Universität Bielefeld

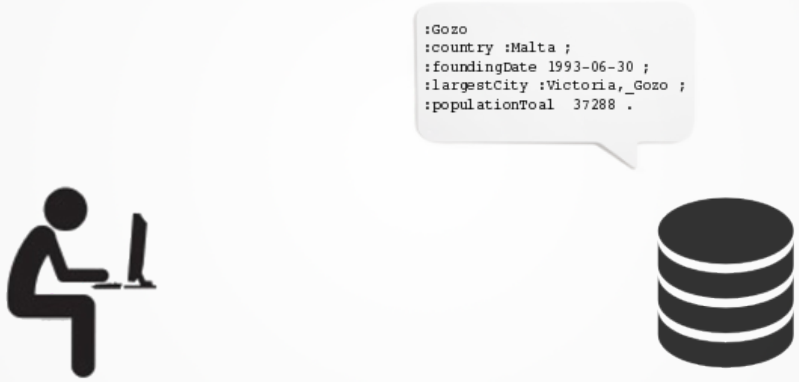
Schema-based application grammars for querying

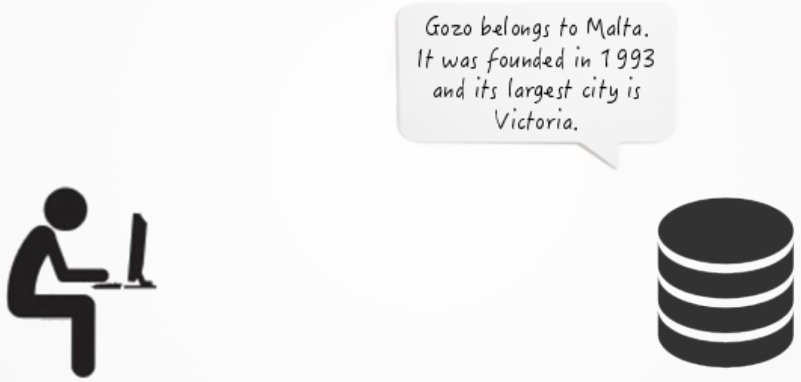
Christina Unger

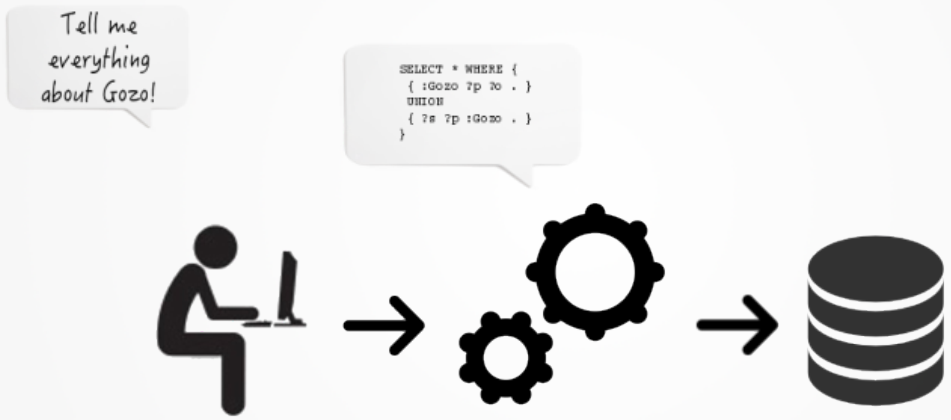
Semantic Computing Group
CITEC @ Universität Bielefeld

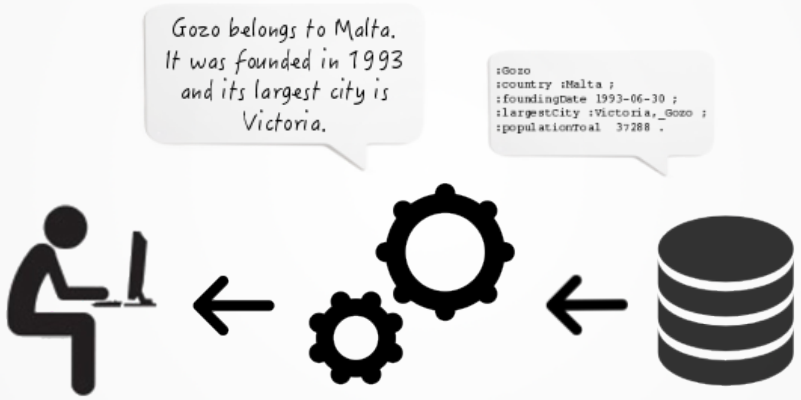

Tell me
everything
about Gozo!

```
SELECT * WHERE {
 { :Gozo ?p ?o . }
 UNION
 { ?s ?p :Gozo . }
}
```


:Gozo
:country :Malta ;
:foundingDate 1993-06-30 ;
:largestCity :Victoria,_Gozo ;
:populationToal 37288 .


