [Nierman & Jagadish 02]



# Instance Generation: Step 1



# Instance Generation: Step 2




Drop the distributional nodes

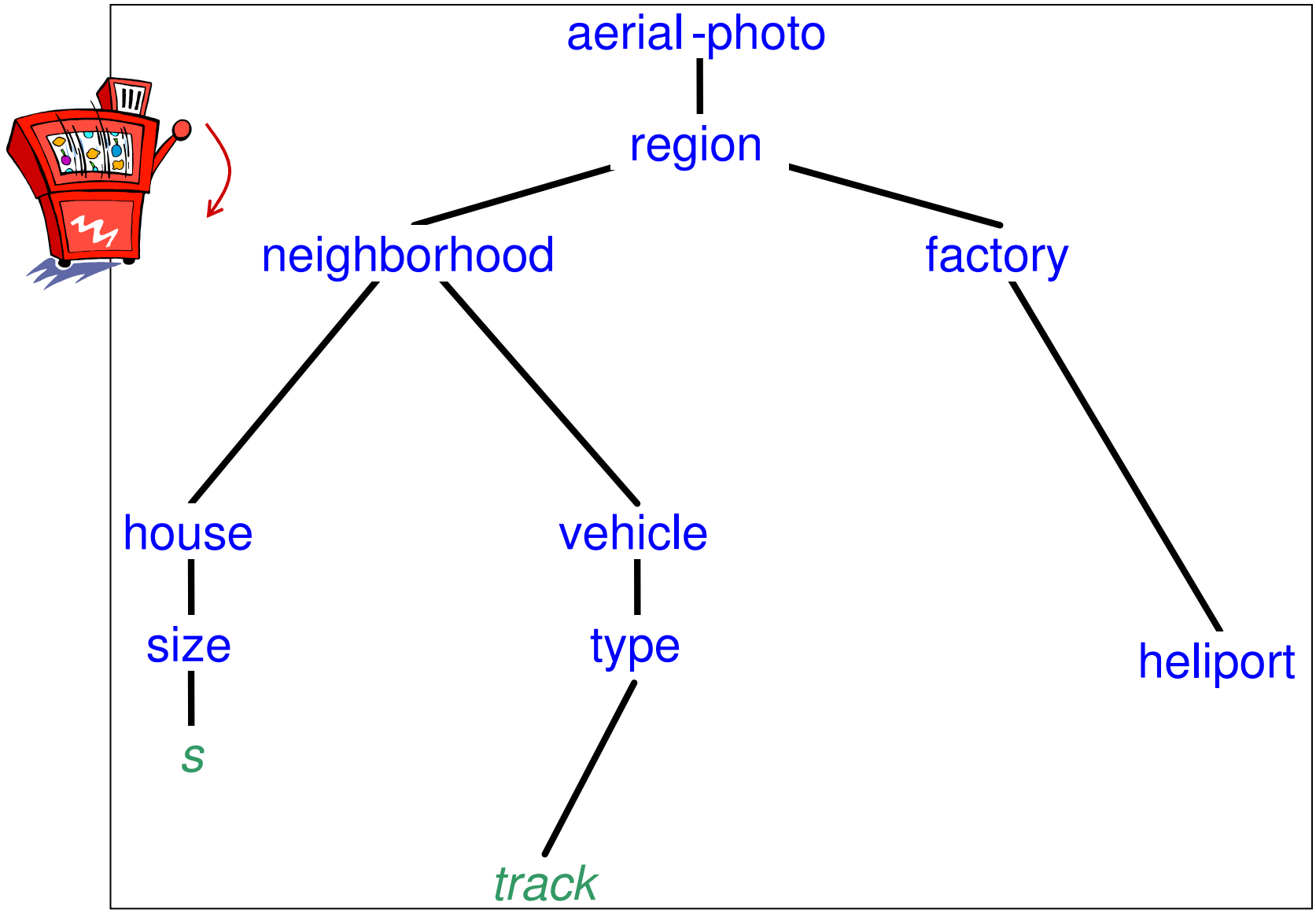
# Instance Generation: Step 2




Connect each ordinary node to its closest ancestor

Drop the distributional nodes

# The Result: An Ordinary Document



# *Talk Overview*

1. Introduction

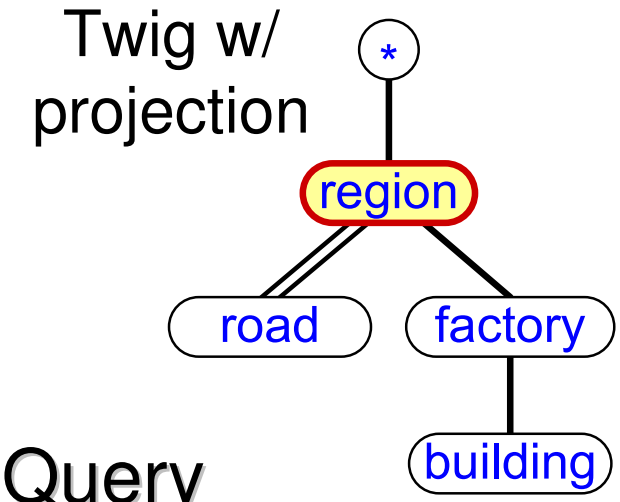
2. Twig Queries over Probabilistic XML
   - XML and Twig Queries
   - Probabilistic XML
   - Querying Probabilistic XML (Complete Semantics)

3. Query Evaluation (Complete Semantics)

4. Finding Maximal Matches

5. Conclusion, Related and Future Work

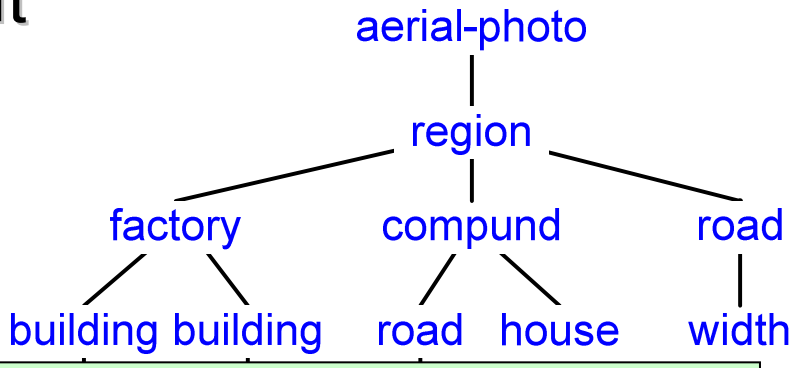
# Querying Probabilistic XML




Users pose an ordinary query

That is, of the type that is applied to non-probabilistic documents

Probabilistic XML document




... but the document is probabilistic

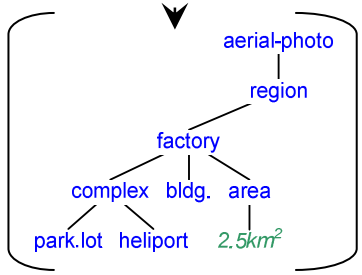
# The Probability of an Answer

When querying probabilistic data,

Each answer has a probability (certainty)

$$\Pr(A) = \Pr\left(\begin{array}{l} A \text{ is obtained by applying } Q \\ \text{to a random document of } P \end{array}\right)$$

$$\Pr\left( A \in Q \right)$$



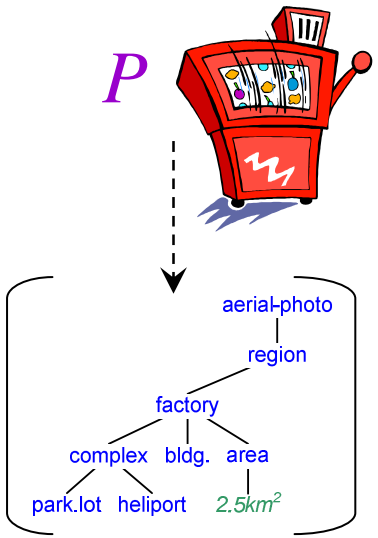
# The Prob. of Satisfying a Boolean Query

