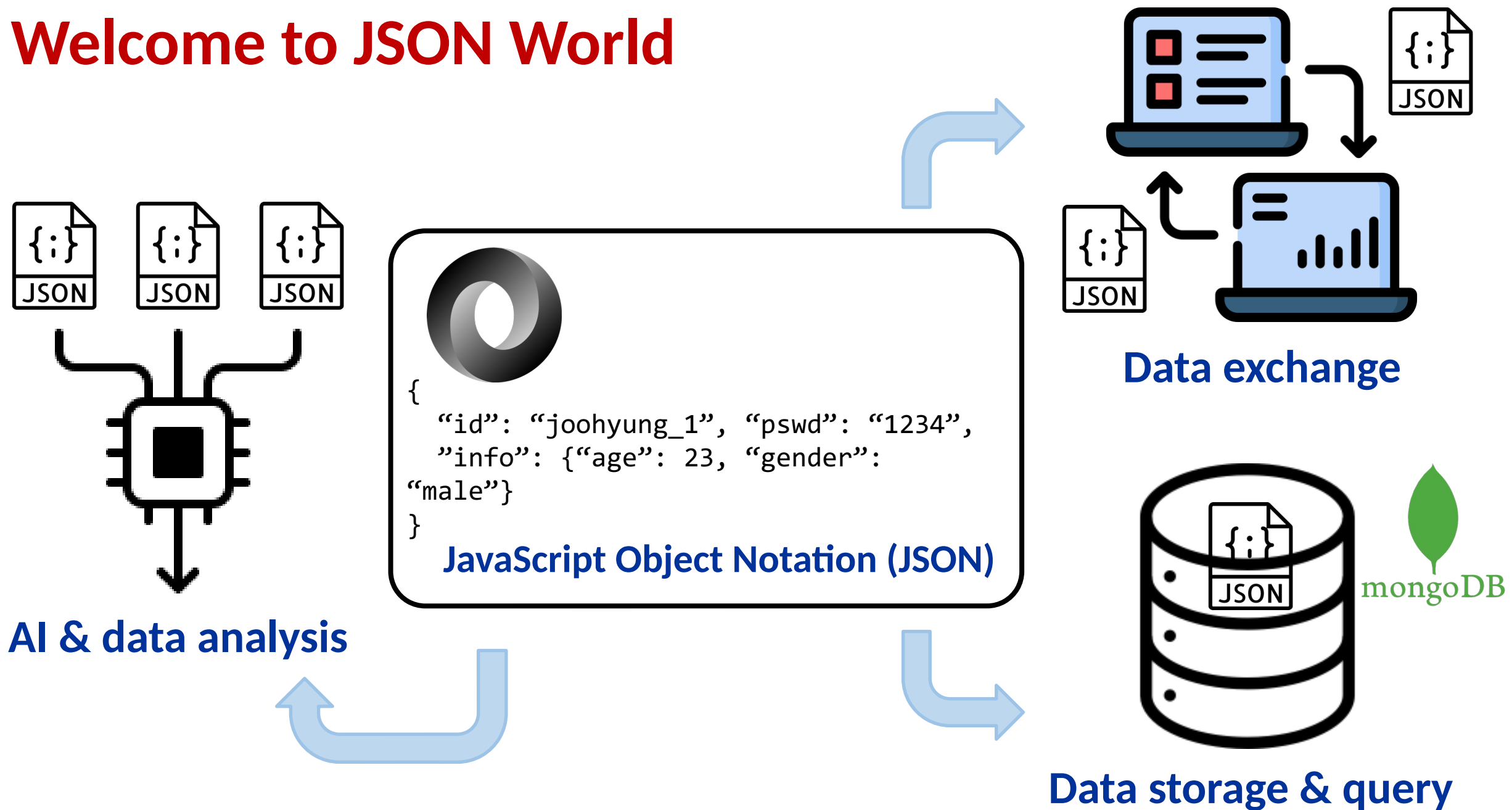


ReCG: Bottom-Up JSON Schema Discovery Using a Repetitive Cluster and Generalize Framework

August 28th, 2024

Joohyung Yun, Byungchul Tak, Wook-Shin Han
POSTECH

Welcome to JSON World



Welcome to JSON World

High degree of freedom in data modeling

```
{“id”: “joohyung_1”, “pswd”: “1234”,
  “info”: {“age”: 23, “gender”: “male”}}
```

```
{“id”: “wook-shin_2”, “pswd”: “qwer”,
  “info”: {“age”: 52, “gender”: “male”}}
```

```
{“id”: “byungchul_3”, “pswd”: “asdf”,
  “info”: {“degree”: “PhD”, “place”: “KNU”}}
```

PEOPLE

id	pswd
joohyung_1	1234
wook-shin_2	qwer
byungchul_3	asdf

INFO

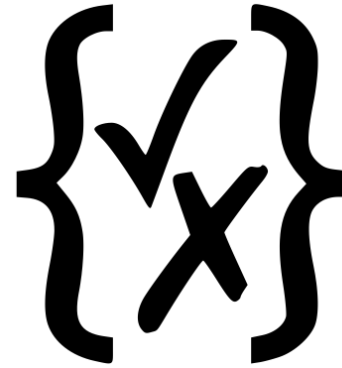
id	age	gender
joohyung_1	23	male
wook-shin_2	52	male

ANOTHER_INFO

id	degree	place
byungchul_3	PhD	KNU

Problem: An accompanying schema is often absent!

JSON Schema



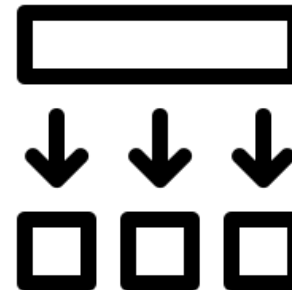
JSON Schema



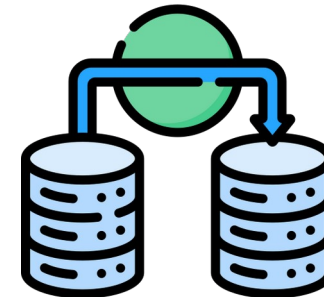
JSON document
validation



Data analytics

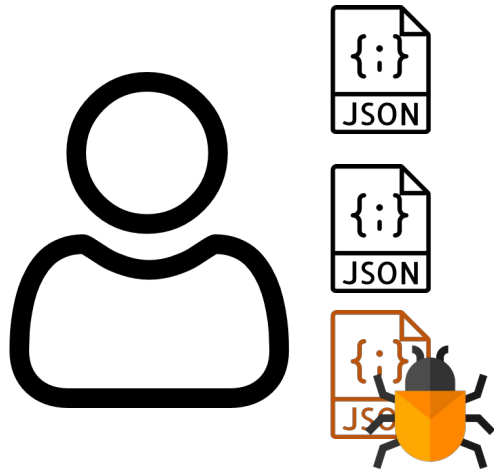
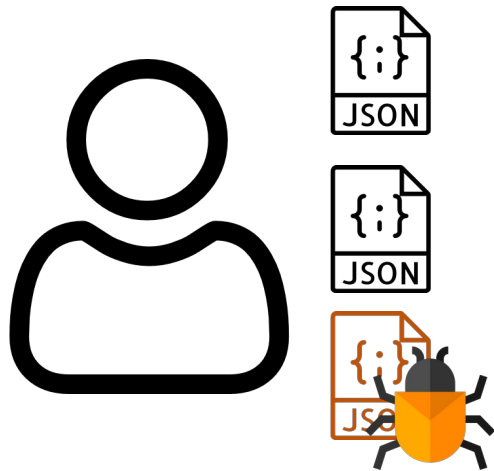


JSON parsing
acceleration

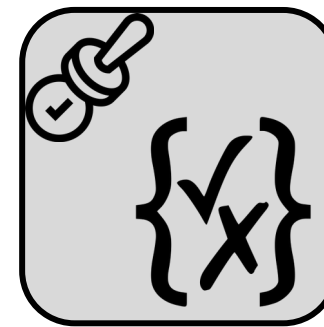


Data migration

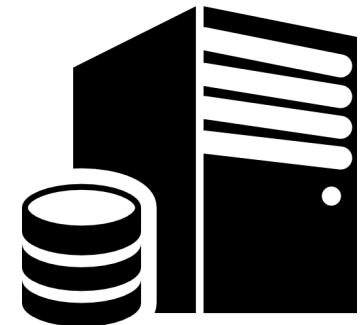
JSON Schema: JSON Document Validation



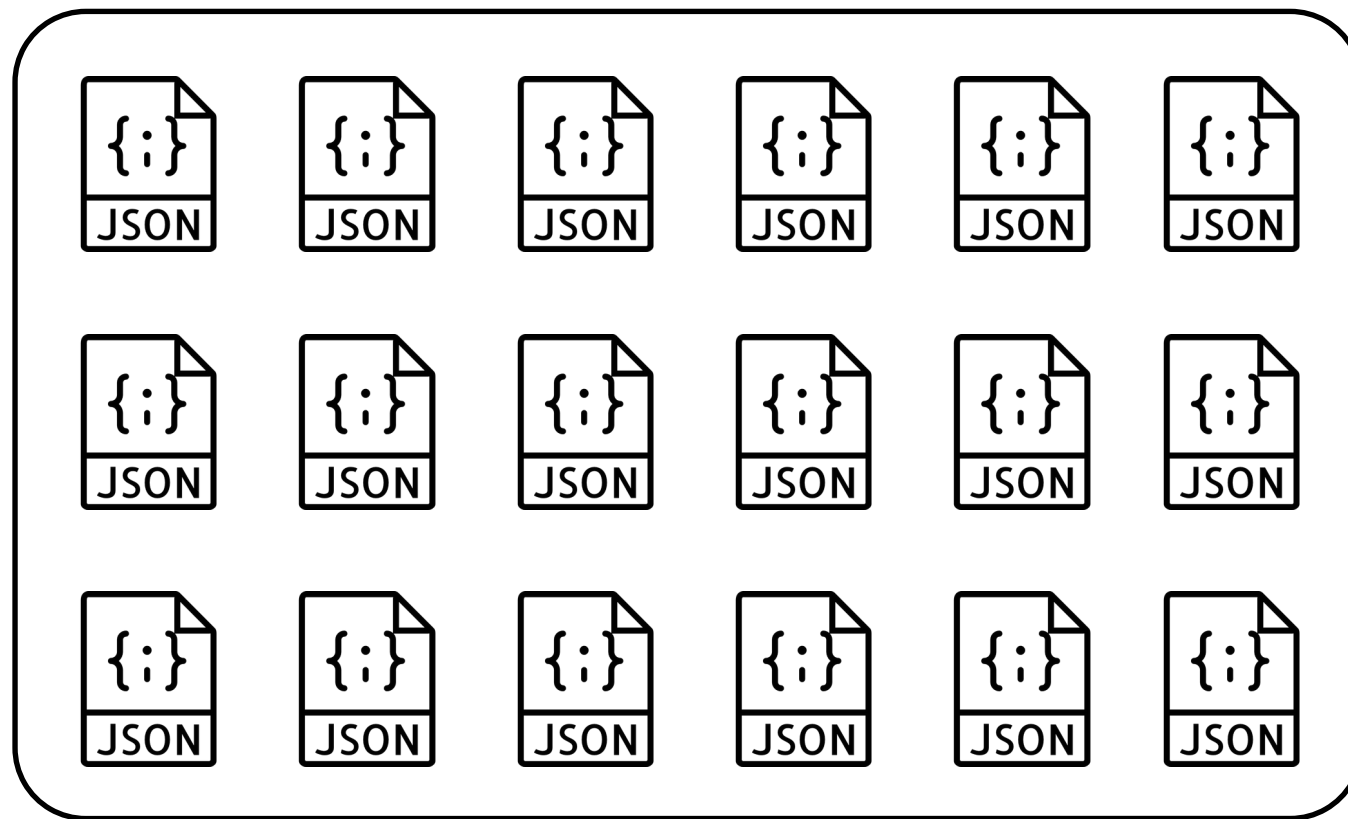
JSON
document
rejected



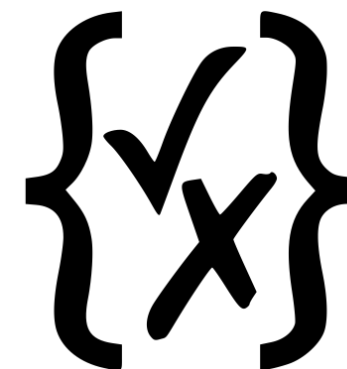
JSON validator



JSON Schema Discovery



JSON documents



JSON schema

Background

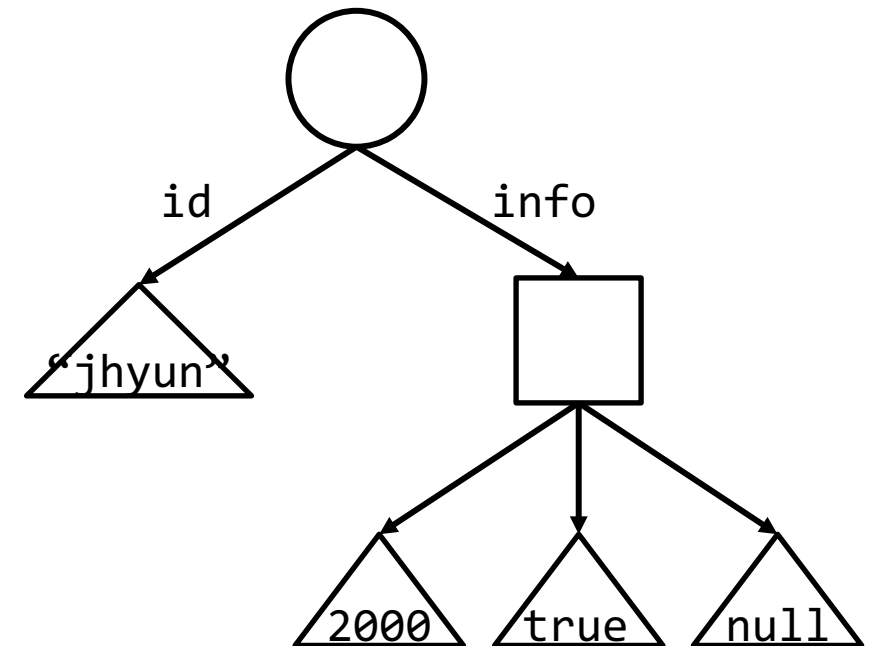
JSON Document and Tree Representation

For visibility, we express a JSON document with a node-labeled, node-typed, edge-labeled tree called an *instance tree*.