Gozo belongs to Malta.
It was founded in 1993
and its largest city is
Victoria.

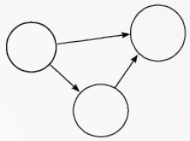

Tell me everything about Gozo!
SELECT * WHERE {
{ :Gozo ?p ?o . }
UNION
{ ?s ?p :Gozo . }
}


Gozo belongs to Malta.
It was founded in 1993
and its largest city is
Victoria.
:Gozo
:country :Malta ;
:foundingDate 1993-06-30 ;
:largestCity :Victoria_Gozo ;
:populationTotal 37288 .

A query system needs to know:

- natural language (Eng, Chi, Mlt,...)
- the formal query language (SQL, SPARQL,...)
- the structure and vocabulary of the data

SCHEMA (ONTOLOGY)



DATABASE (FACTS)

Outline

1. Schema-based grammars
2. From natural language to data
   - Transformation
   - Translation
   - Denotation
3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars

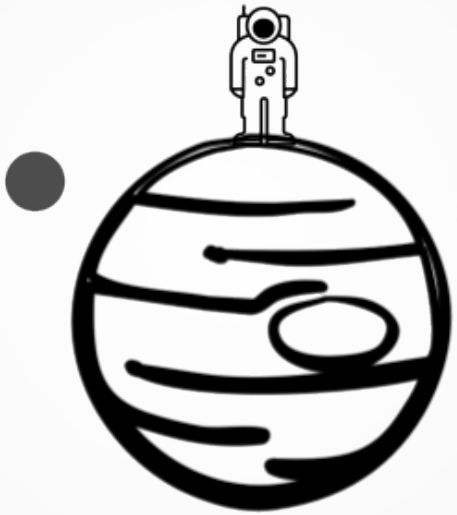
Outline

1. Schema-based grammars

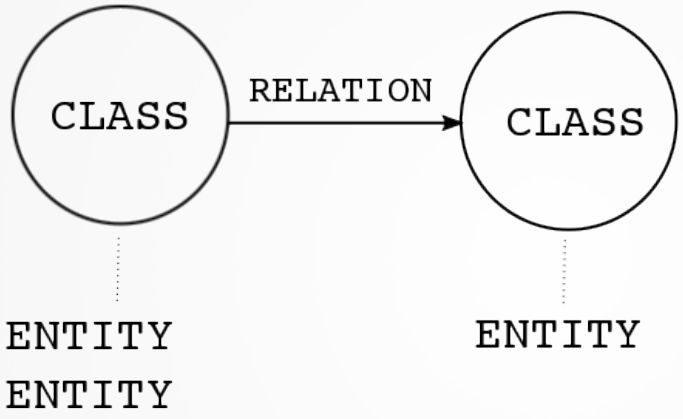
2. From natural language to data
   - Transformation
   - Translation
   - Denotation