When querying probabilistic data,

Each answer has a probability (certainty)

If $B$ is a Boolean pattern, we have interest in:

$$\Pr(\text{There is a match of } B \text{ in a random document of } P)$$

$$\Pr\left[ Q\left( \text{  } \right) = \text{true} \right]$$

# *Talk Overview*

1. Introduction

2. Twig Queries over Probabilistic XML
   - XML and Twig Queries
   - Probabilistic XML
   - Querying Probabilistic XML (Complete Semantics)

3. Query Evaluation (Complete Semantics)

4. Finding Maximal Matches

5. Conclusion, Related and Future Work

# Computational Problems

Non-Boolean Queries:

| | |
|---|---|
| Input: | A prob. document $P$, a non-Boolean twig query $Q$, a threshold $p \geq 0$ |
| Goal: | Find all answers $A$, s.t. $\Pr(A \in Q(P)) \geq p$ |

Boolean Queries:

| | |
|---|---|
| Input: | A prob. document $P$, a Boolean twig query $B$ |
| Goal: | Compute $\Pr(B(P)=\text{true})$ |

# *From Regular to Boolean Queries*

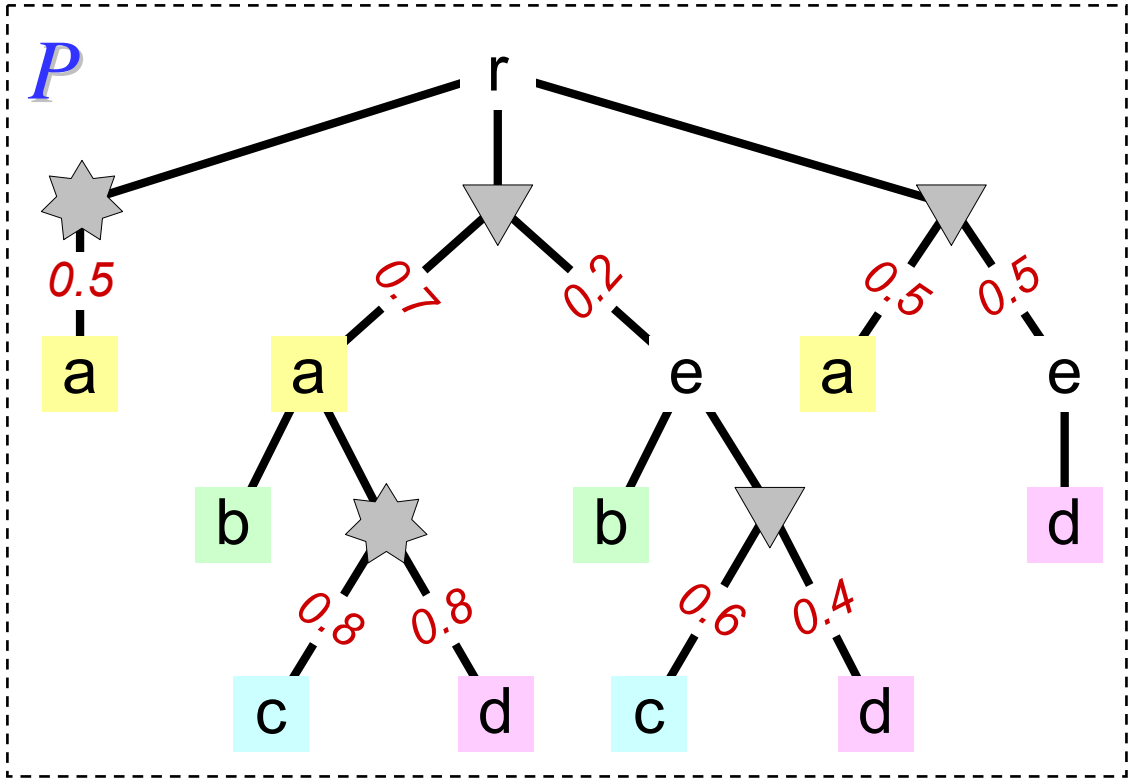
We apply a standard reduction from regular queries (that generate mappings) to Boolean ones:

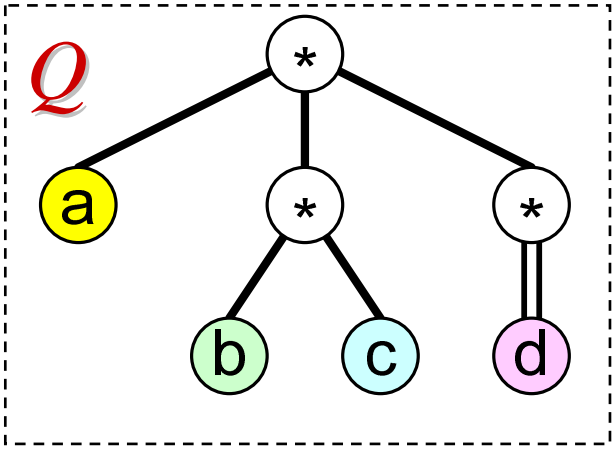
1. Compute the answers as if the document is ordinary (i.e., ignore the distributional nodes)
2. Compute the probability of each answer

