

Welcome to JSON World

JSON is used everywhere nowadays.



Pros:
High degree of freedom in modeling data

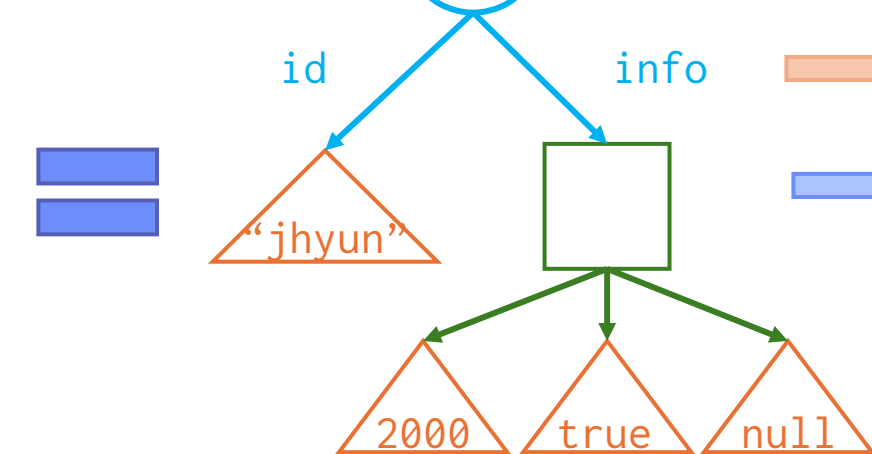
Cons:
JSON documents often not accompanied by a JSON schema

JSON document: a string derived by the JSON grammar

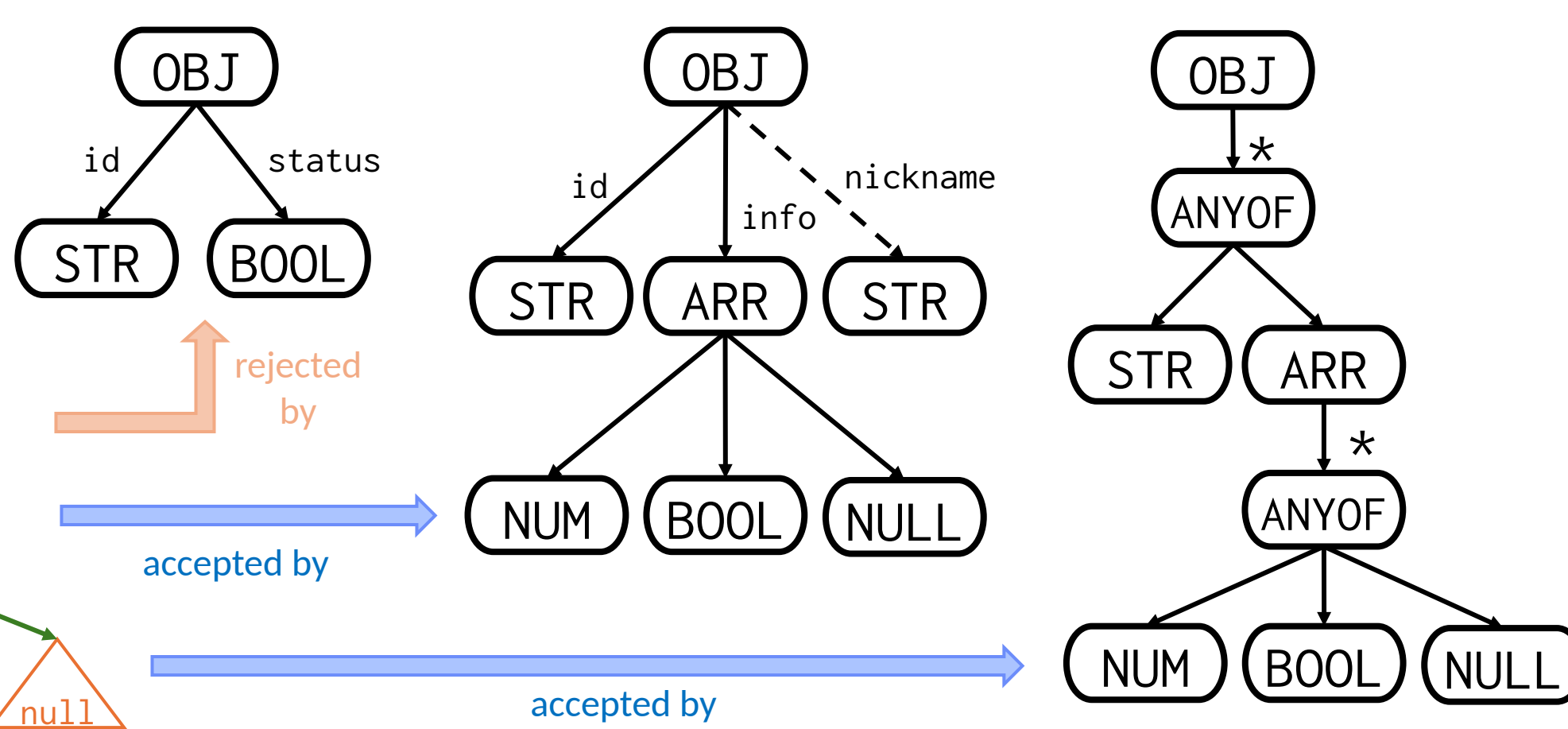
- JSON data format has 6 data types.
- JSON is recursively defined.