3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars

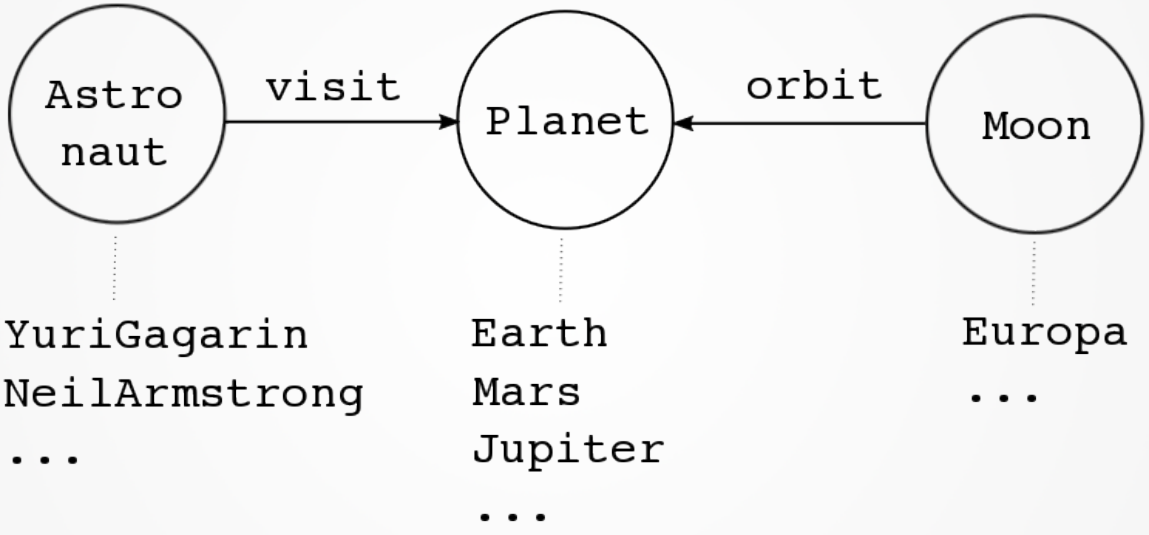
The domain



The domain as an ontology



The domain as an ontology




```
onto:Europa onto:orbit onto:Jupiter .
...
```

The ontology as a grammar

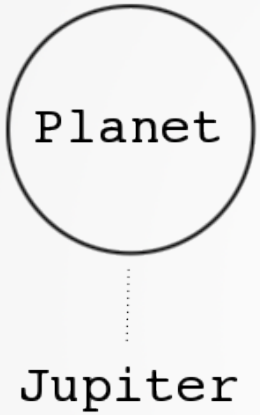
```
abstract Grammar where {

cat

 Class;
 Entity Class;
 Relation Class Class;
 Statement;
```

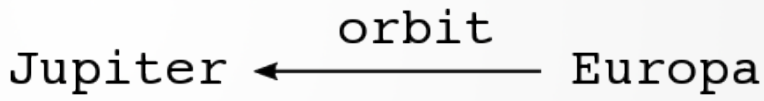
The ontology as a grammar

```
fun

  Astronaut, Planet, Moon : Class;

  Yuri, Neil, Scott, Mikhail  : Entity Astronaut;
  Earth, Mars, Jupiter        : Entity Planet;
  Europa                      : Entity Moon;

  orbit : Relation Moon Planet;
  visit : Relation Astronaut Planet;
```

```
 isA : (c : Class)
    -> Entity c
    -> Statement;

 app : (c1,c2 : Class)
    -> Relation c1 c2
    -> Entity c1
    -> Entity c2
    -> Statement;
}
```

Examples






```
(isA Planet Jupiter)    -- Jupiter is a planet.
(isA Moon Europa)       -- Europa is a moon.

(app Moon Planet orbit Europa Jupiter)
                        -- Europa orbits Jupiter.
```

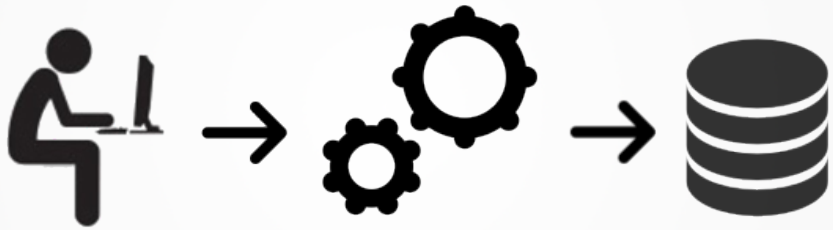
```
concrete GrammarEng where {

lincat

 Class     = CN;
 Entity    = NP;
 Relation  = V2;
 Statement = Cl;

lin

 Yuri  = mkNP (mkPN "Yuri Gagarin");
 Moon  = mkCN (mkN  "moon");
 orbit = mkV2 (mkV  "orbit");

 isA c e      = mkCl e  (mkVP c);
 app r e1 e2 = mkCl e1 (mkVP r e2);
}
```