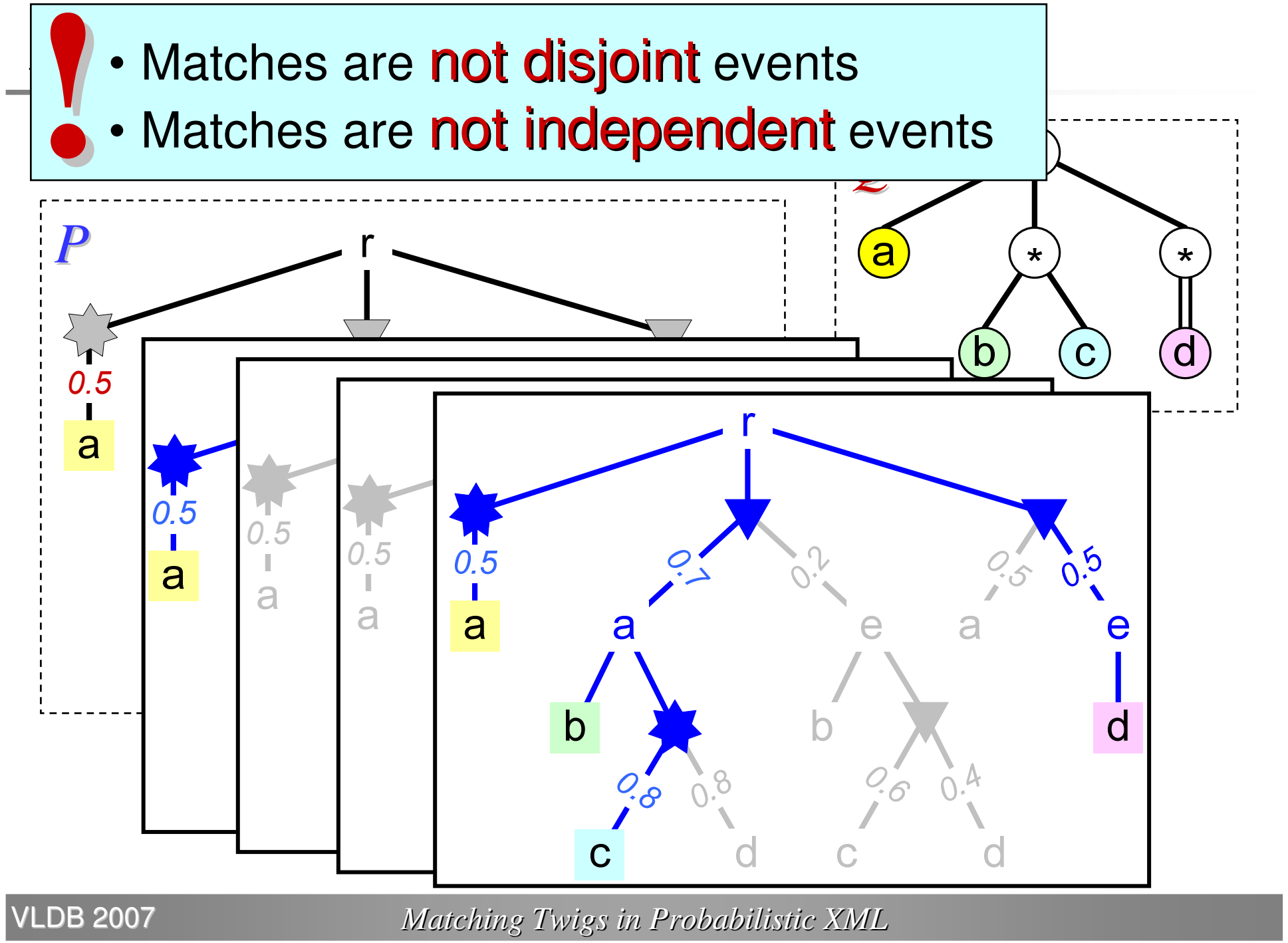
Step 2 is done by evaluating a **Boolean query**
That is, computing the probability of a match

Next, we consider the evaluation of Boolean queries

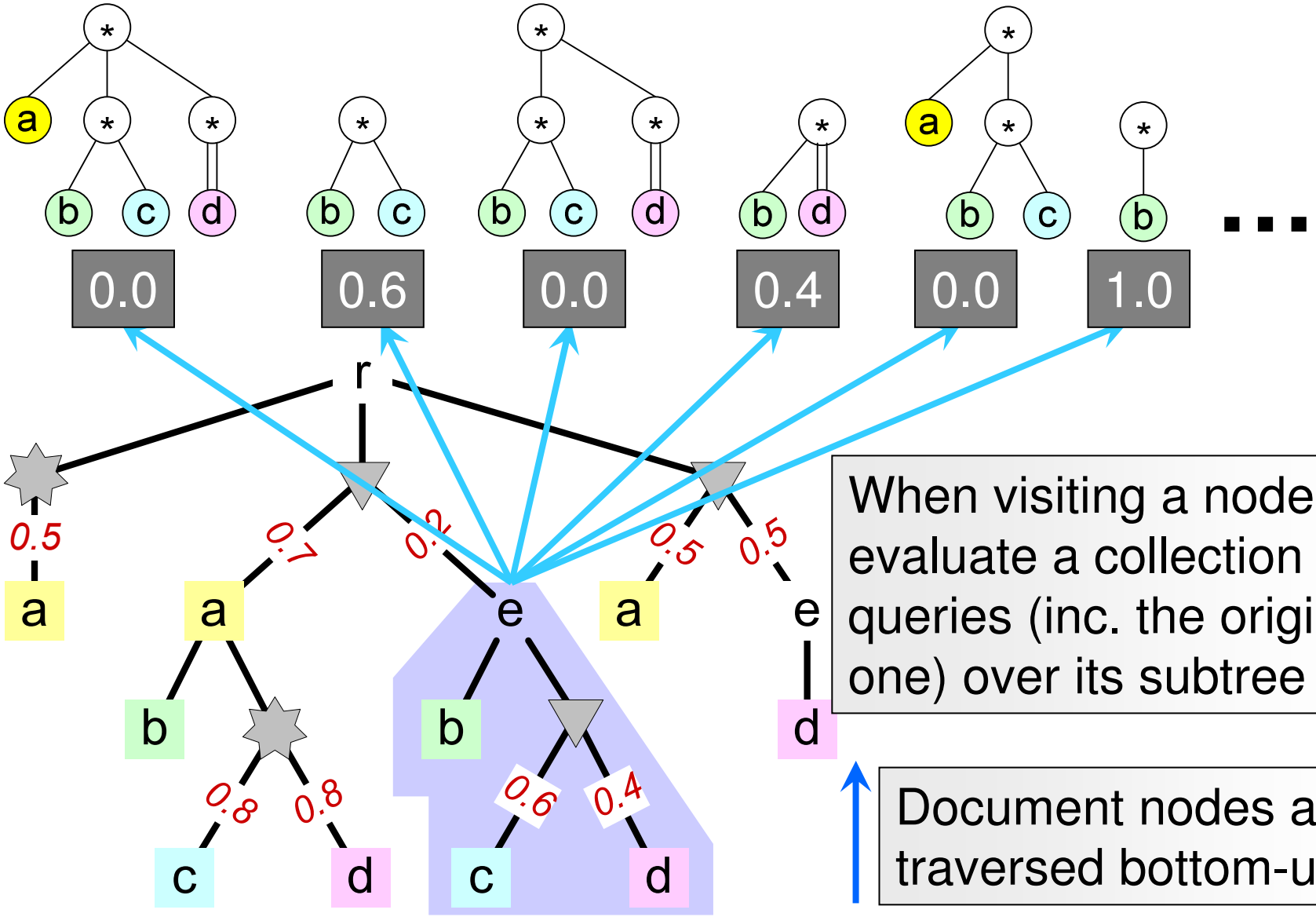
# An Example



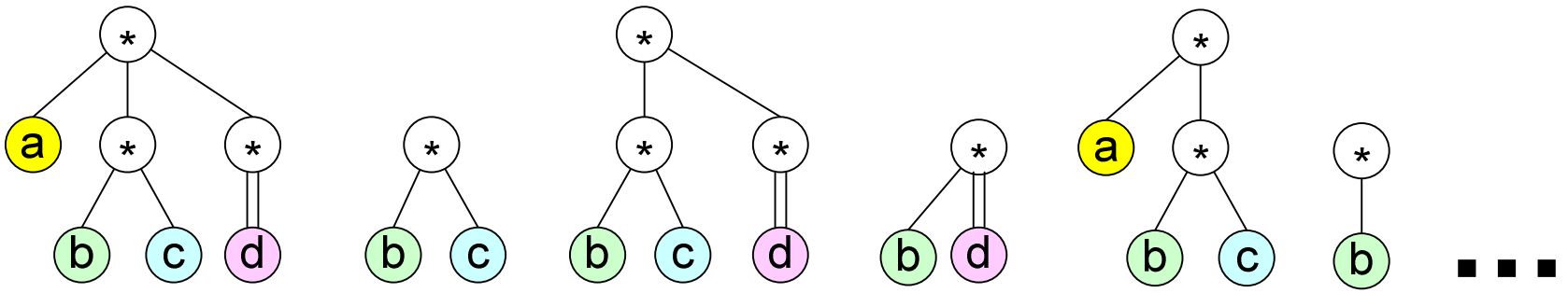




• Matches are not disjoint events
• Matches are not independent events
P
r
0.5
a
*
b
c
d
0.5
0.5
0.5
0.5
0.1
0.2
0.5
0.5
e
0.8
0.8
0.6
0.4