A JSON document can be expressed in a tree format

```
{
  "id": "jhyun",
  "info": [2000, true, null]
}
```

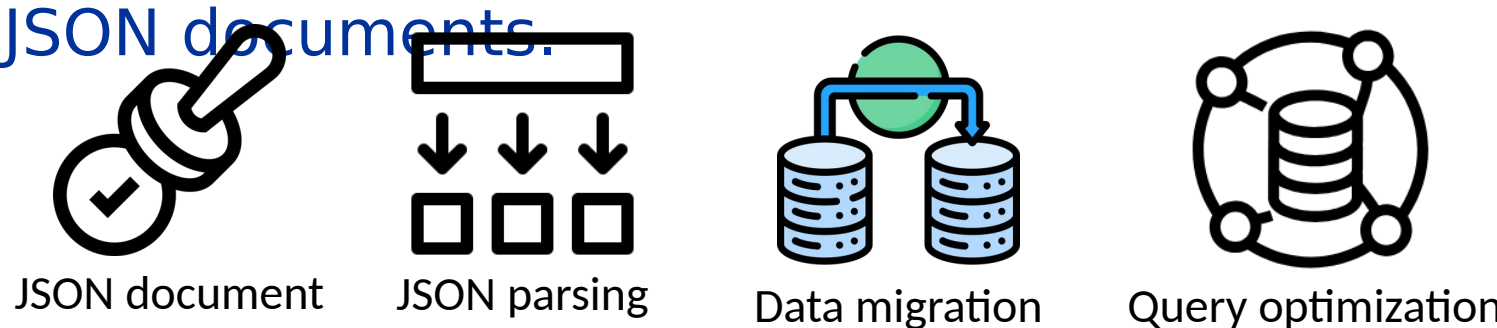


JSON & JSON Schema



JSON Schema

JSON schemas are JSON documents that constrain and explain the structure of other JSON documents.



- Imposes constraints to other JSON documents
- Either accepts or rejects a JSON document
- Is recursively defined