- 6 data types in total

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

An example JSON document



An equivalent instance tree

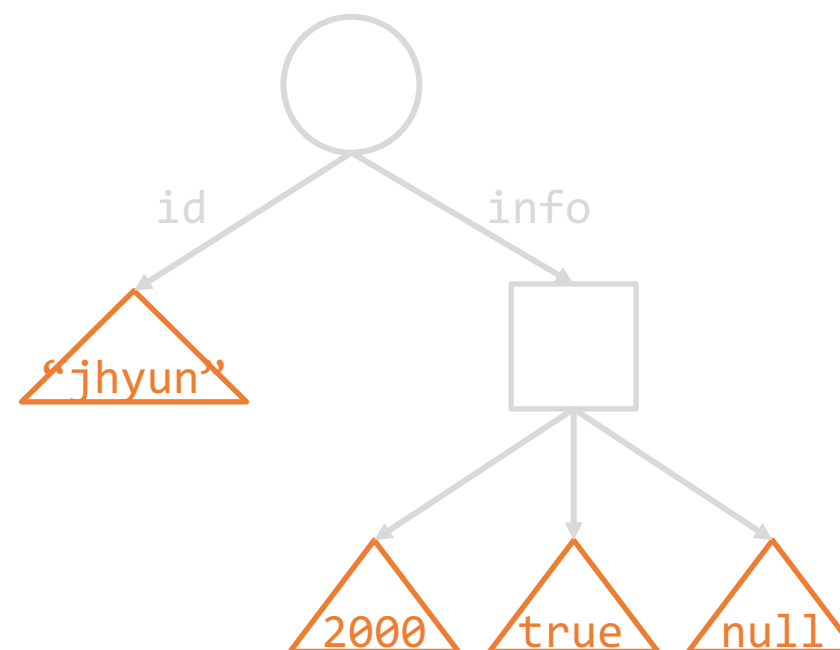
Background

JSON Document and Tree Representation

Primitive type document is

- one of *string*, *number*, *boolean*, *null*.
- expressed with a primitive-typed node with its value as label.

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



Background

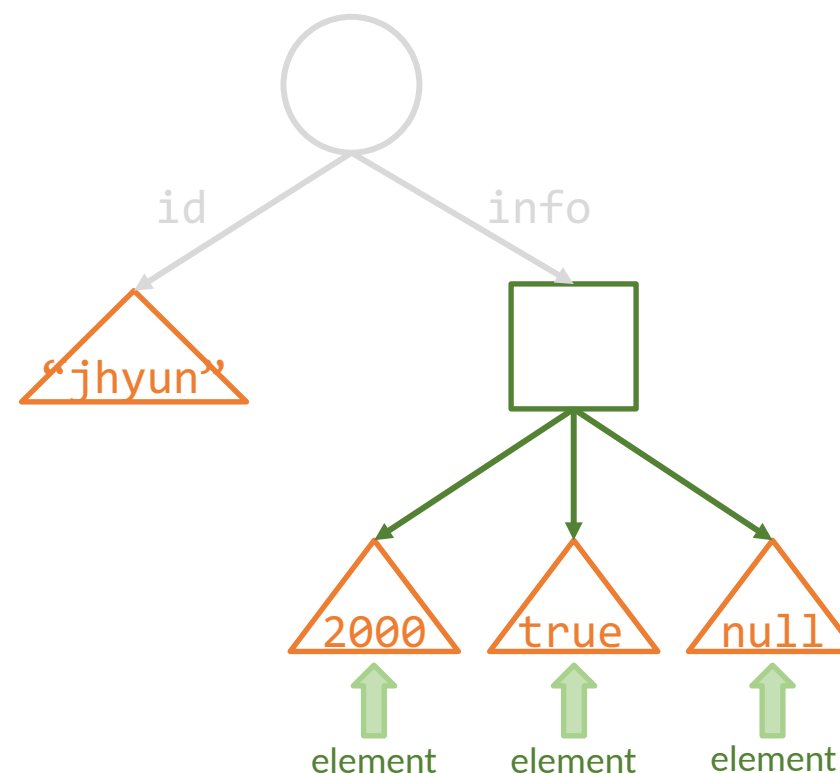
JSON Document and Tree Representation

Array type document is

- a *sequence of elements* (JSON documents).
- expressed with an array-typed node with elements as ordered children.

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

↑
element
↑
element
↑
element

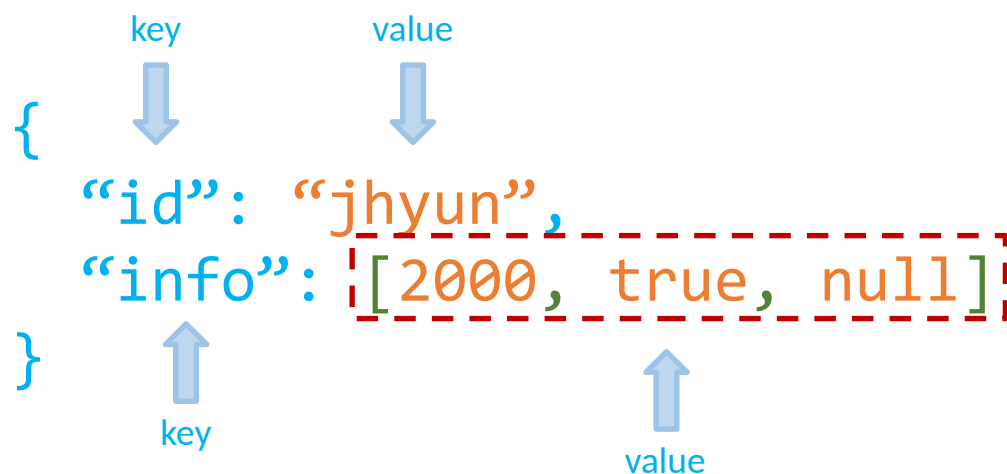
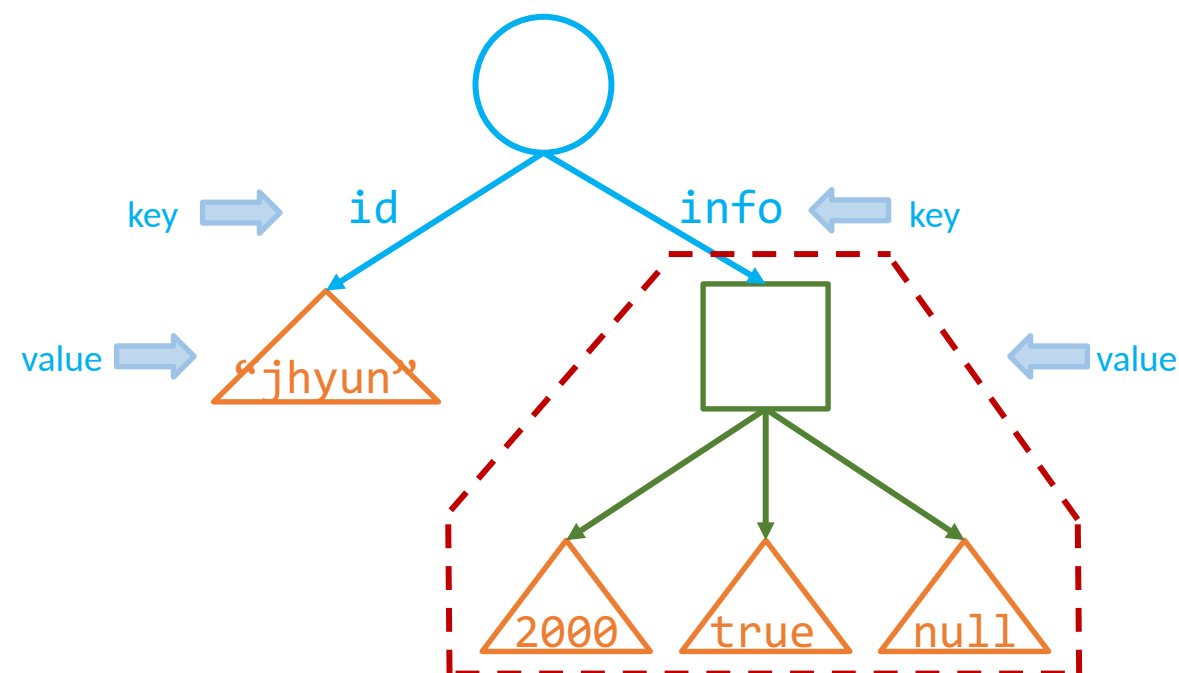


Background

JSON Document and Tree Representation

Object type document is

- a set of *key, value* (JSON document) *pairs*.
- expressed with an object-typed node with keys as edge labels and values as unordered children.



Background

JSON Schema

JSON schema

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

Background

JSON Schema and Tree Representation

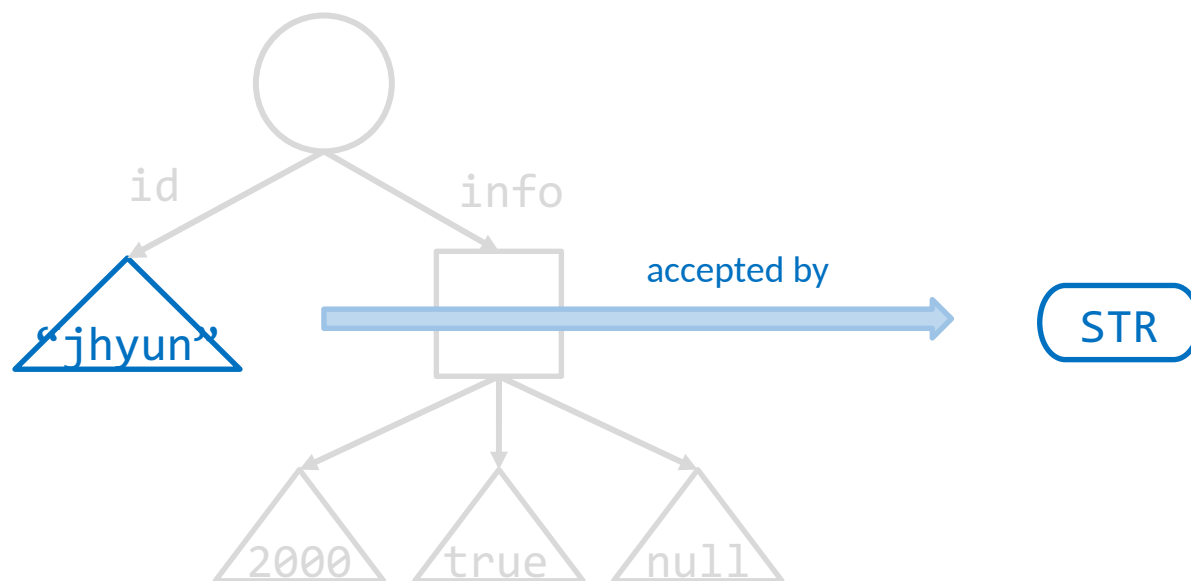
Primitive type schemas trivially impose JSON documents to be a document of corresponding type.

Background

JSON Schema and Tree Representation

Primitive type schemas trivially impose JSON documents to be a document of corresponding type.