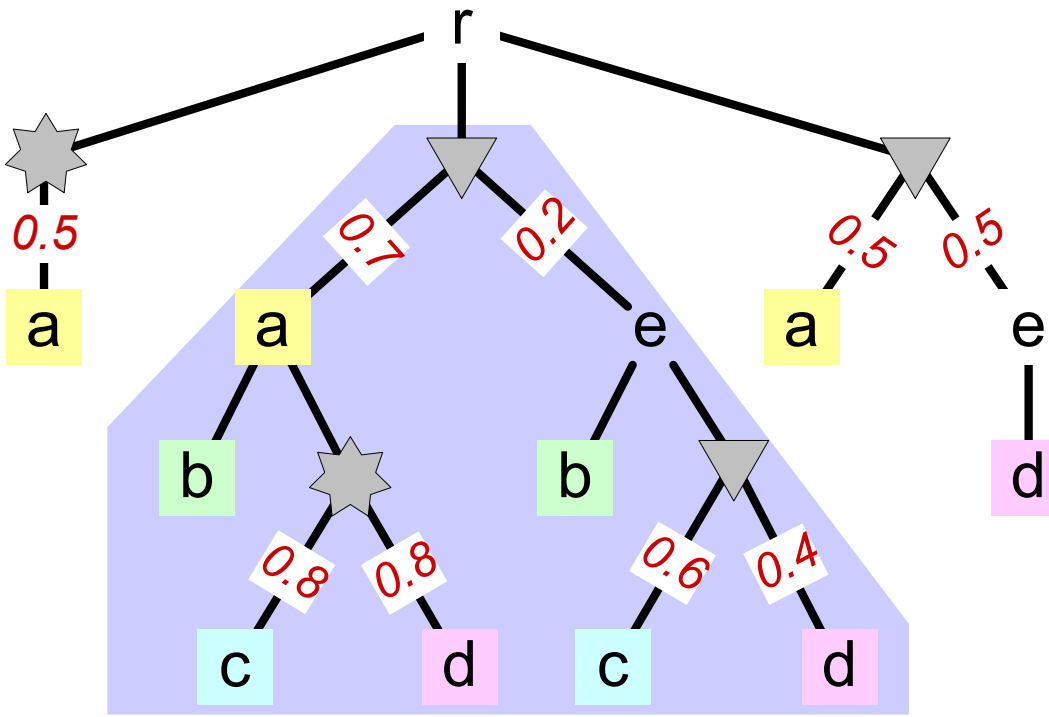
# Our Approach: Dynamic Programming



# Our Approach: Dynamic Programming



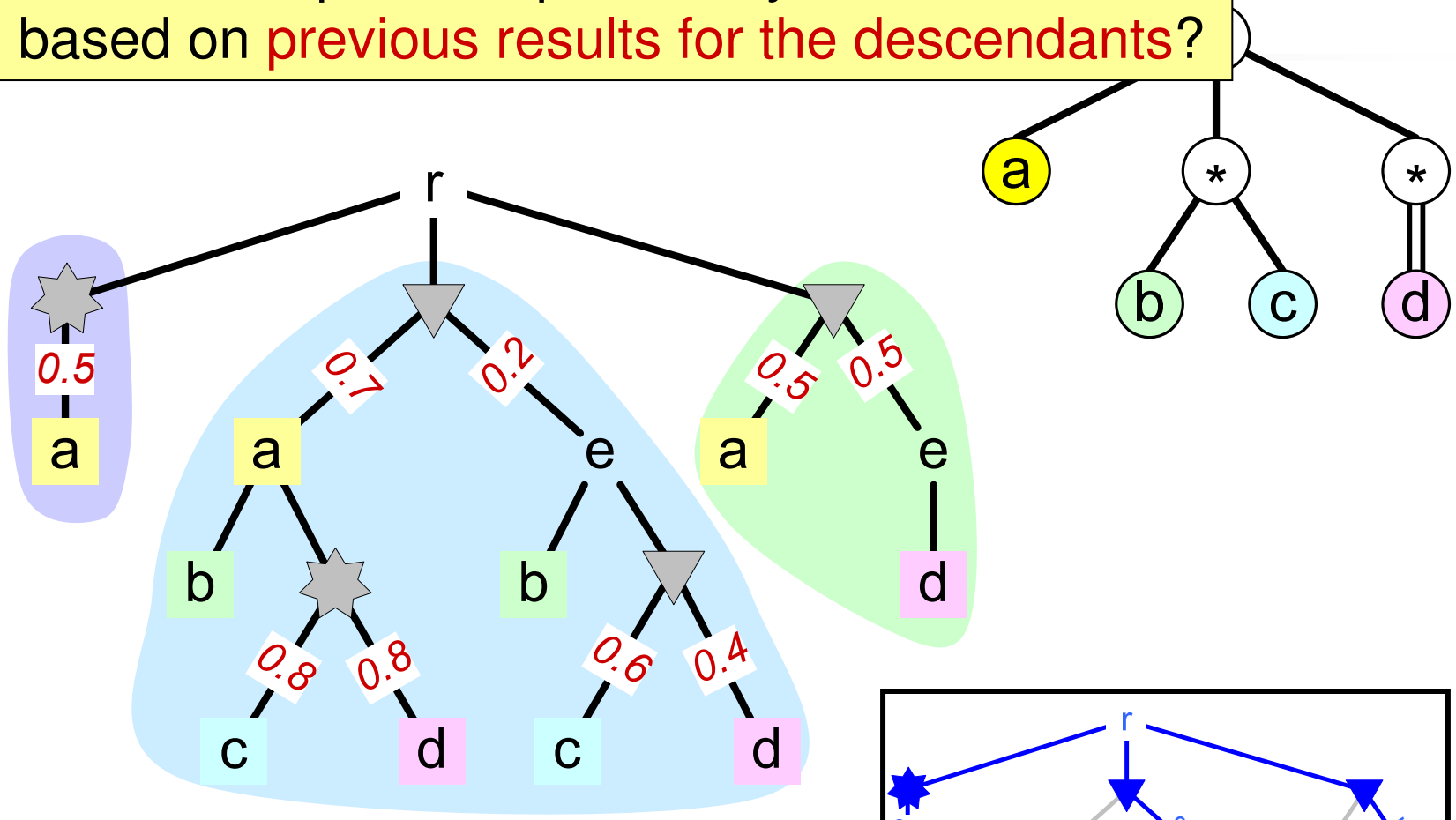

Special treatment if the visited node is distributional

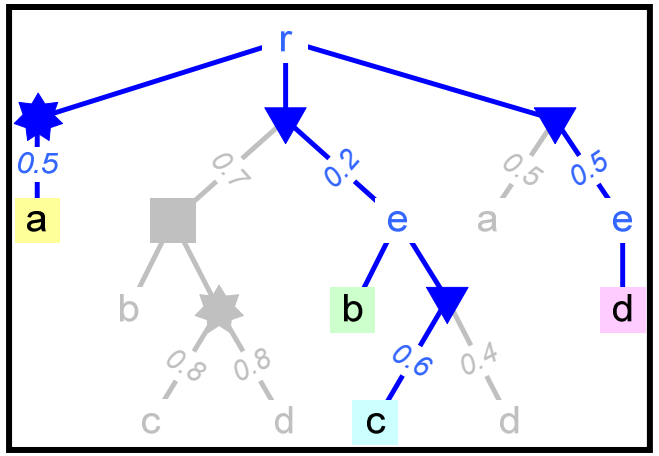



When visiting a node, evaluate a collection of queries (inc. the original) over its subtree

Document nodes are traversed bottom-up

How can we compute the probability that there is a match, based on previous results for the descendants?