A JSON schema can also be expressed in a tree format

Research Goal

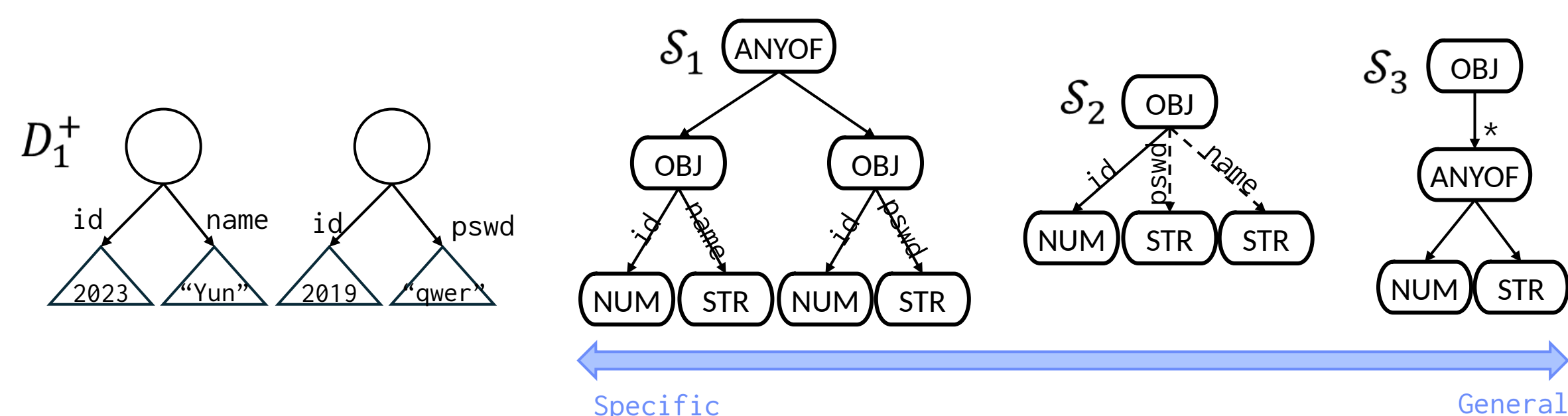
Suppose there is a ground truth schema $\mathcal{S}_G$. Given D_1^+ , derive an accurate schema $\mathcal{S}_D$ that shows the maximum F1-score calculated with D_2^+ , D^- .

- D_1^+ , D_2^+ : sets of JSON documents that are accepted by $\mathcal{S}_G$.
- D^- : a set of JSON documents that are not accepted by $\mathcal{S}_G$.

An ideal $\mathcal{S}_D$ should accept all $j^+ \in D_2^+$ and reject all $j^- \in D^-$.

Challenges

- Innumerable candidates of schemas that can be derived**
 - Schemas of various generalities can be candidates for $\mathcal{S}_D$.
- Ambiguity in picking the best schema among the candidates**
 - We do not have access to D_2^+ and D^- while deriving schemas.



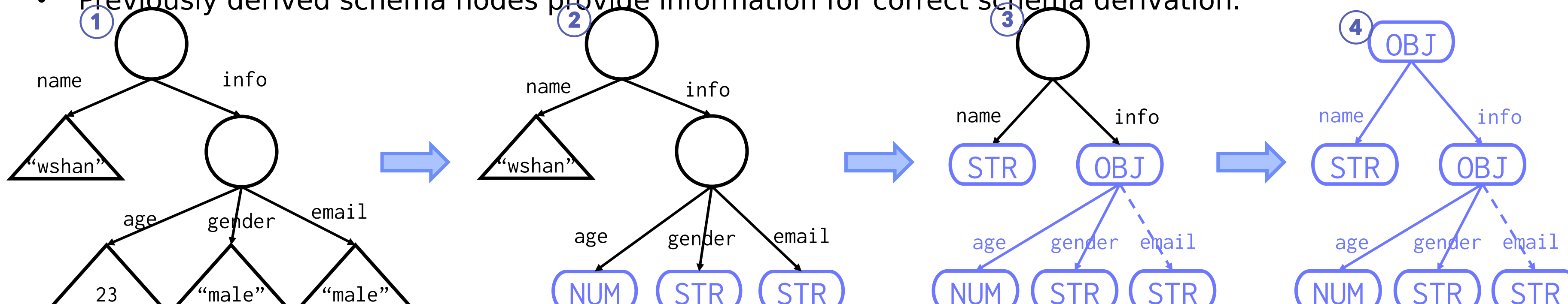
1. Bottom-Up Schema Generation

Existing works derive schema in a top (root) - down (leaf) manner.