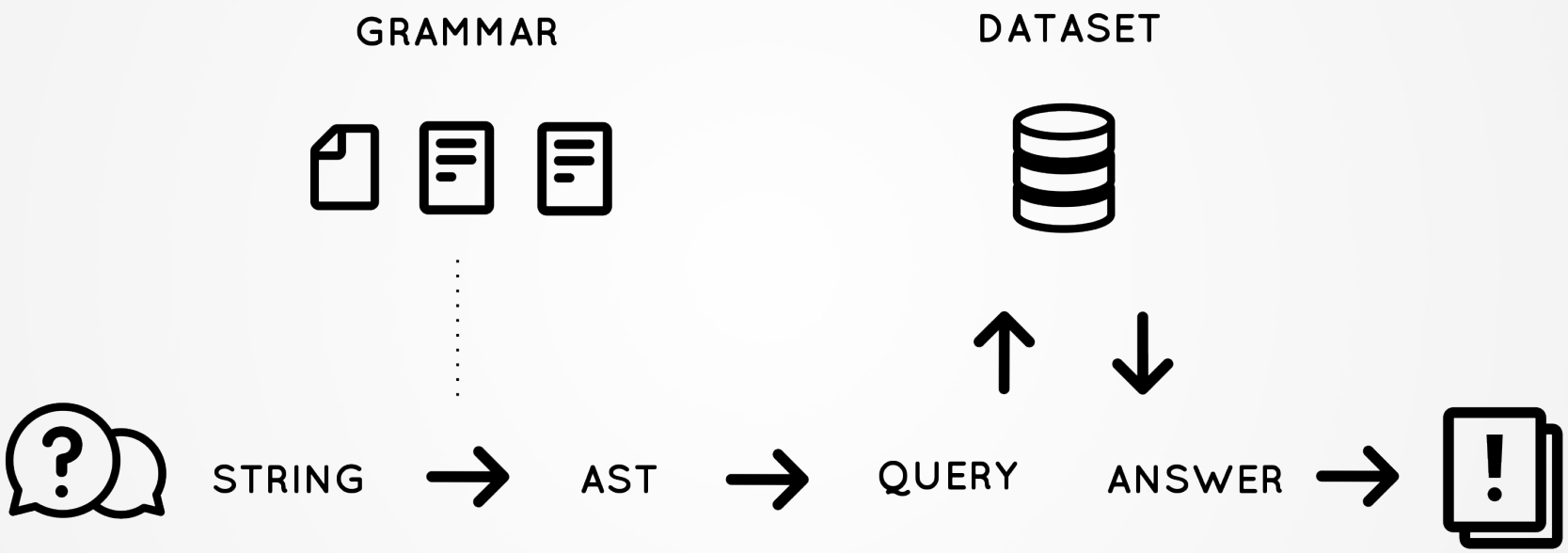
Outline

1. Schema-based grammars

2. From natural language to data
   - Transformation
   - Translation
   - Denotation

3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars




GRAMMAR
DATASET
STRING
AST
QUERY
ANSWER

Give me all astronauts.

↓

```
(queryAll Astronaut)
```

↓

```
SELECT ?x WHERE { ?x rdf:type onto:Astronaut . }
```

↓

```
{onto:YuriGagarin,onto:NeilArmstrong,…}
```

Is Yuri Gagarin an astronaut?