- String schema

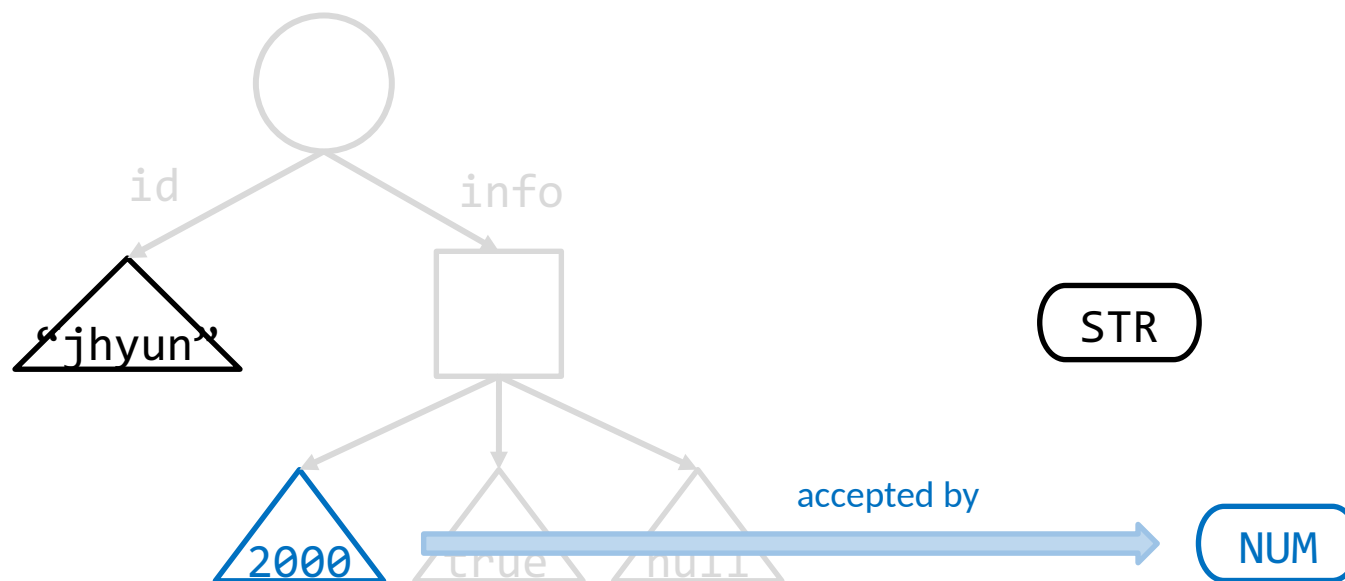


Background

JSON Schema and Tree Representation

Primitive type schemas trivially impose JSON documents to be a document of corresponding type.

- String schema, **number schema**

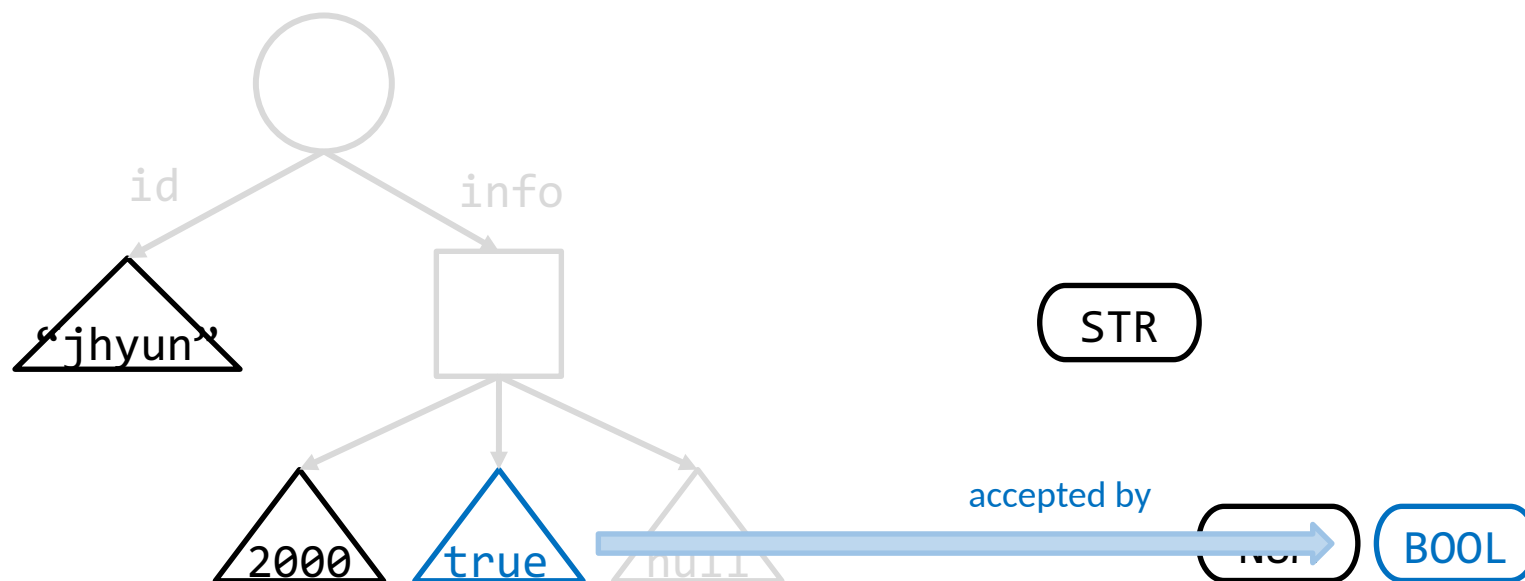


Background

JSON Schema and Tree Representation

Primitive type schemas trivially impose JSON documents to be a document of corresponding type.

- String schema, number schema, **boolean schema**

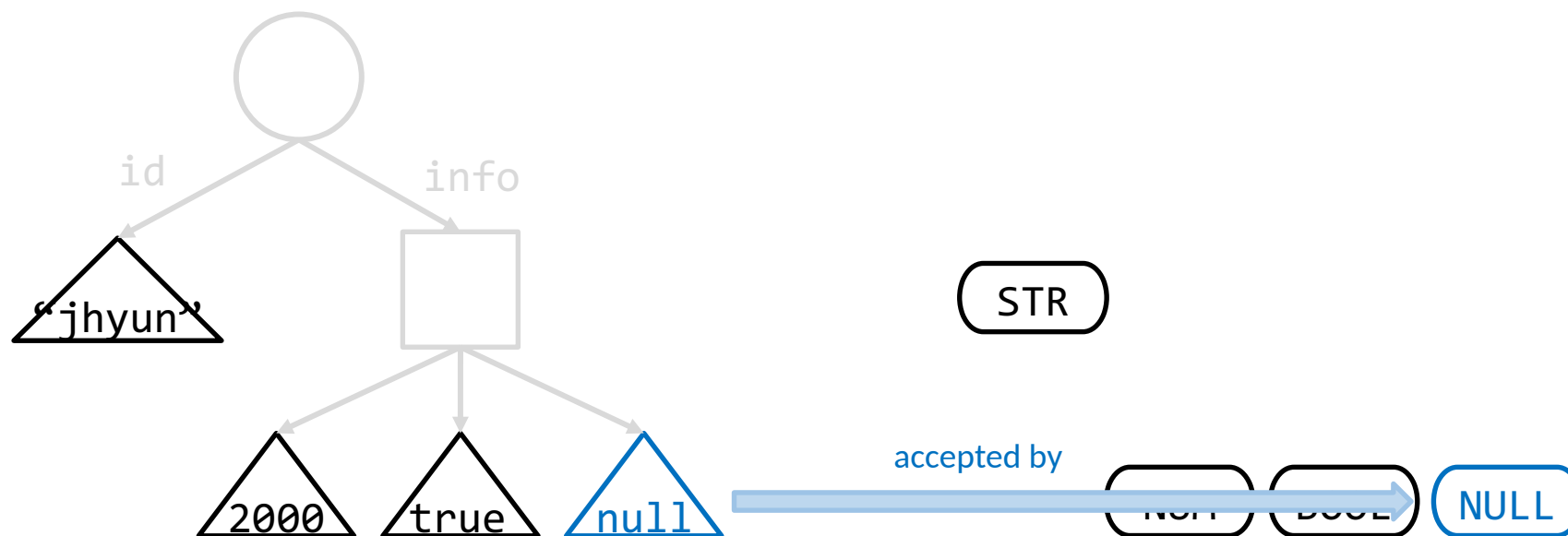


Background

JSON Schema and Tree Representation

Primitive type schemas trivially impose JSON documents to be a document of corresponding type.

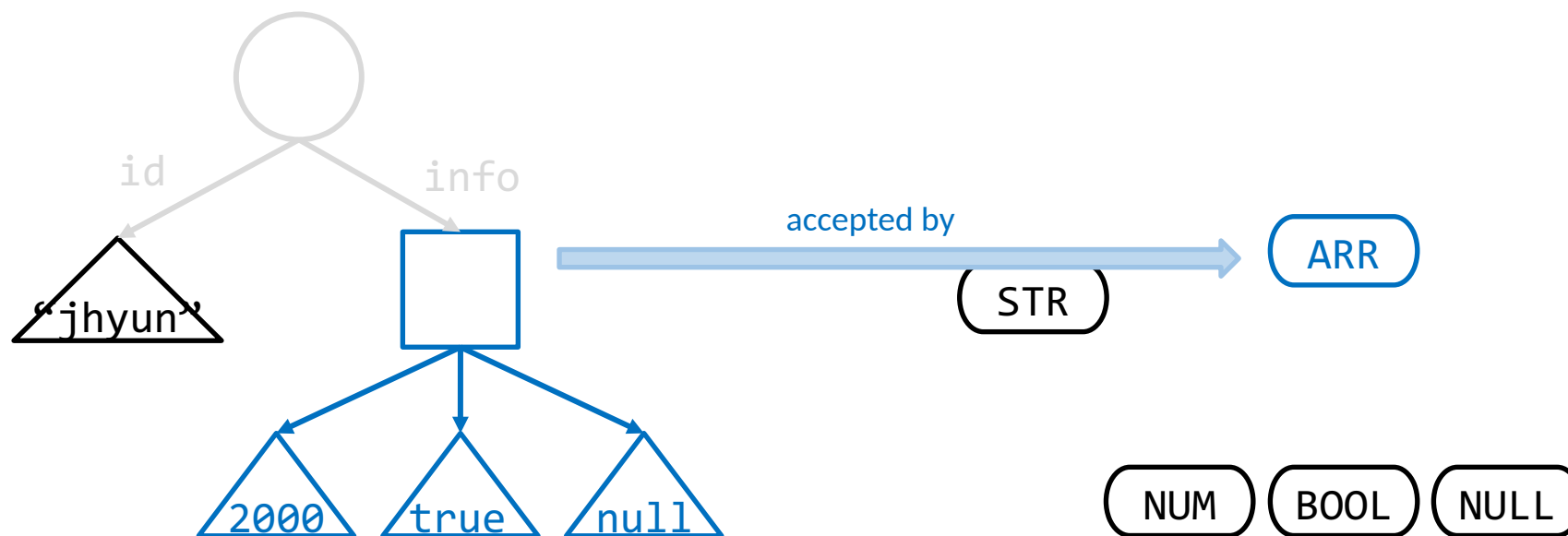
- String schema, number schema, boolean schema, **null schema**



Background

JSON Schema and Tree Representation

Array schemas additionally impose constraints on the number and schemas of elements.

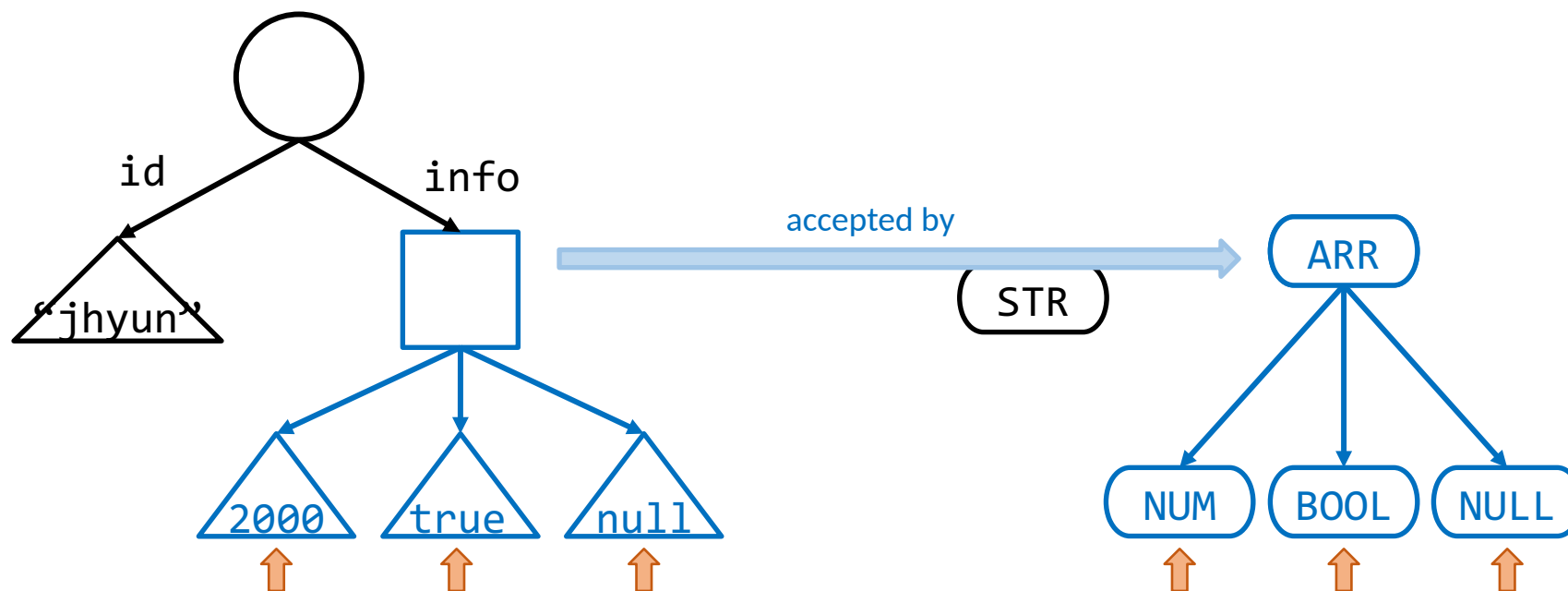


Background

JSON Schema and Tree Representation

Array schemas additionally impose constraints on the number and schemas of *elements*.