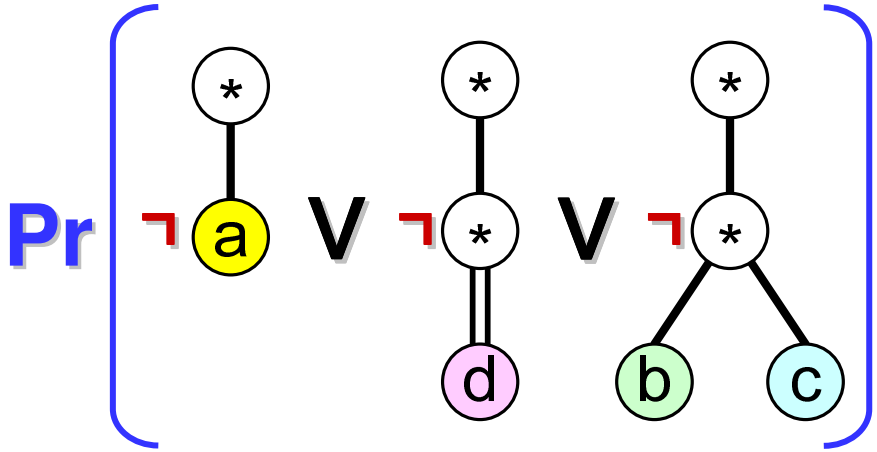



**Problem:** Each specific match can involve several different children

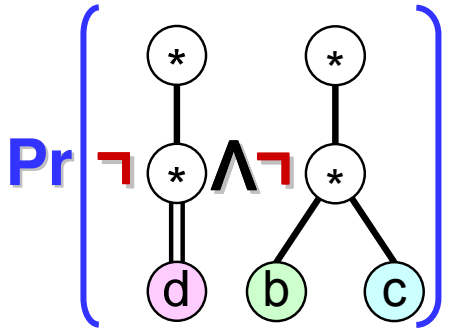
# *From Twig to Negated Branches*

Next: How to compute this value

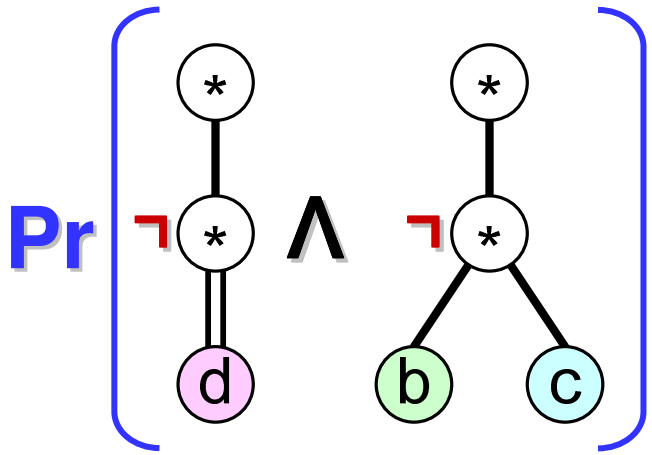


# *From a Disjunction to Conjunctions*

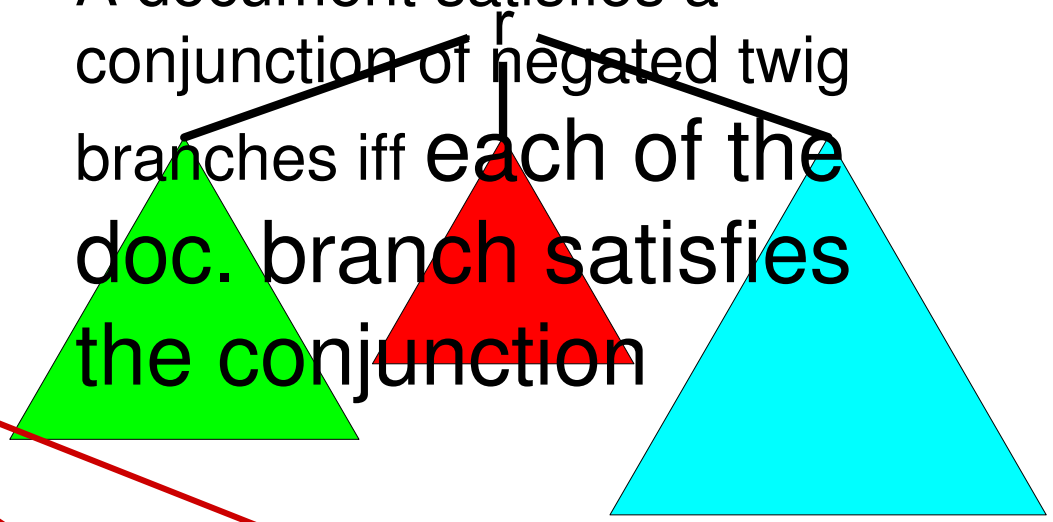
Next: How to compute this value



# *From a Document to Branches*

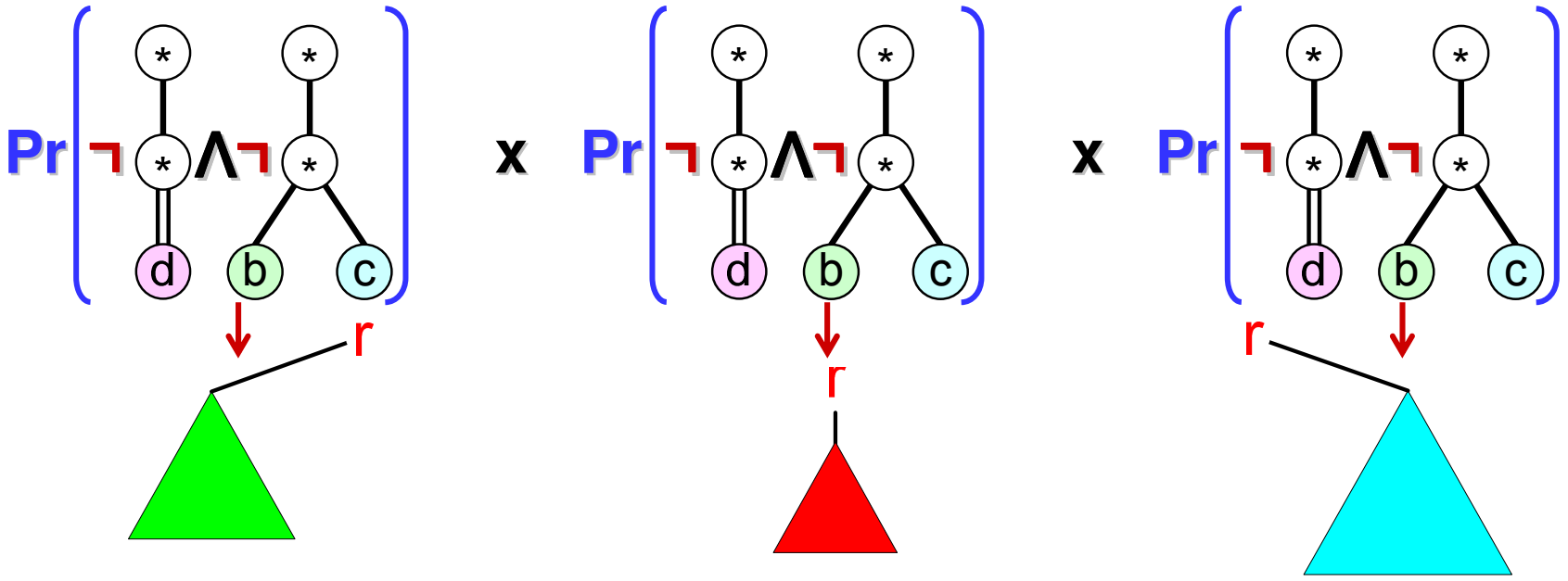



A document satisfies a conjunction of negated twig branches iff each of the doc. branch satisfies the conjunction

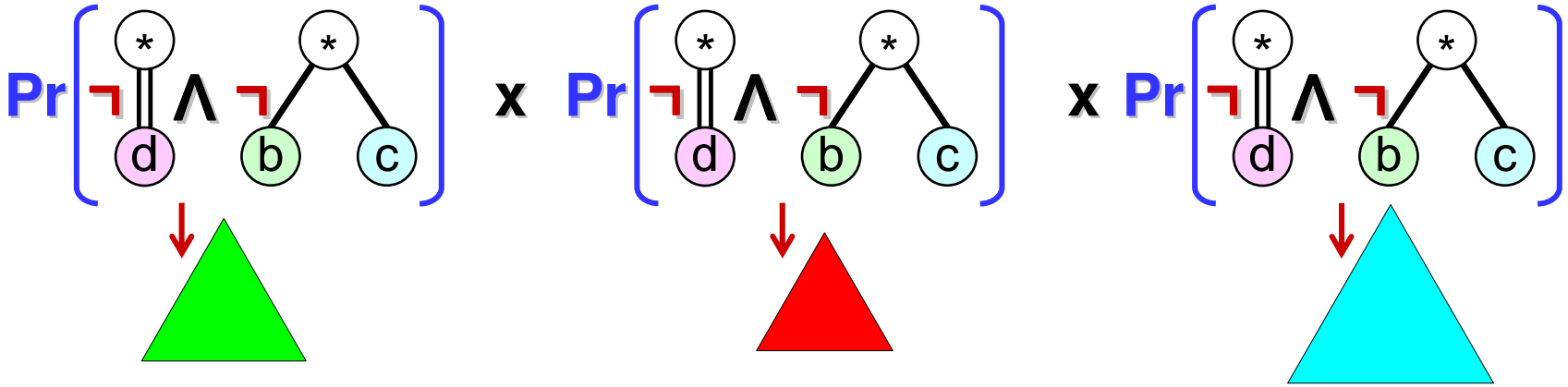



**Good news:** Document branches are **independent**!

# *Using Previous Computations on Children*



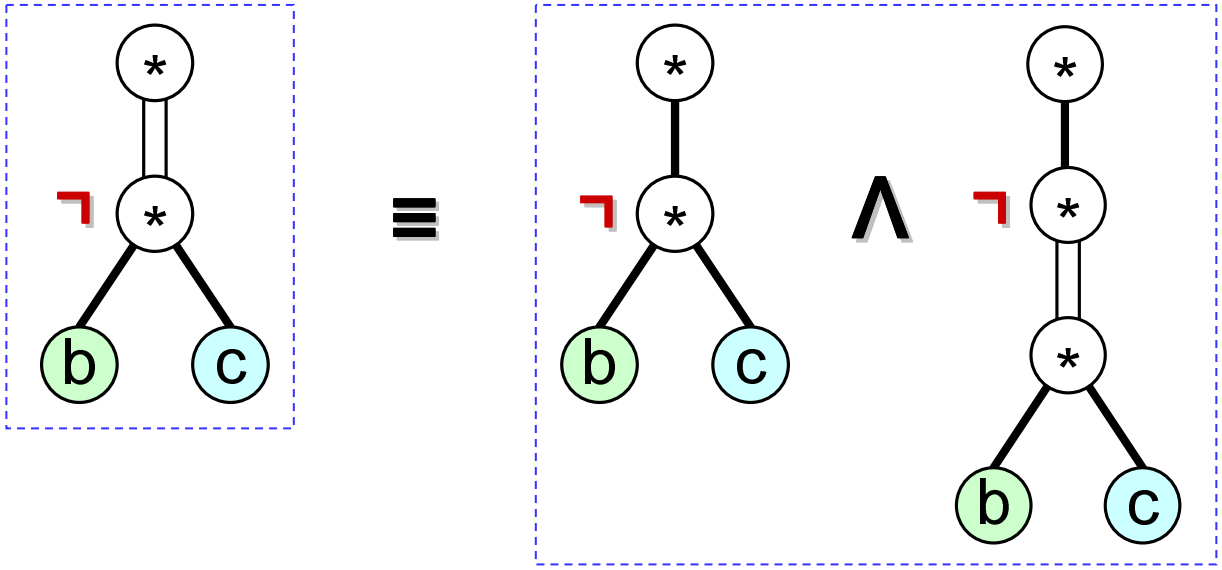

Cut the roots from both twig and doc. branches:



# Descendant Edges

- In the computation we described, we assumed that the root has only child edges; it would not work otherwise!
- What about descendant edges?

The corresponding twig branches are replaced:



# *Missing Details*

- Creating the list of twigs that are evaluated over the subtree rooted at each visited node
- Different evaluation methods, depending on the type of the visited node
  - Ordinary node (sketched in the previous slides)
  - Distributional node
    - Independent distribution
    - Mutually-exclusive distribution
- Dealing with node predicates of the twig

All the details of the algorithm are in the paper

# Efficiency

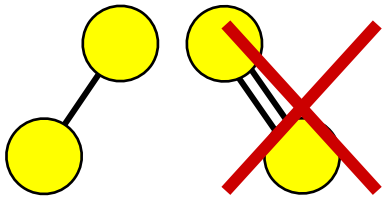
The algorithm computes Pr($B(P)$=true) in time

$$O(c^{|B|} \cdot |P|)$$

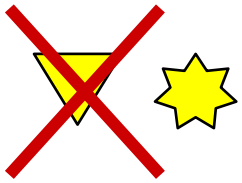
Is there an efficient algorithm under query-and-data complexity (polynomial in the query also)?