↓

```
(queryIs Astronaut Yuri)
```

↓

```
ASK WHERE { onto:YuriGagarin rdf:type
onto:Astronaut . }
```

↓

```
true
```

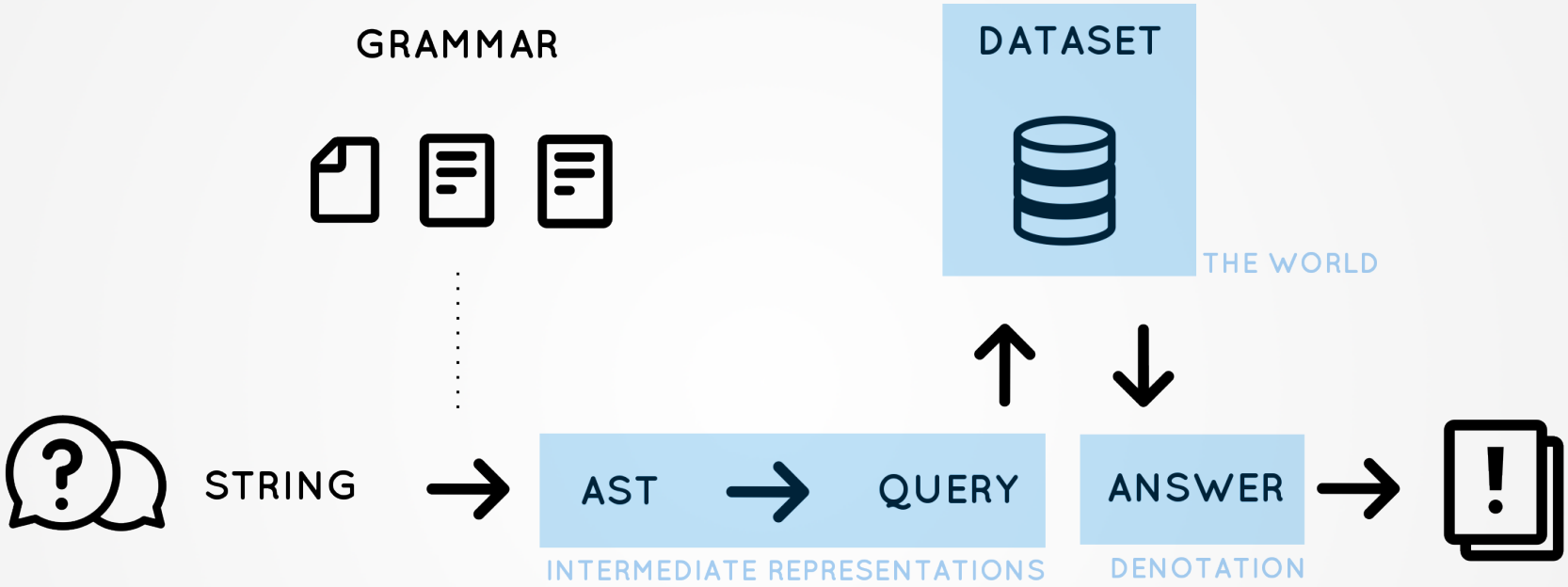
Natural language

↓

Logical representation

↓

Denotation in a model

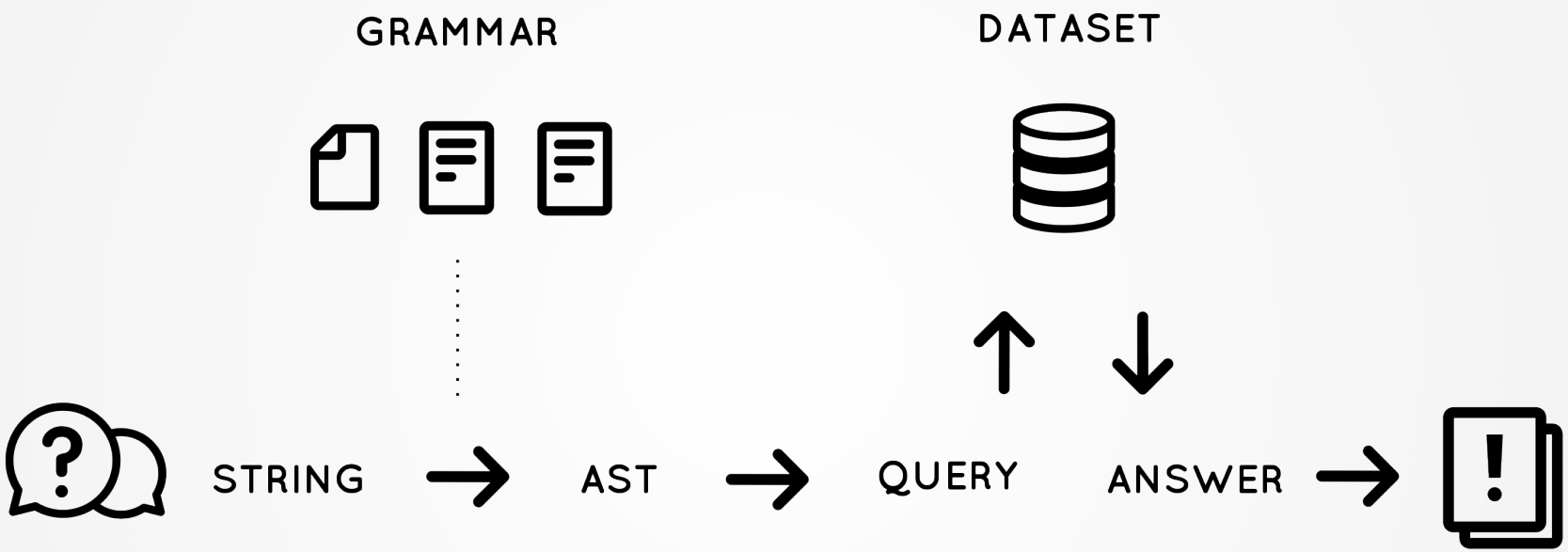

GRAMMAR
DATASET
THE WORLD
STRING
AST
QUERY
ANSWER
INTERMEDIATE REPRESENTATIONS
DENOTATION

Outline

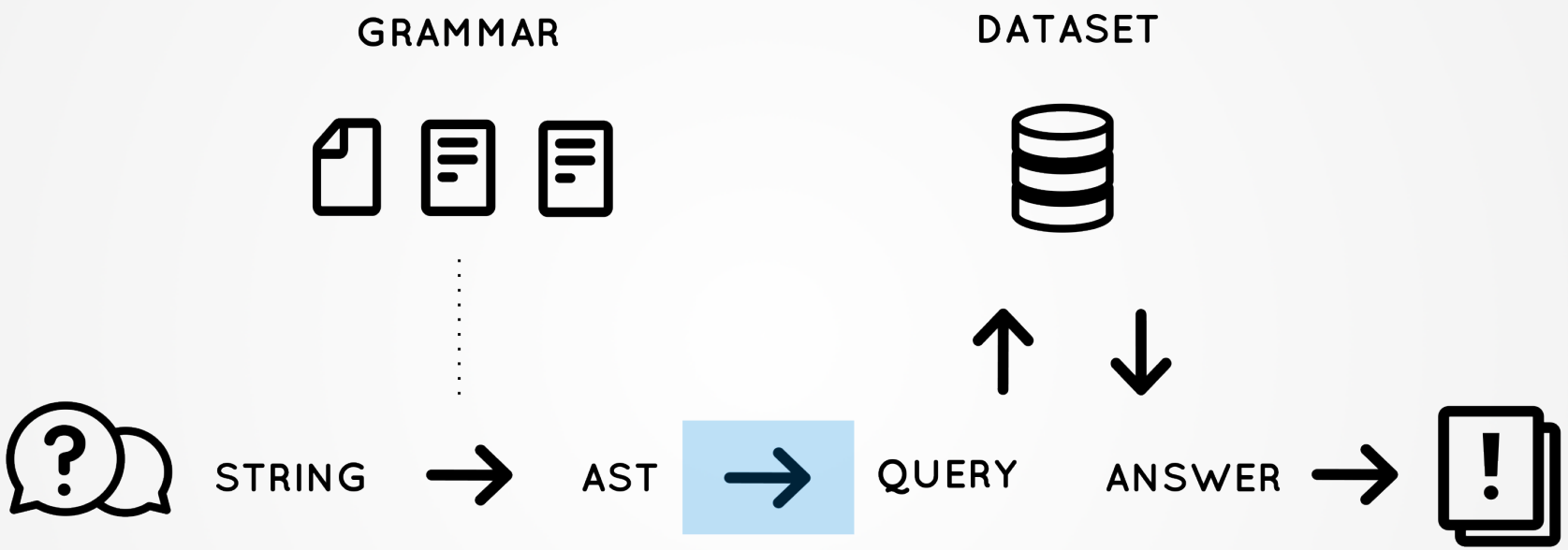
1. Schema-based grammars

2. From natural language to data
   - Transformation