- **Homogeneous array schema** constrains arrays to have
 - Fixed length
 - Fixed subschema for element at each index

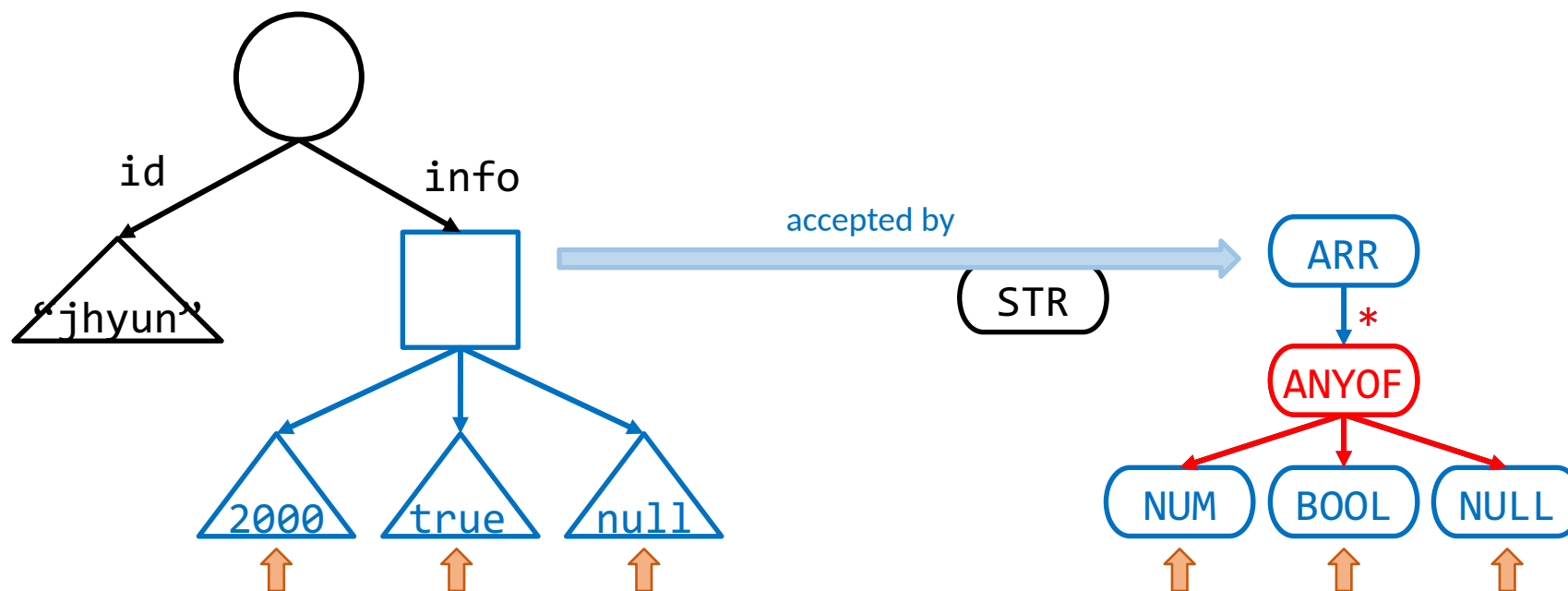


Background

JSON Schema and Tree Representation

Array schemas additionally impose constraints on the number and schemas of *elements*.

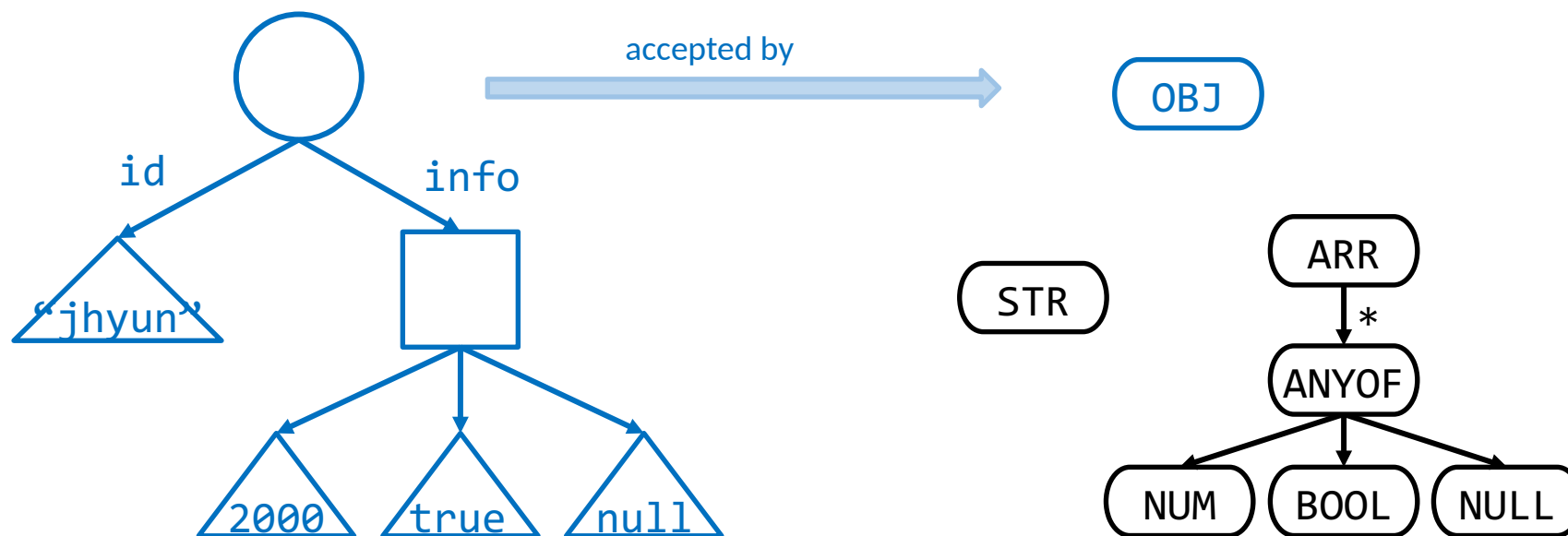
- **Heterogeneous array schema** constrains arrays to have
 - Random lengths
 - Fixed single subschema for every element



Background

JSON Schema and Tree Representation

Object schemas additionally impose constraints on each *keys* and schema of each *values*.

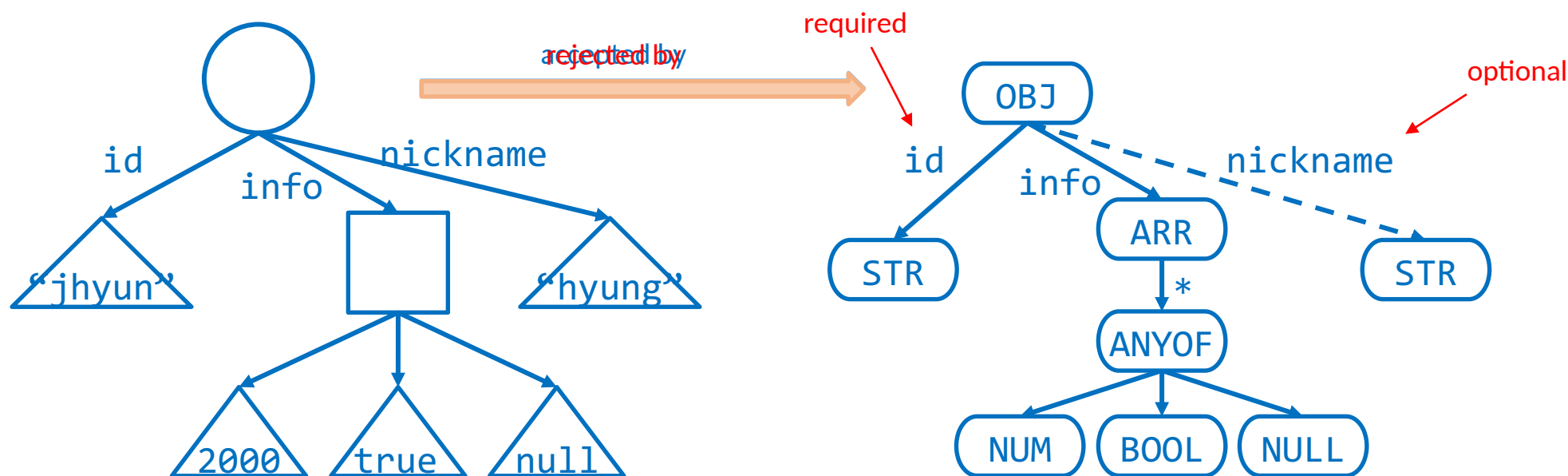


Background

JSON Schema and Tree Representation

Object schemas additionally impose constraints on each *keys* and schema of each *values*.

- **Homogeneous object schema** constrains objects to have
 - Homogeneous keys (either required or optional)
 - One subschema per key

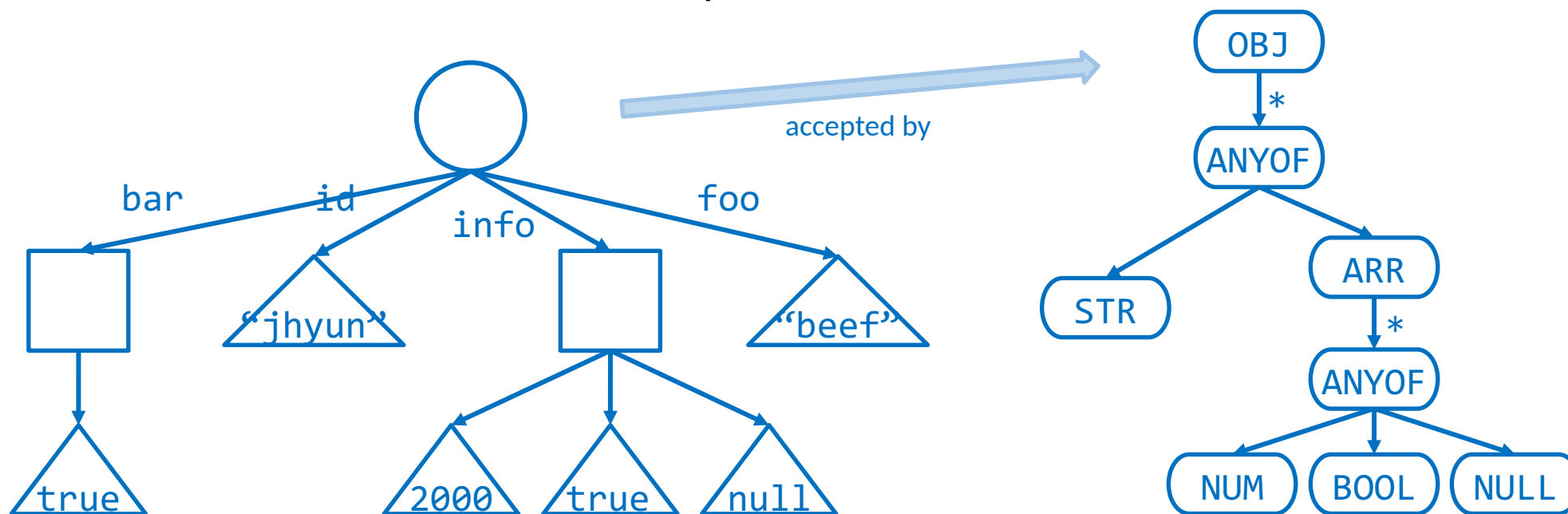


Background

JSON Schema and Tree Representation

Object schemas additionally impose constraints on each *keys* and schema of each *values*.

- **Heterogeneous object schema constrains objects to have**
 - Heterogeneous keys & random number of key-value pairs
 - Same subschema for every value



JSON Schema Discovery Problem

• Suppose there is a **ground truth schema** $\mathcal{S}_G$.

Input (D^+): a bag of JSON documents accepted by $\mathcal{S}_G$

Output ($\mathcal{S}_D$): a discovered schema

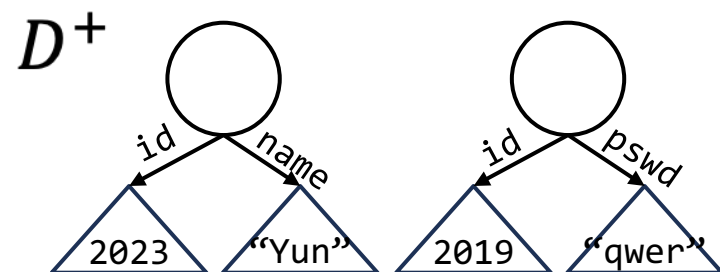
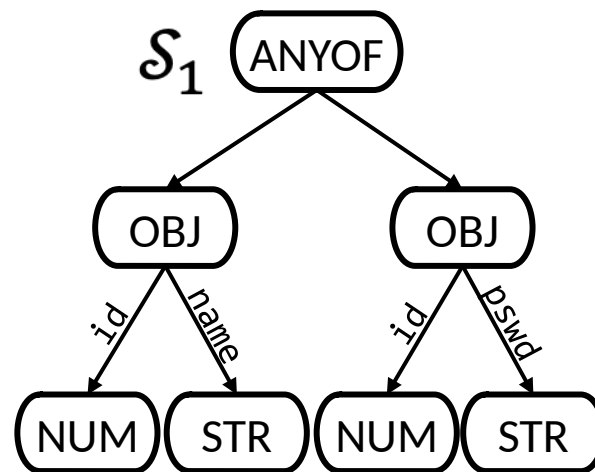
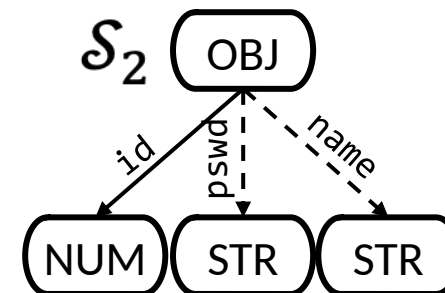
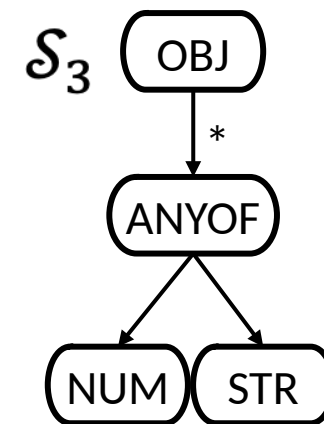
$\mathcal{S}_D$ is assessed with a positive set and a negative set.