**No!** Computing Pr($B(P)$=true) is #P-complete under query & data complexity!

Even if:



No desc. edges



Only independent distributions

• • •

# *Talk Overview*

1. Introduction

2. Twig Queries over Probabilistic XML
   - XML and Twig Queries
   - Probabilistic XML
   - Querying Probabilistic XML (Complete Semantics)

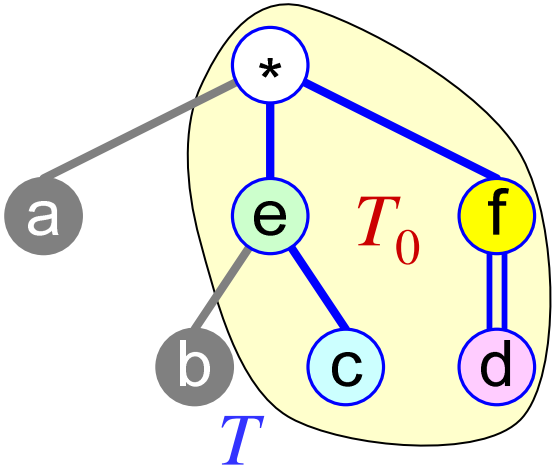
3. Query Evaluation (Complete Semantics)

4. Finding Maximal Matches

5. Conclusion, Related and Future Work

# Standard Terminology

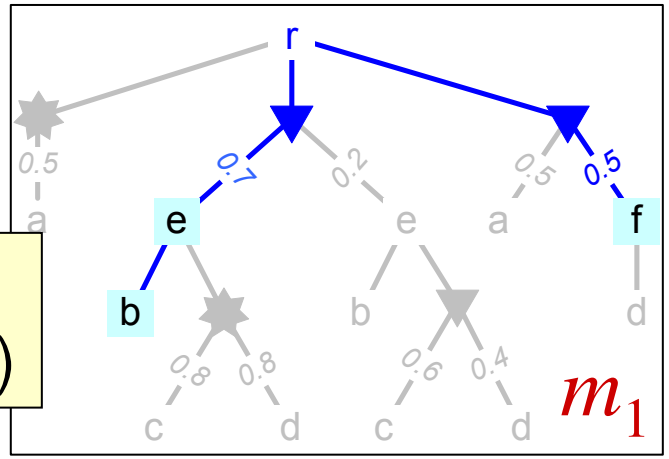
$T_0$: a subtree of twig $T$, includes the root

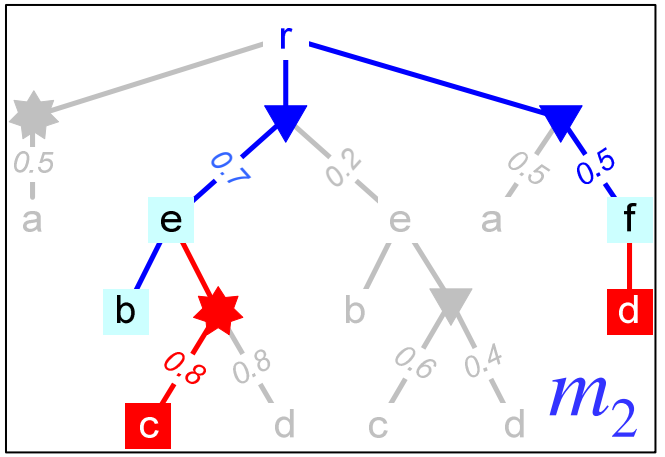



A match $m_0$ of $T_0$ is a *partial match* of $T$

$m_2$ **subsumes** $m_1$ if $m_2$ includes the mappings of $m_1$

That is, $m_1$=$m_2$ over domain($m_1$)



# *Maximal Answer: Definition*

$m$ is a *maximal* answer:

**Ordinary Data:**

$\nexists\ m_0$, such that $m_0 \neq m$ and $m_0$ subsumes $m$

**Probabilistic Data:**

- $\Pr(m) \geq$ threshold
- $\forall\ m_0$, if $m_0 \neq m$ and $m_0$ subsumes $m$, then $\Pr(m_0) <$ threshold

In other words, $m$ is maximal among the partial answers with a sufficient probability

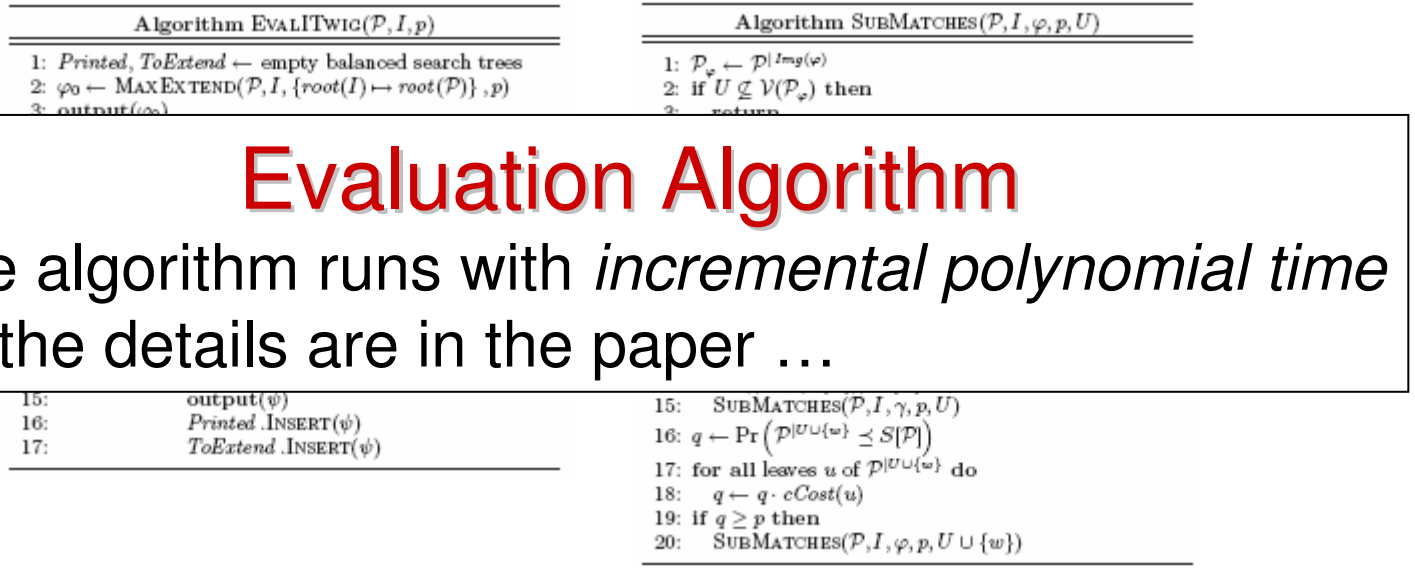
# The Computational Problem

| | |
|---|---|
| Input: | A probabilistic document $P$, a twig pattern $T$, a threshold $p \geq 0$ |
| Goal: | Find all maximal matches of $T$ in $P$ w.r.t. $p$ |