   - Translation
   - Denotation

3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars


GRAMMAR
DATASET
STRING
AST
QUERY
ANSWER

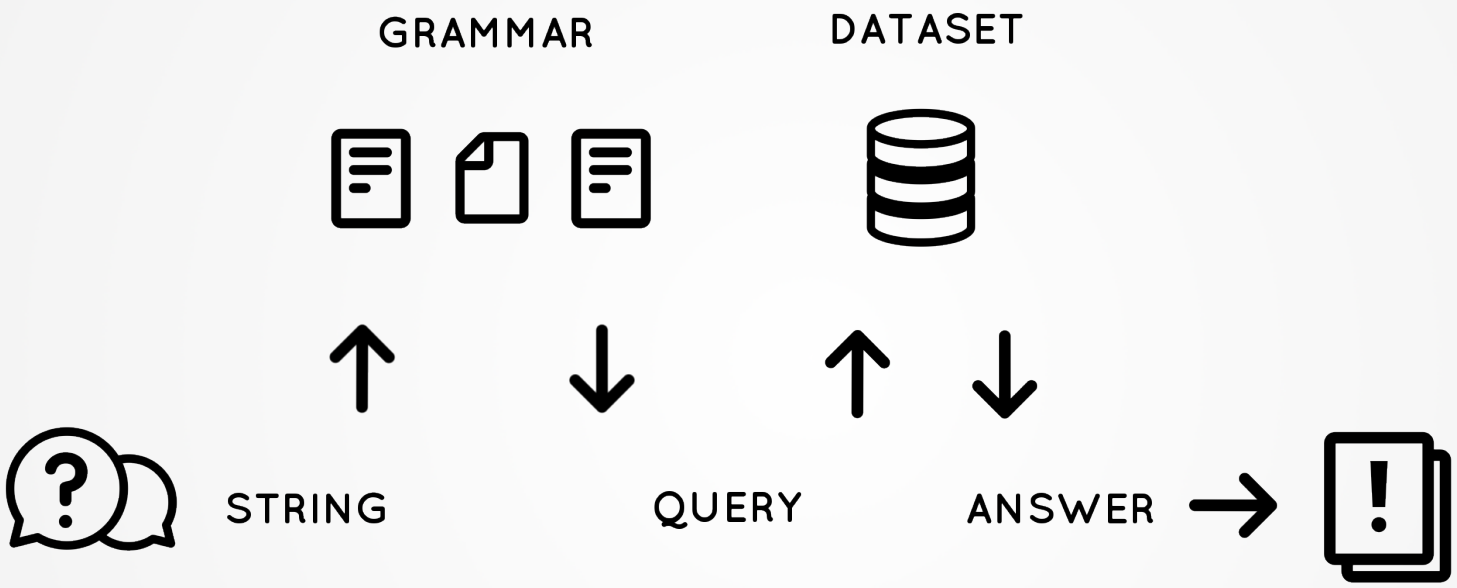
Transformation




```
transform :: AST -> Query
```

Outline

1. Schema-based grammars
2. From natural language to data
   - Transformation
   - Translation
   - Denotation
3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars


GRAMMAR
DATASET
STRING
QUERY
ANSWER

Grammar.gf

GrammarEng.gf

...