- **Positive set:** JSON documents accepted by $\mathcal{S}_G$
- **Negative set:** JSON documents rejected by $\mathcal{S}_G$

An ideal (i.e., good) $\mathcal{S}_D$ should *accept all documents in the positive set and reject all documents in the negative set*

Challenges of JSON Schema Discovery

- Innumerable candidates of schemas that can be derived
- Ambiguity in picking the best schema among the candidates
 - We do not have access to positive and negative set while deriving $\mathcal{S}_D$.



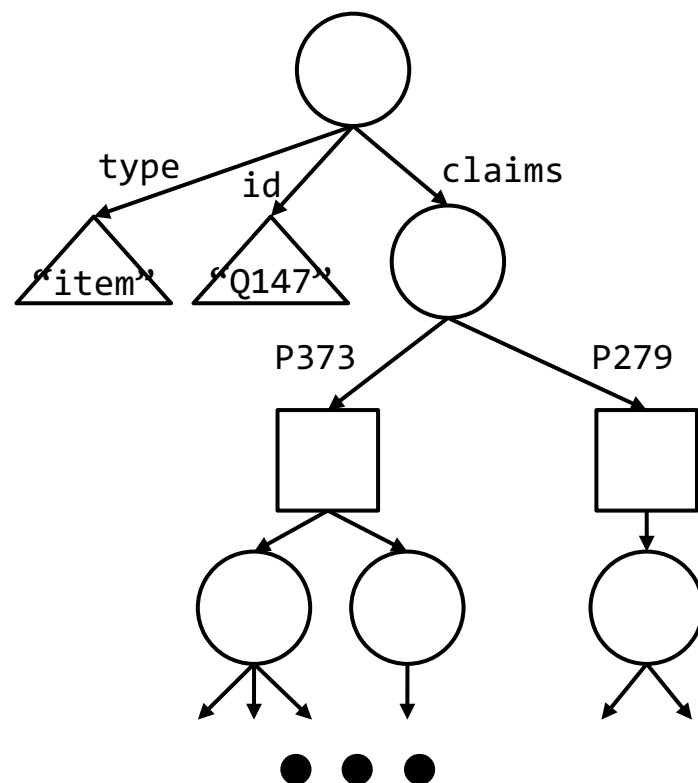
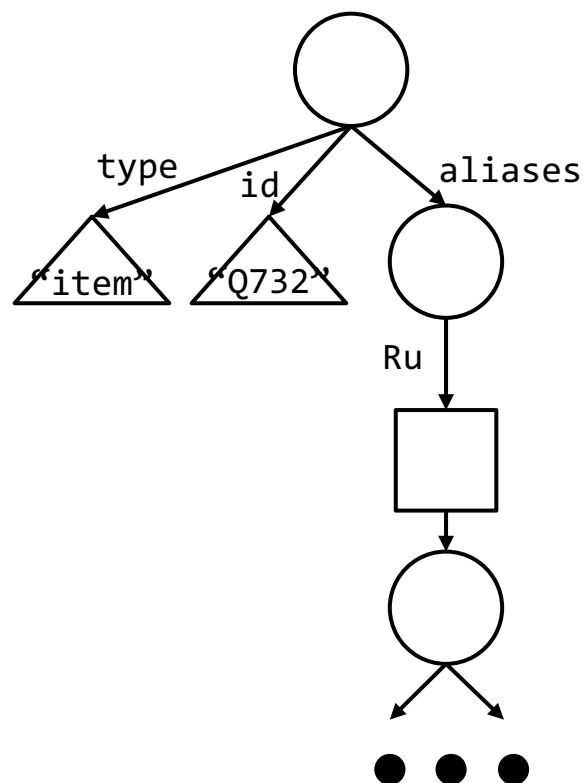
Specific General

Challenges of JSON Schema Discovery

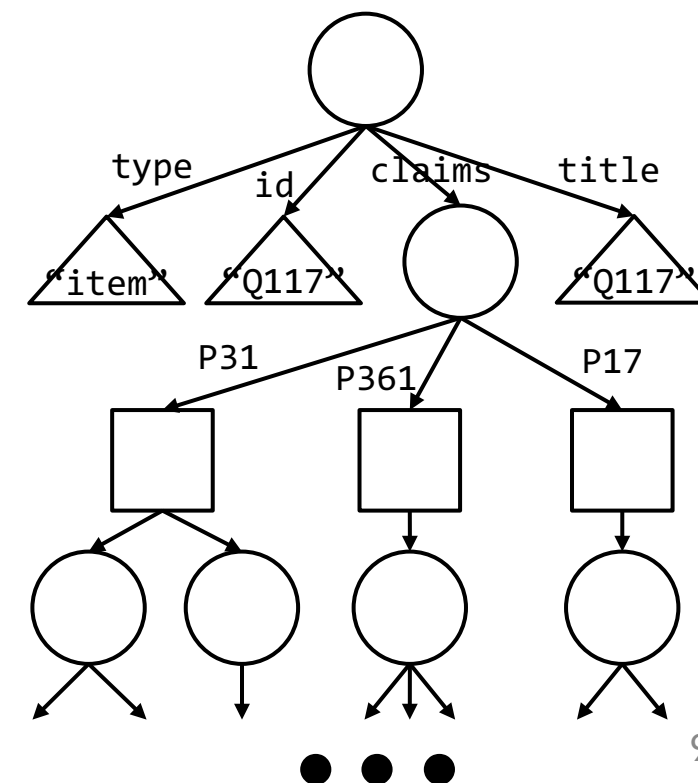
- Innumerable candidates of schemas that can be derived
- Ambiguity in picking the best schema among the candidates
 - We do not have access to positive and negative set while deriving $\mathcal{S}_D$.

What if there are many, highly nested JSON documents?

D^+

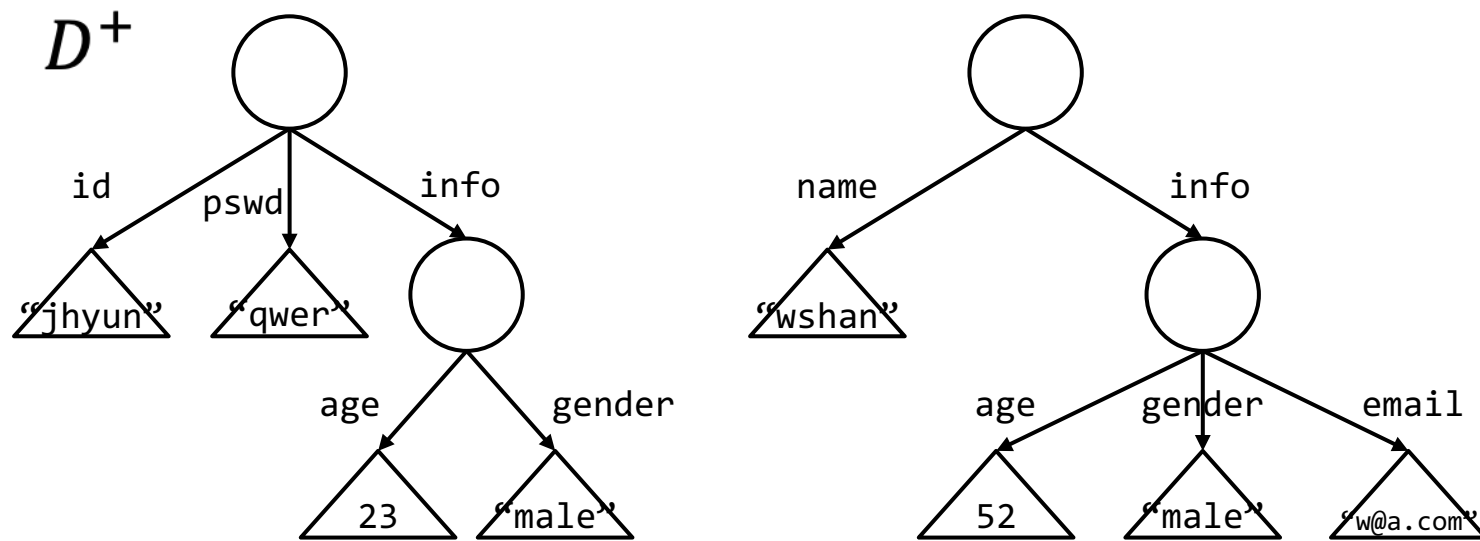


...



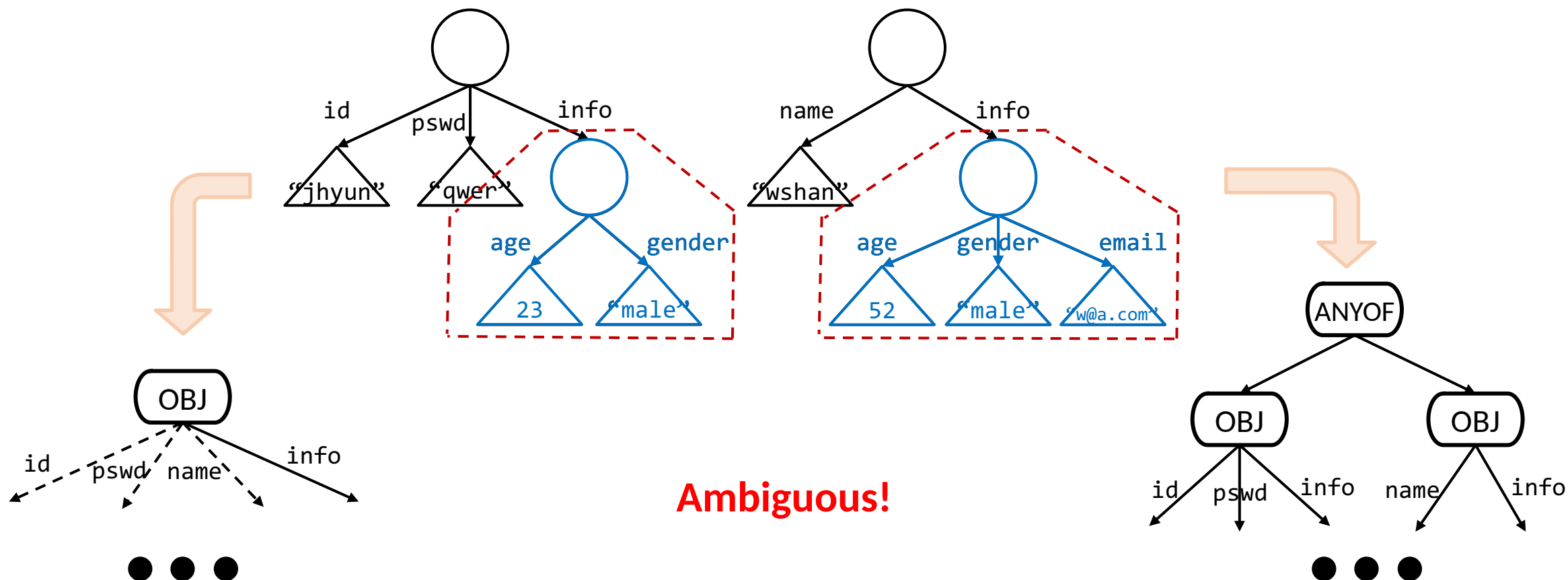
Global Example

- Let's derive an ideal schema S_D for D^+ comprised of the following two JSON instances.