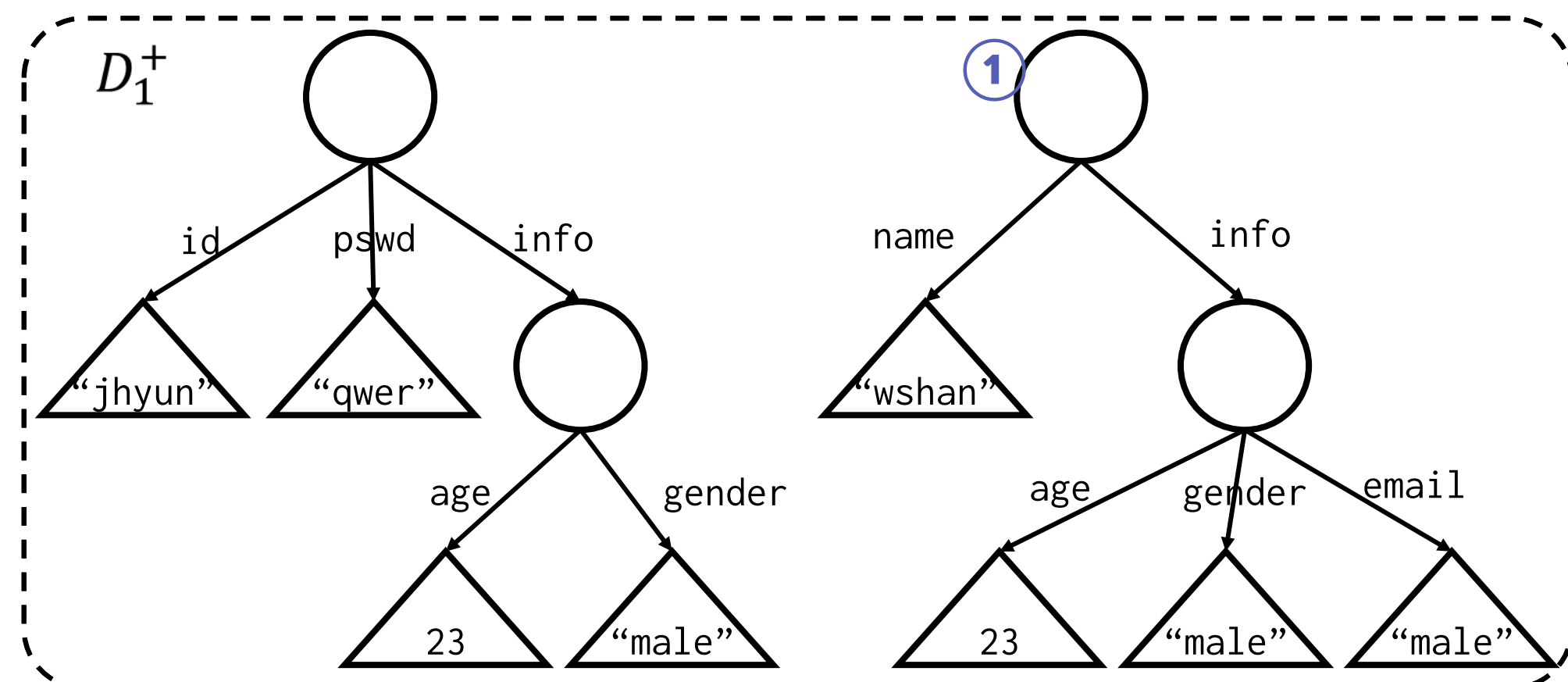
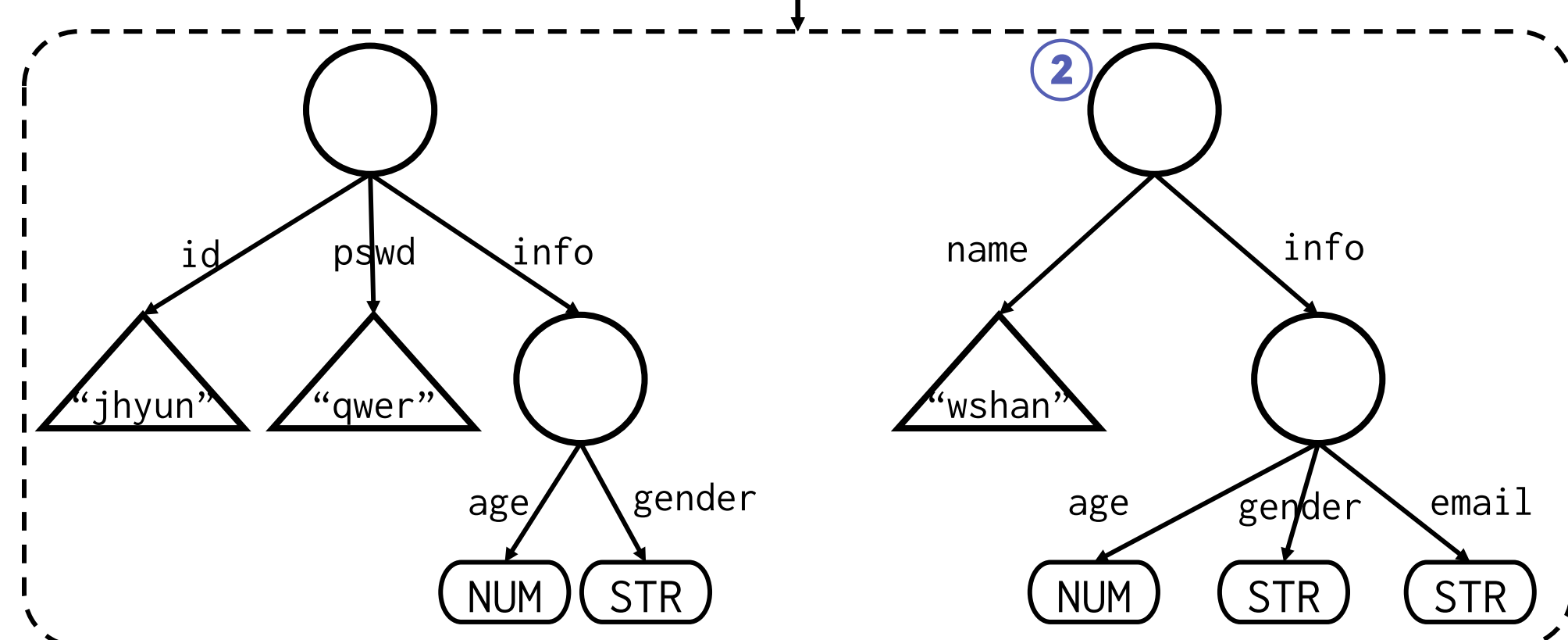
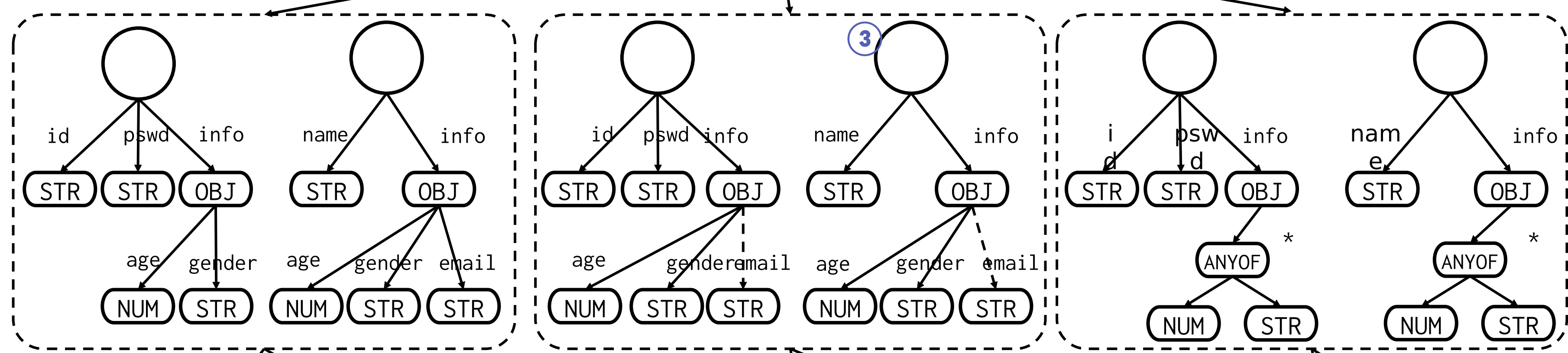