GrammarSPARQL.gf

```
abstract Grammar where {

cat

 Class;
 Entity Class;
 Relation Class Class;

 Statement;
 Question;

fun

 Astronaut, Planet, Moon : Class;

 YuriGagarin : Entity Astronaut;
 Jupiter     : Entity Planet;
 Europa      : Entity Moon;

 orbit : Relation Moon Planet;
}
```

```
abstract Grammar where {

cat

 Class;
 Entity Class;
 Relation Class Class;

 Statement;
 Question;

fun

 querySubj : (c1,c2 : Class) -> Relation c1 c2
                             -> Entity c2 -> Question;
 queryObj  : (c1,c2 : Class) -> Relation c1 c2
                             -> Entity c1 -> Question;
}
```

```
concrete GrammarEng where {

lincat

 Class     = CN;
 Entity    = NP;
 Relation  = V2;

 Statement = Cl;
 Question  = QCl;

lin

 Yuri  = mkNP (mkPN "Yuri Gagarin");
 Moon  = mkCN (mkN  "moon");
 orbit = mkV2 (mkV  "orbit");
}
```

```
concrete GrammarEng where {

lincat

 Class     = CN;
 Entity    = NP;
 Relation  = V2;

 Statement = Cl;
 Question  = QCl;

lin

 querySubj c _ r e2 = mkQCl (mkIP which_IDet c)
                            r e2;
 queryObj  _ c r e1 = mkQCl (mkIP which_IDet c)
                            (mkClSlash r e1);
}
```

```
concrete GrammarSPARQL where {

lincat

 Class     = Str;
 Entity    = Str;
 Relation  = Str;

 Statement = Str;
 Question  = Str;

lin

 Yuri  = "onto:YuriGagarin";
 Moon  = "onto:Moon";
 orbit = "onto:orbit";
```

```
concrete GrammarSPARQL where {

lincat

 Class     = Str;
 Entity    = Str;
 Relation  = Str;

 Statement = Str;
 Question  = Str;

lin

 querySubj c _ r e2 = "SELECT ?x WHERE {
                           ?x rdf:type " ++ c ++ "."
                       ++ "?x" ++ r ++ e2 ++ ". }";
 queryObj  c _ r e1 = "SELECT ?y WHERE {
                           ?y rdf:type " ++ c ++ "."
                        ++ e1 ++ r ++ "?y . }";
```

Which moons orbit Jupiter?

$\downarrow *^1$

```
(querySubj Moon Planet orbit Jupiter)
```

$\downarrow *^2$

```
SELECT ?x WHERE { ?x rdf:type onto:Moon .
    ?x onto:orbit onto:Jupiter . }
```

$\downarrow$

```
{onto:Europa,...}
```

$*^1$ `parse (from=Eng)`
$*^2$ `linearize (to=SPARQL)`

What difference does it make?

Specification of query language and system implementation are separated, i.e. extension to other query languages only requires a new concrete syntax, but no code changes.

What difference does it make?

Specification of query language and system implementation are separated, i.e. extension to other query languages only requires a new concrete syntax, but no code changes.