Motivation 1: Insufficient Information to Derive Schema

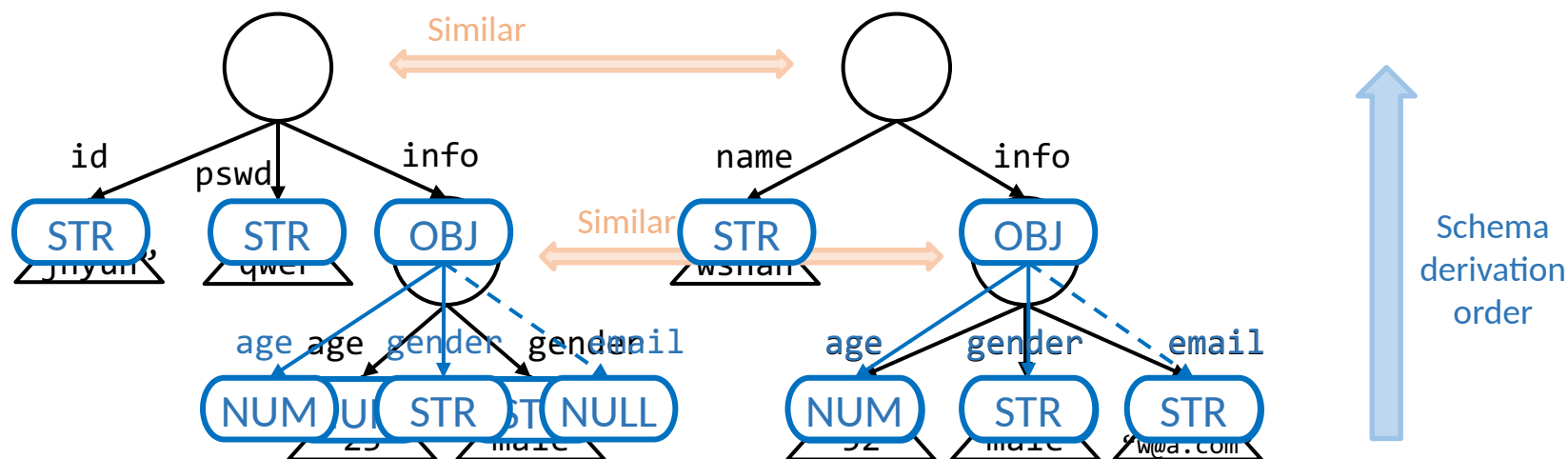
- Existing works derive schema in a top (root) – down (leaf) manner.
- Information may be insufficient to correctly derive a schema
 - Schemas of children instances are not derived.



Key Idea 1: Bottom-up Schema Generation

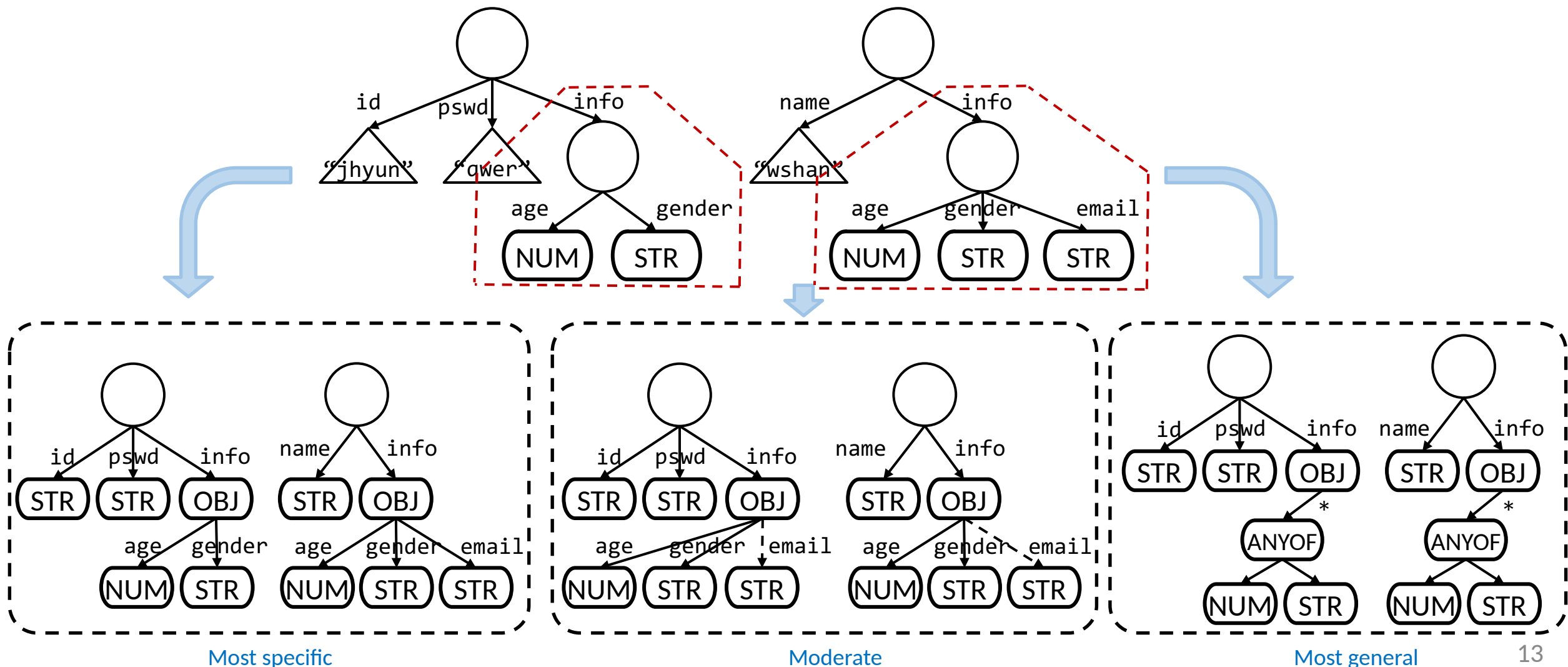
We propose deriving schema in a bottom-up manner.

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



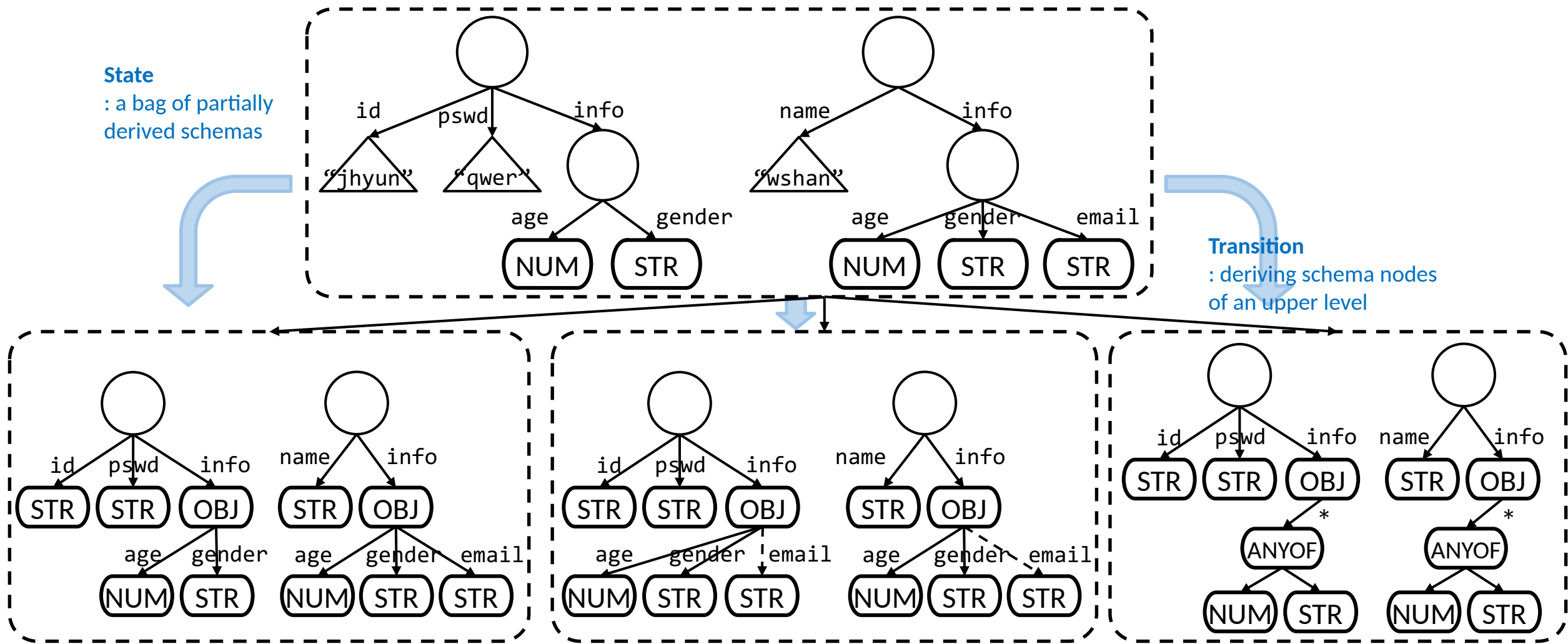
Motivation 2: Many Candidate Derivations

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



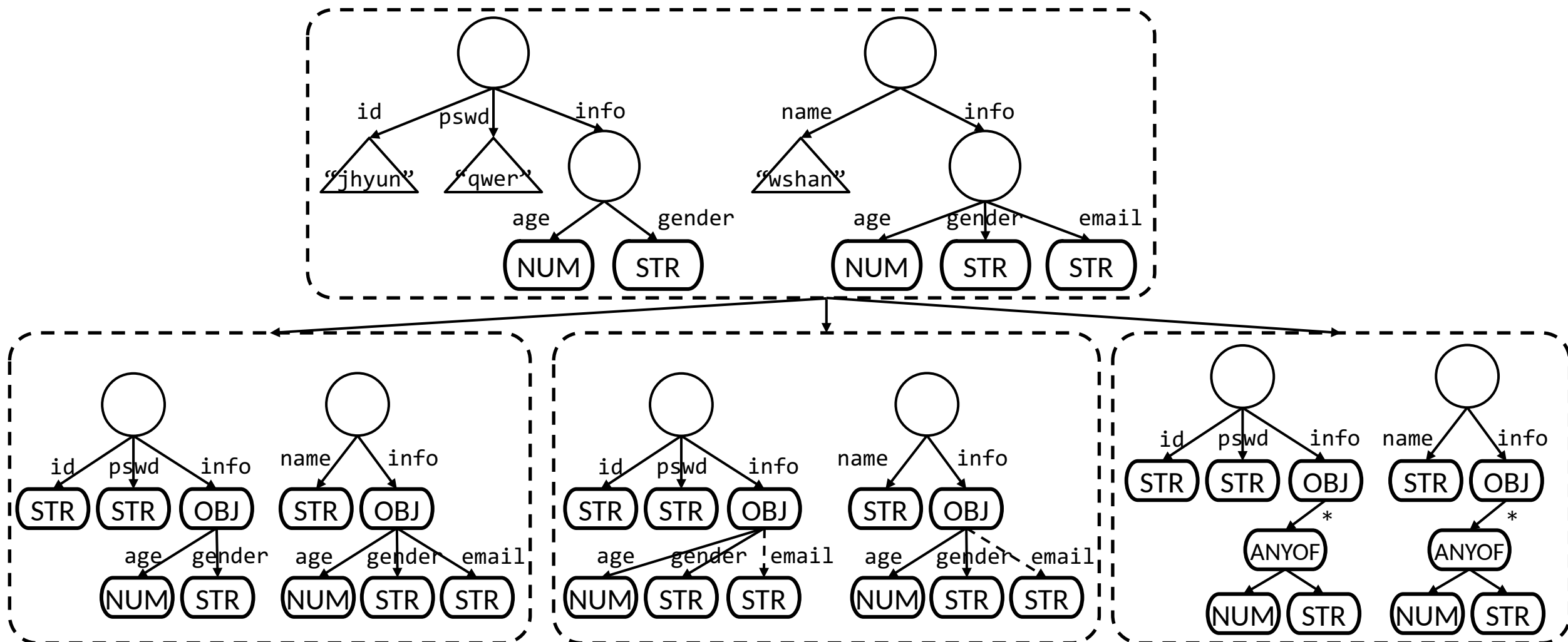
Key Idea 2: Candidate Schema Exploration

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



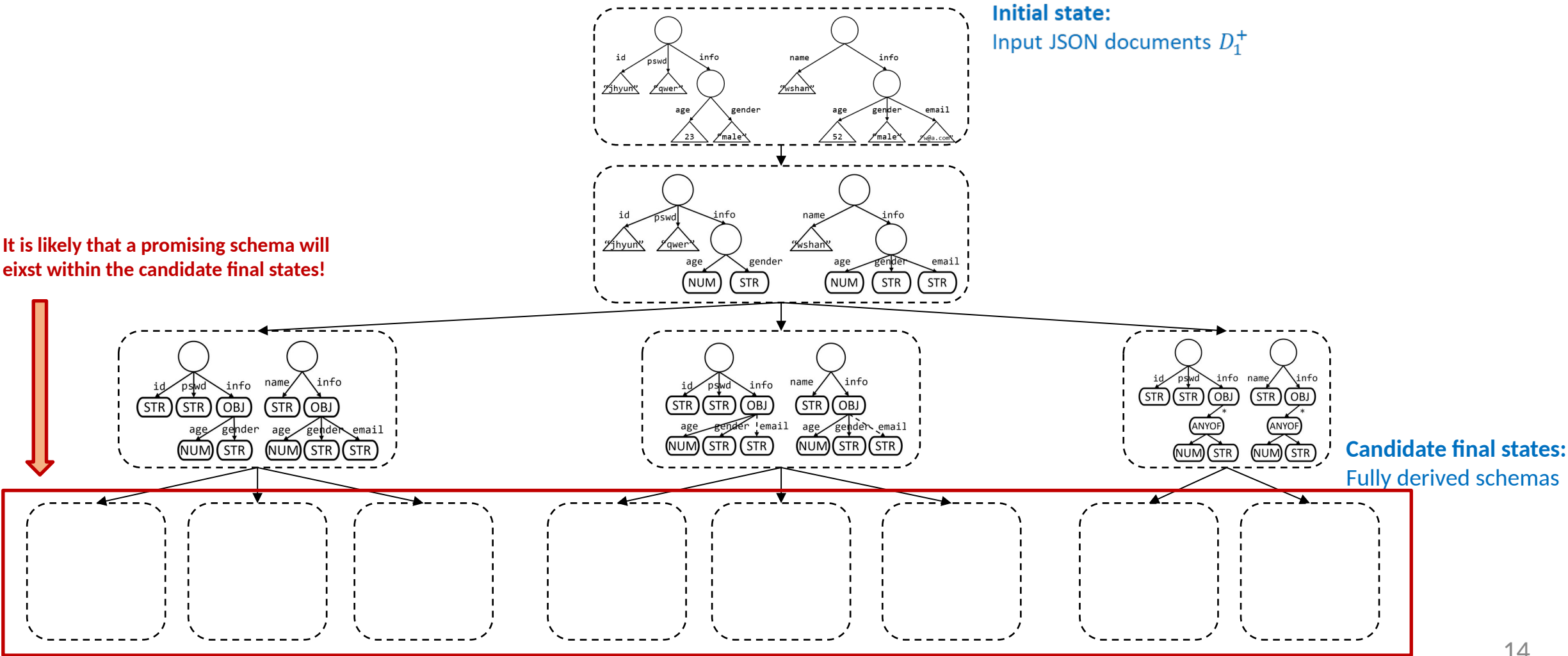
Key Idea 2: Candidate Schema Exploration