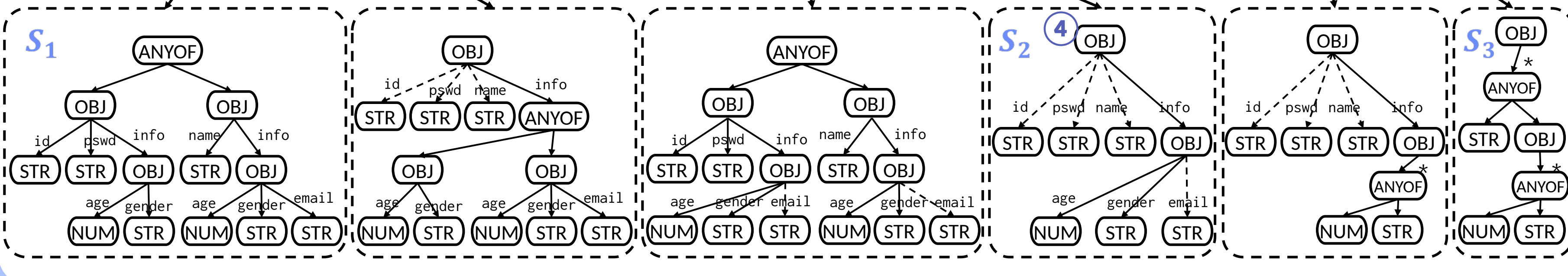
- Information may be insufficient to correctly derive a schema as schemas of children instances are not derived