# *Complexity of Finding Maximal Matches*

- It is trivial to show that maximal matches can be found efficiently under data complexity
- Unlike the case of complete matches (NP-complete),

Maximal matches can be computed efficiently under query-and-data complexity



Algorithm EvalITwig($\mathcal{P}, I, p$)

1: *Printed*, *ToExtend* ← empty balanced search trees
2: $\varphi_0 \leftarrow$ MaxExtend($\mathcal{P}, I, \{root(I) \mapsto root(\mathcal{P})\}, p$)
3: output($\varphi_0$)
15: output($\psi$)
16: *Printed*.Insert($\psi$)
17: *ToExtend*.Insert($\psi$)

Algorithm SubMatches($\mathcal{P}, I, \varphi, p, U$)

1: $\mathcal{P}_\varphi \leftarrow \mathcal{P}^{|Img(\varphi)}$
2: if $U \not\subseteq \mathcal{V}(\mathcal{P}_\varphi)$ then
3: return
15: SubMatches($\mathcal{P}, I, \gamma, p, U$)
16: $q \leftarrow \Pr\left(\mathcal{P}^{|U \cup \{w\}} \preceq S|\mathcal{P}|\right)$
17: for all leaves $u$ of $\mathcal{P}^{|U \cup \{w\}}$ do
18: $q \leftarrow q \cdot cCost(u)$
19: if $q \geq p$ then
20: SubMatches($\mathcal{P}, I, \varphi, p, U \cup \{w\}$)

## Evaluation Algorithm

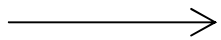
- The algorithm runs with *incremental polynomial time*
- All the details are in the paper …

# *Talk Overview*

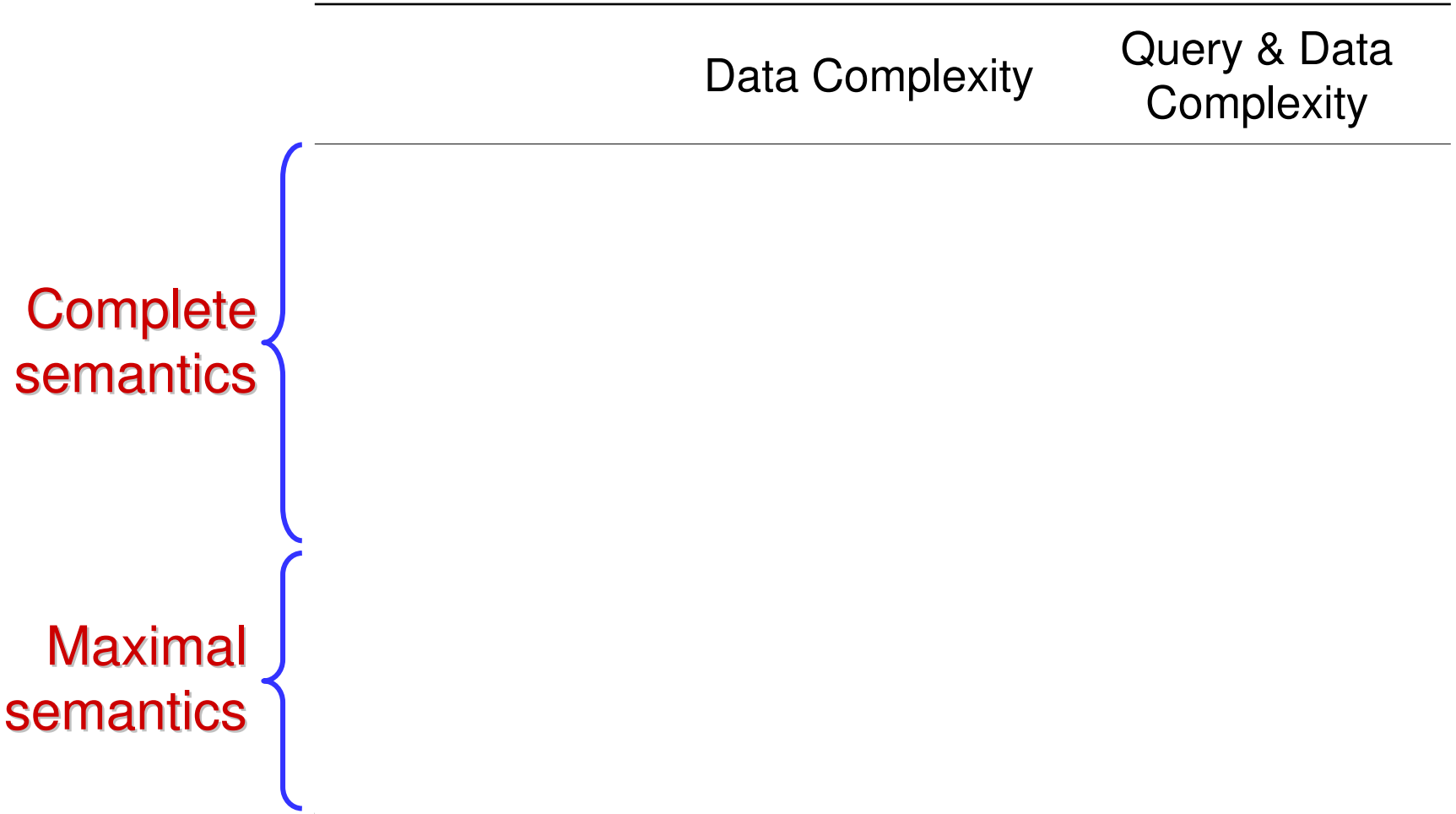
1. Introduction
2. Twig Queries over Probabilistic XML
   - XML and Twig Queries
   - Probabilistic XML
   - Querying Probabilistic XML (Complete Semantics)
3. Query Evaluation (Complete Semantics)
4. Finding Maximal Matches
5. Conclusion, Related and Future Work

# Paper Summary

- Query evaluation over probabilistic XML is investigated
  - Known data model
  - Twig patterns (node predicates, child & desc. edges)
  - Complete & maximal semantics, projection
- Evaluation algorithm for Boolean queries
  - Also used for evaluating queries with projection
  - Efficient under data complexity
- An algorithm for finding the maximal matches
  - Efficient under query-and-data complexity
- Analysis of the complexity of querying prob. XML



# Complexity Results



The complexity results in the different prob. XML models are a part of our ongoing research

**Fuzzy trees** [Abiteboul & Senellart, 2006]

Query Evaluation: ***#P-Complete***

*Our model*

**ProTDB** [Nierman and Jagadish, 2002]

Query Evaluation: ***Tractable***

**Simple prob. trees** [Abiteboul & Senellart, 2006]

Query Evaluation: ***Tractable***

**PXML** [Hung, Getoor & Subrahmanianm, 2003]

Query Evaluation:

Tree docs.: ***Tractable***, DAG docs.: ***#P-hard***

**Query evaluation:** Complete semantics w/ projection

# *Ongoing and Future Work*

- Implementing a system for representing and querying probabilistic XML
- Optimization of the proposed algorithms
  - We already obtained significant improvements, both experimentally and analytically
- Extending the expressiveness of the model of probabilistic XML
  - New types of distributional nodes
  - **Ongoing work:** A combination of ProTDB [Nierman and Jagadish, 2002] and PXML [Hung, Getoor & Subrahmanianm, 2003]
- Combining incompleteness and projection

# Thank you!

Questions?