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



Key Idea 2: Candidate Schema Exploration

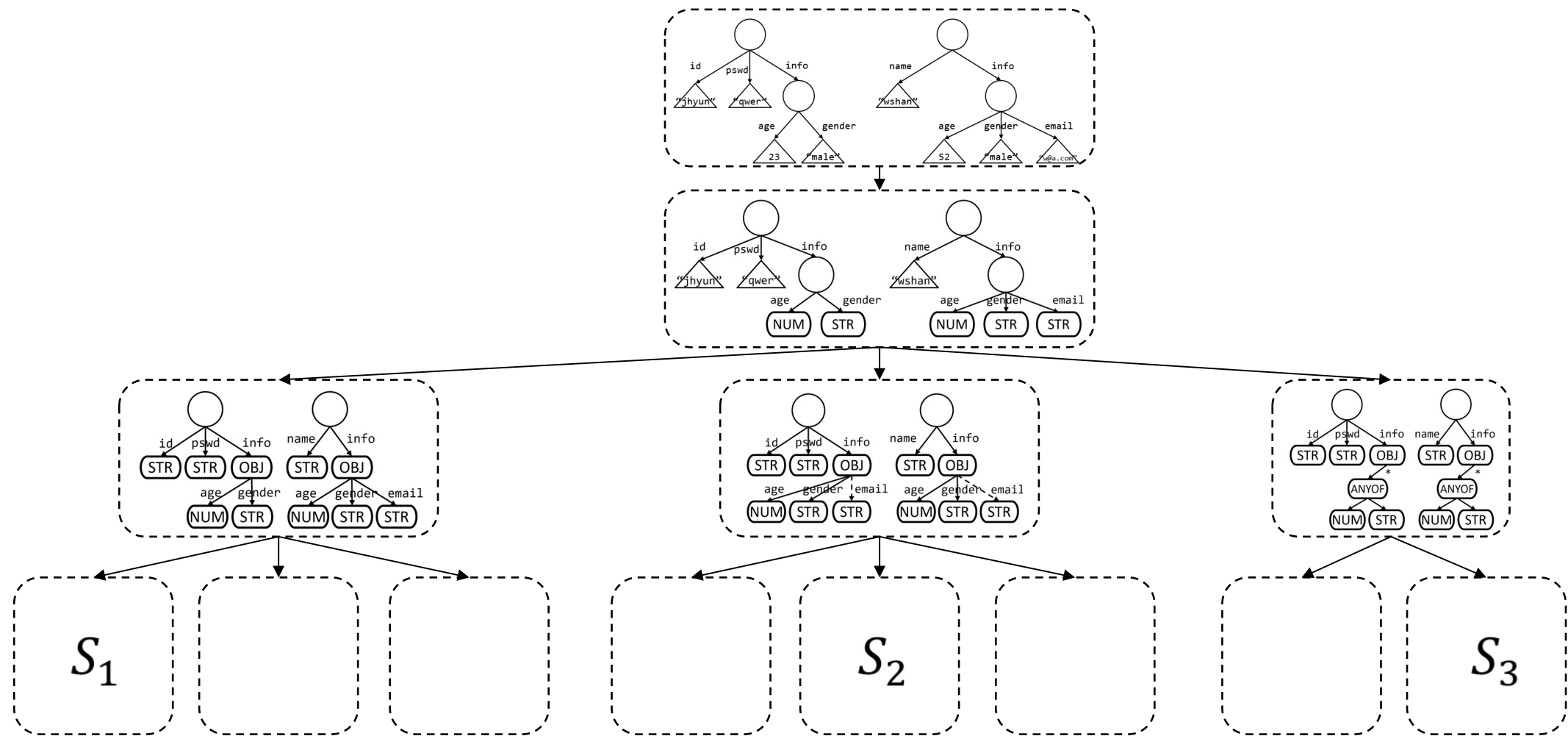
ReCG explores multiple schema derivation candidates rather than settling on just one.



* The schemas within the candidate final states are not expressed for brevity.

Motivation 3: Ambiguity in Choosing the Best Schema

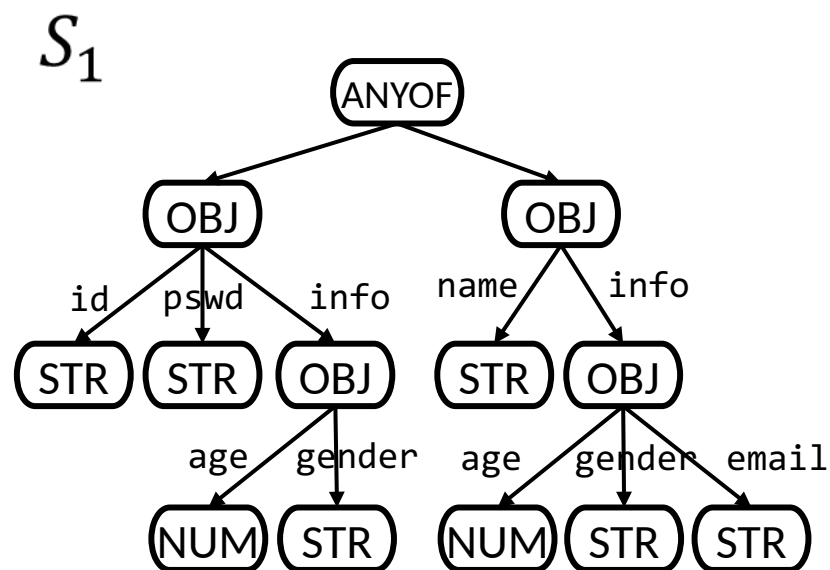
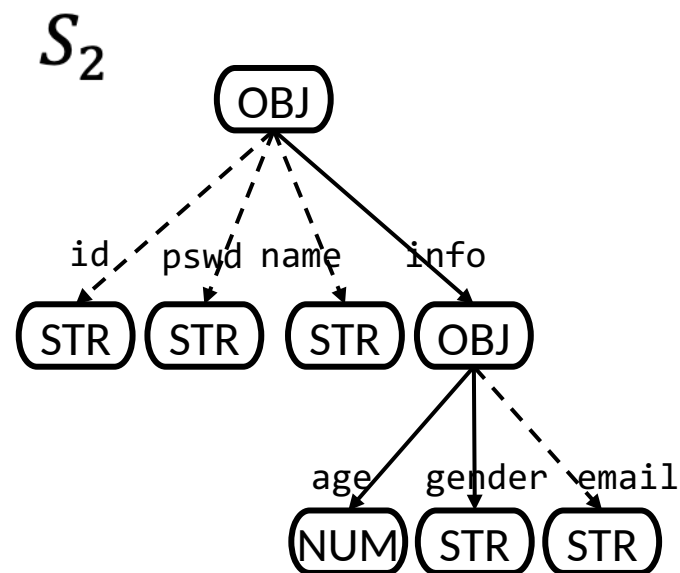
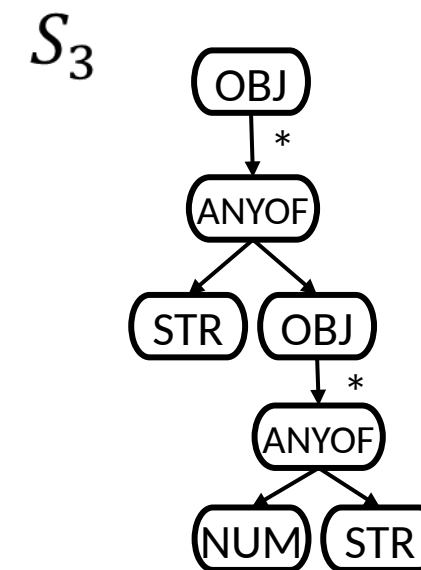
We need a metric that correctly measure the “goodness” of a schema.



* The schemas within the candidate final states are not expressed for brevity.

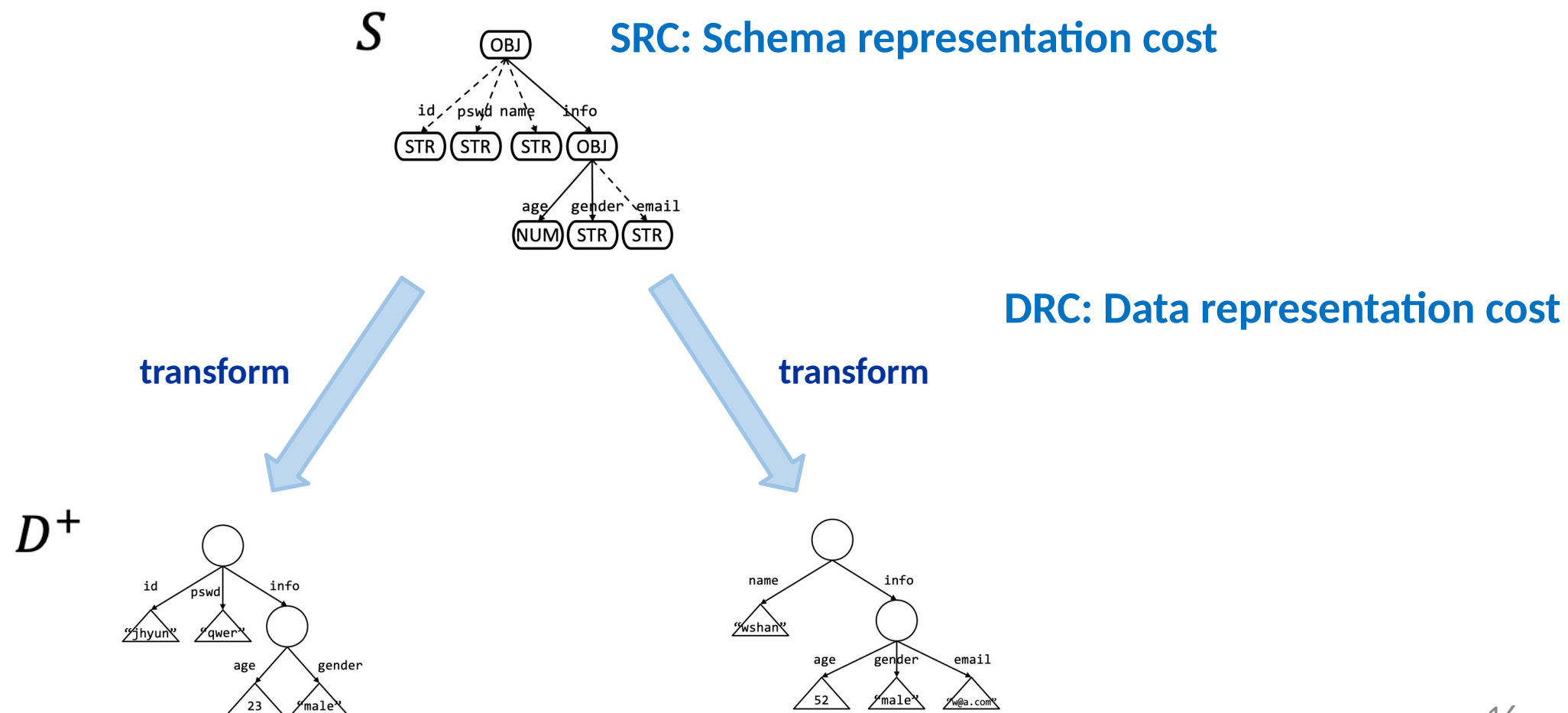
Motivation 3: Ambiguity in Choosing the Best Schema

We need a metric that correctly measure the “goodness” of a schema.