We propose deriving schema in a bottom (leaf) - up (root) manner.

- Previously derived schema nodes provide information for correct schema derivation.



Global Example



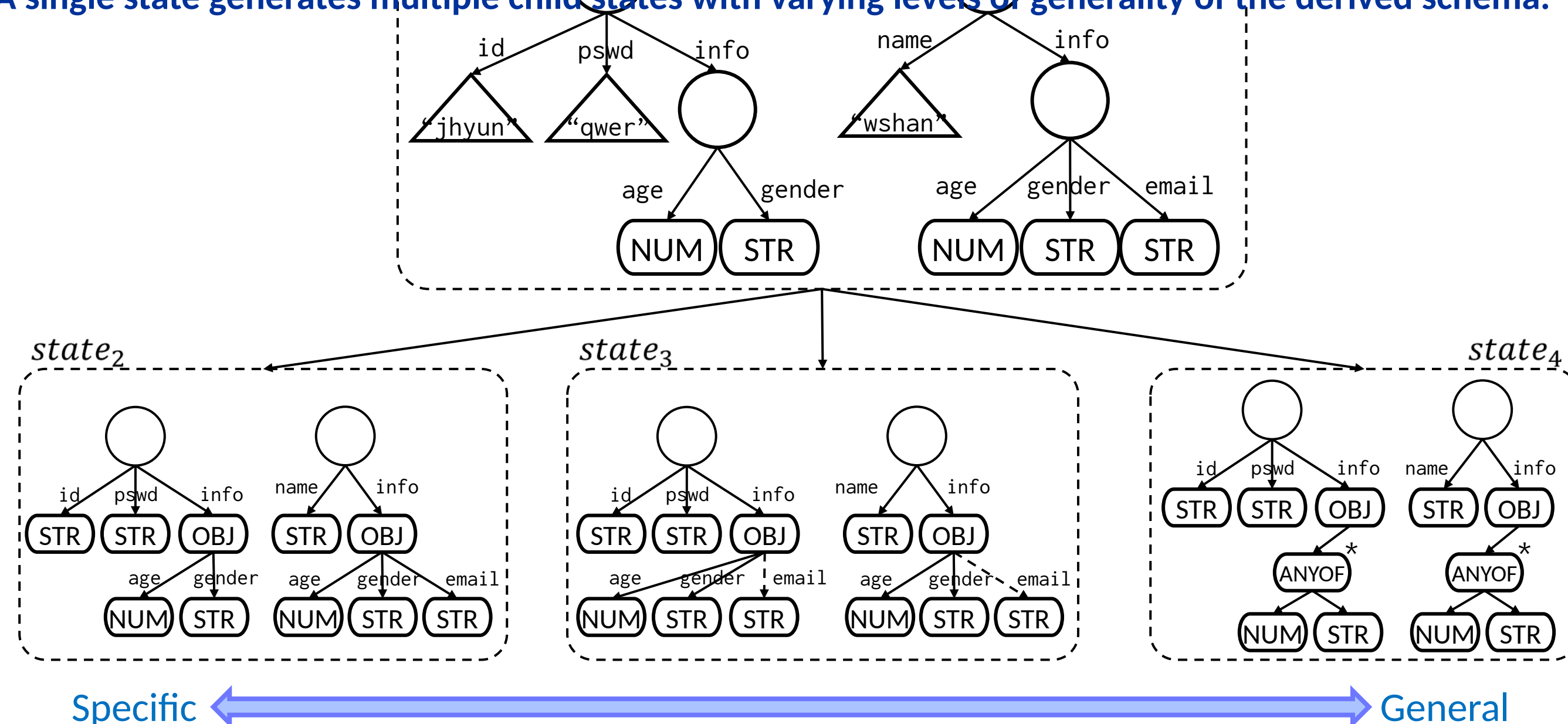
2. Schema Candidate Exploration

In most cases, it may be ambiguous to pick one set of schemas to derive.

ReCG explores multiple schema derivation candidates rather than settling on just one.

- State: a bag of partially derived schemas
- Transition: deriving schema nodes of an upper level

A single state generates multiple child states with varying levels of generality of the derived schema.



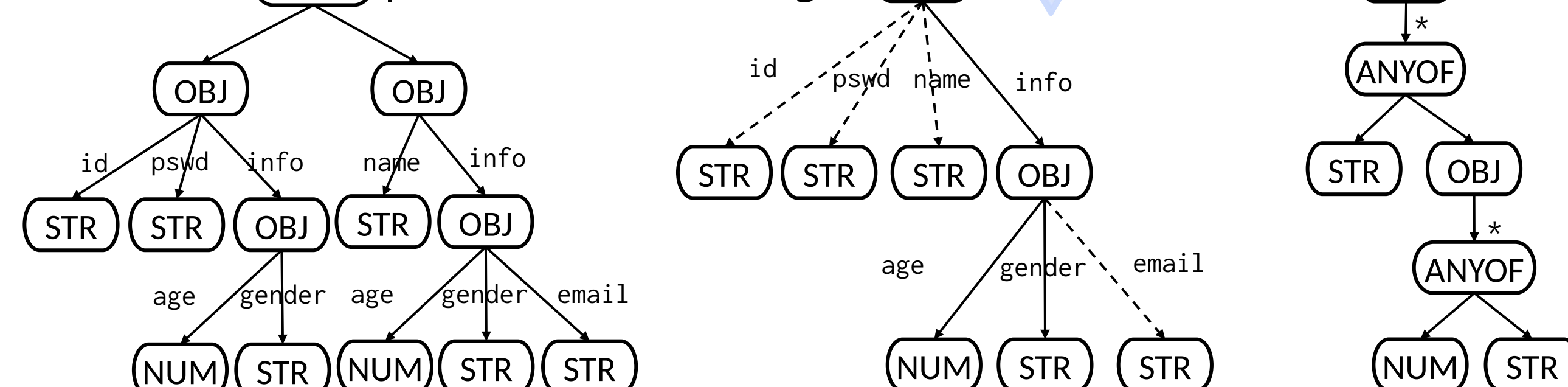
3. Minimum Description Length Principle

Motivation: A metric that correctly measure the "goodness" of a schema is necessary.

Idea: A "good" schema will neither be too general nor too specific.

We propose MDL Cost, which is the sum of two costs measuring each aspect.