Key Idea 3: Minimum Description Length Cost

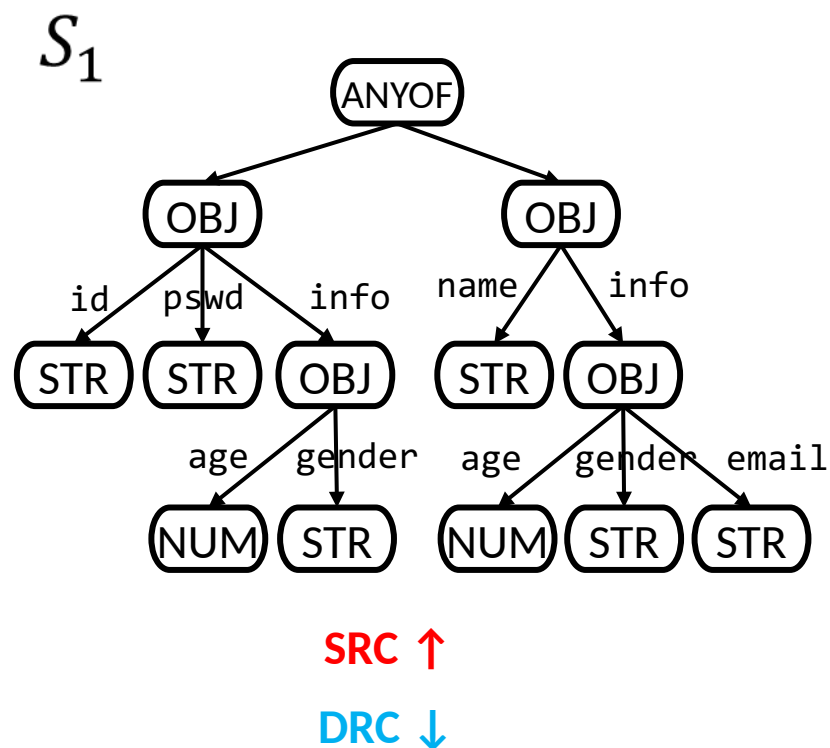
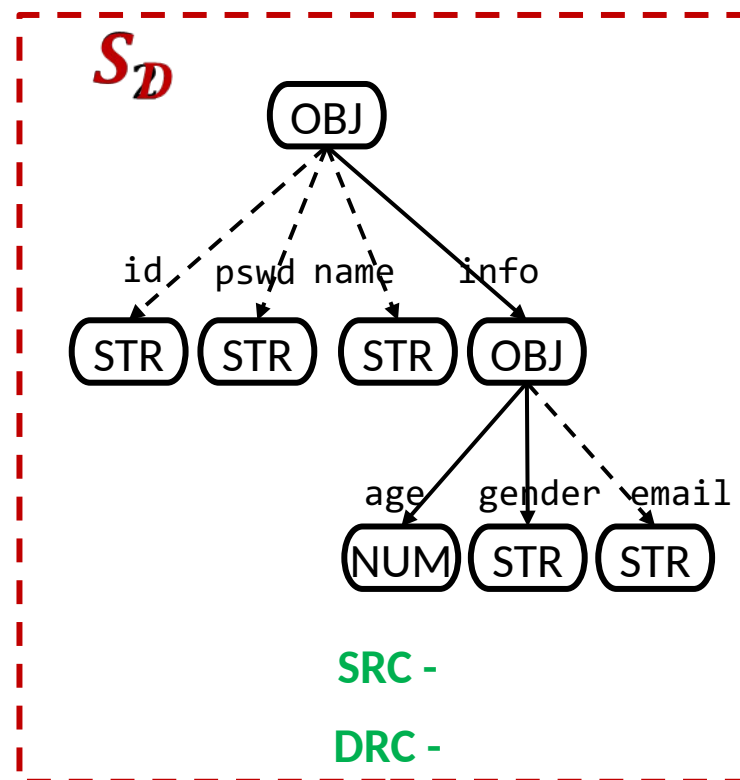
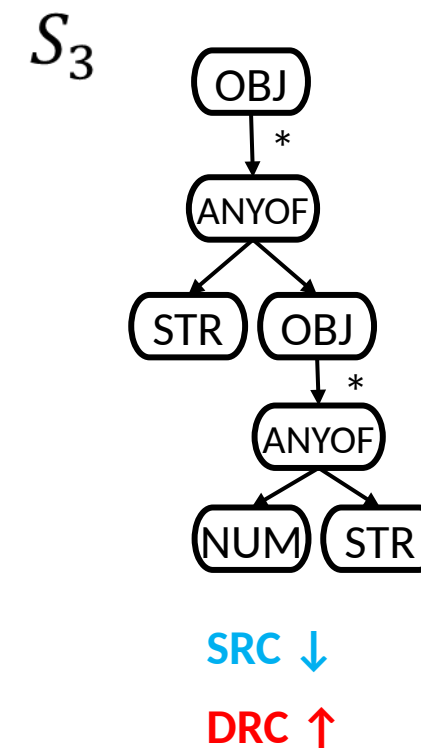
Minimum description length principle: minimize the total amount of information required to describe the input data



Key Idea 3: Minimum Description Length Cost

Intuition: minimize the total amount of information required to describe the input data

- **SRC:** the size of the schema tree *(penalizes too specific schema)*
- **DRC:** the cost needed to transform schema into D^+ *(penalizes too general schema)*



Contributions of ReCG

- 46% higher F1 score while also performing 2.11× faster on average against the state-of-the-art algorithm
- Bottom-up schema generation
- Search for schema candidate
- Minimum description length cost

Experiment

Algorithms compared

- Jxplain (SOTA), KReduce, LReduce, KSS (Kletteke et al.), FMC (Frozza et al.)

Machine

- Intel Xeon E5-2680 v4 @2.40GHz CPU and 756 GB RAM

Datasets

- Table 1: Statistics of 20 datasets used in experiments.

Dataset			Schema								Instance	
Type	Name	Height	V _s	# of OBJ nodes			# of ARR nodes		# of ANYOF nodes	D ⁺	avg(V _t)	
				Hom Obj	Het Obj	Com Obj	Hom Arr	Het Arr				
Real-life	NYT	6	92	9	0	0	0	3	14	10k	85.21	
	Twitter	∞	∞	20	1	0	12	10	16	10k	206.16	
	Github	11	3471	171	0	3	0	29	335	10k	116.64	
	Pharmaceutical	3	12	2	1	0	0	0	0	10k	31.77	
	Wikidata	14	179	31	7	0	0	8	15	10k	1927.96	
	Yelp	5	79	7	1	0	0	0	5	10k	12.32	
	VK	11	335	40	0	0	0	7	2	10k	30.50	
	ETH	8	112	8	0	0	1	6	6	10k	1004.69	
	Iceberg	4	9	1	1	0	0	1	0	1523	1288.30	
	Ember	6	68	8	1	0	0	9	0	10k	902.86	
Synthetic	GeoJSON	8	41	6	0	0	2	5	1	10k	52.65	
	ThaiMovies	8	112	14	0	0	0	11	6	1364	433.79	
	RDB	3	13	1	0	1	0	1	0	10k	14.76	
	AdonisRC	7	64	5	2	2	0	9	3	10k	27.77	
	HelmChart	7	50	4	0	1	0	6	1	10k	33.76	
	Dolittle	6	52	14	6	0	0	3	1	10k	48.82	
	Drupal	6	100	10	7	0	0	17	5	10k	47.96	
	DeinConfig	8	97	3	1	2	0	13	17	10k	44.94	
	Ecosystem	6	120	5	3	1	0	12	9	10k	132.59	
	Plagiarize	4	15	2	1	1	0	0	2	10k	8.23	

Accuracy Comparison

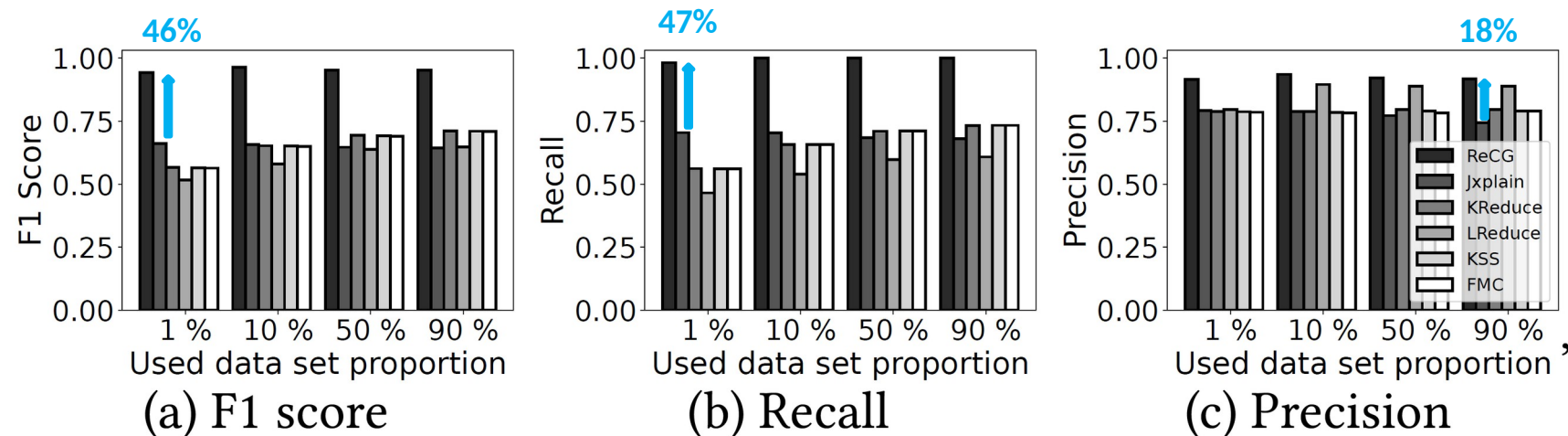


Figure 9: Accuracy comparison with varying dataset sizes.

ReCG outperforms all competitors in F1 score, especially shows 46% increase compared to the SOTA

- Shows upto 47% higher recall compared to SOTA
- Shows upto 18% higher precision compared to SOTA

MDL Cost Analysis

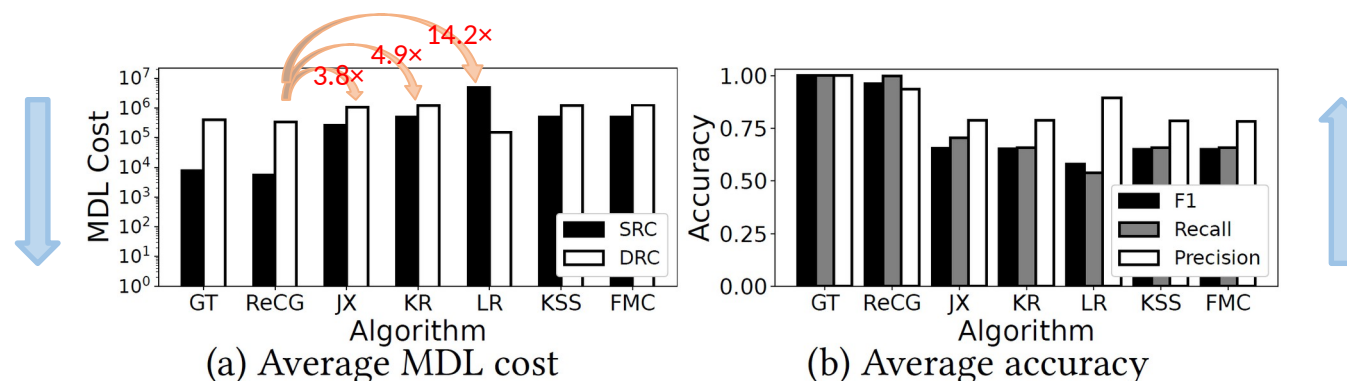


Figure 10: Average MDL cost and accuracy of the ground truth schema, and the schemas found by six algorithms for 20 datasets. The proportion of the used dataset is 10%. (GT: ground truth, JX: Jxplain, KR: KReduce, LR: LReduce)

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

Algorithm Execution Time

Table 3: Algorithm execution time comparison of ReCG against competitors. The third subcolumn under each algorithm is the relative speedup of each competitor against ReCG.

Used dataset proportion	ReCG			Jxplain			KReduce			LReduce			KSS			FMC		
	Avg runtime	Stdev	$\frac{\text{ReCG}}{\text{ReCG}}$	Avg runtime	Stdev	$\frac{\text{ReCG}}{\text{Jxplain}}$	Avg runtime	Stdev	$\frac{\text{ReCG}}{\text{KReduce}}$	Avg runtime	Stdev	$\frac{\text{ReCG}}{\text{LReduce}}$	Avg runtime	Stdev	$\frac{\text{ReCG}}{\text{KSS}}$	Avg runtime	Stdev	$\frac{\text{ReCG}}{\text{FMC}}$
10%	1489.62 ms	116.30	1.00	3130.67 ms	206.18	0.48	883.99 ms	34.89	1.69	1030.81 ms	58.16	1.45	168.32 ms	12.80	8.85	267.74 ms	73.37	5.56
50%	6571.55 ms	547.28	1.00	10128.23 ms	429.52	0.65	2689.82 ms	130.62	2.44	4892.20 ms	333.83	1.34	856.21 ms	45.51	7.68	1776.20 ms	557.19	3.70
100%	12571.91 ms	448.65	1.00	26479.15 ms	790.02	0.47	5006.07 ms	340.87	2.51	12603.57 ms	761.72	1.00	1748.04 ms	81.56	7.19	4959.49 ms	1644.84	2.53

ReCG's performance is positioned in the middle of KReduce and Jxplain

- ReCG outperforms Jxplain by 1.54× to 2.11× on average varying on dataset size
- ReCG is slower than KReduce by about 2×
 - ReCG spends time searching for various candidates of schema

More experiments in our paper!

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

THANK YOU