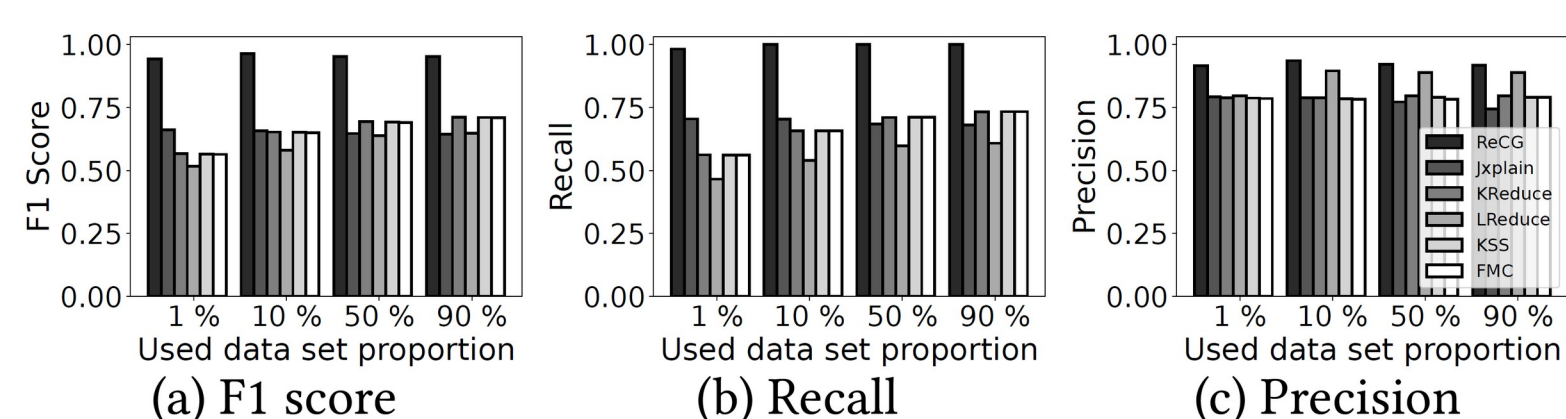
- A schema is penalized for being too general (small in size)
- A schema is penalized for being too specific (large in size)



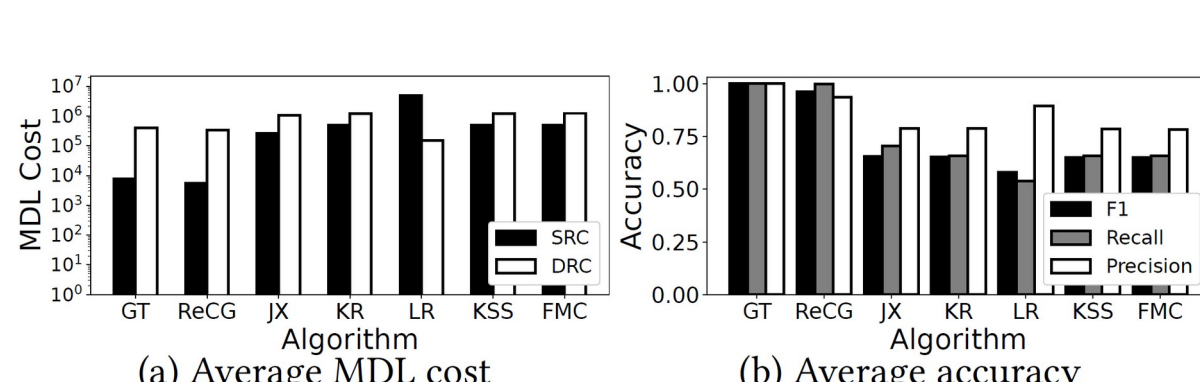
Generality	↓	-	↑
Specificity	↑	-	↓

Evaluation

Accuracy Comparison



MDL Cost Analysis



Algorithm Execution Time

Used dataset proportion	Avg runtime	Stddev	ReCG	Jxplain	KReduce	LReduce
10%	1489.62 ms	116.30 ms	3130.67 ms	206.18 ms	883.99 ms	1030.81 ms
50%	6571.55 ms	547.28 ms	10128.23 ms	429.52 ms	2689.82 ms	4892.20 ms
100%	12571.91 ms	448.63 ms	26479.15 ms	790.02 ms	5006.07 ms	12603.57 ms

- ReCG outperforms all competitors in F1 score, especially shows **46% increase compared to the SOTA**.
- upto **47% higher recall** compared to SOTA
- upto **18% higher precision** compared to SOTA

- ReCG's MDL cost is **3.8x, 4.9x and 14.2x smaller** than Jxplain, KReduce, and LReduce, respectively.
- MDL costs show a **strong negative correlation (-0.77)** with F1 scores.

- ReCG's performance is positioned in the middle of KReduce and Jxplain
- ReCG **outperforms Jxplain by 1.54x to 2.11x** on average varying on dataset size
- ReCG is slower than KReduce by about 2x

Conclusion

- We present an **accurate** and **efficient** JSON schema discovery algorithm called **ReCG**
- ReCG solves the accuracy problems of existing algorithms
- We propose the novel concept of **bottom-up schema generation** to provide more information within the schema derivation process
- We solve the JSD problem as a **search problem**, while applying the **minimum description length principle** to choose the most promising schema