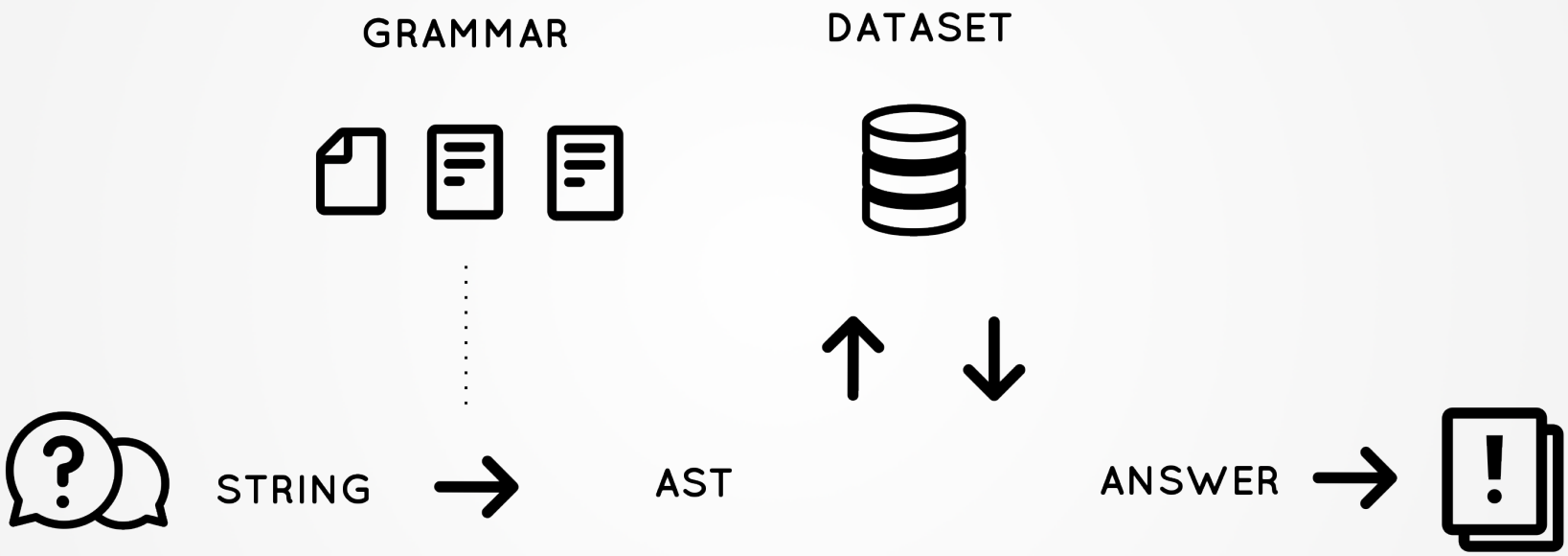
However, a processing step on ASTs might be required (e.g. for pronoun resolution, disambiguation, etc).

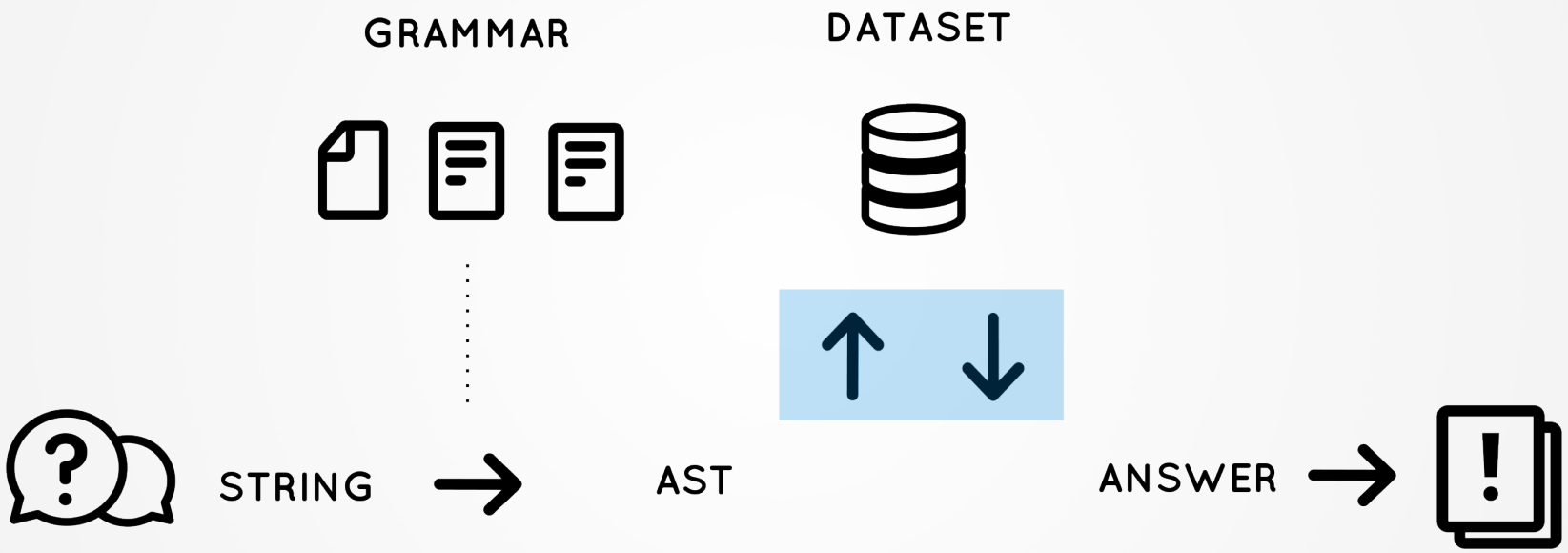
Outline

1. Schema-based grammars

2. From natural language to data
   - Transformation
   - Translation
   - Denotation

3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars


GRAMMAR
DATASET
STRING
AST
ANSWER


GRAMMAR
DATASET
STRING
AST
ANSWER

$$
\begin{aligned}
\texttt{Entity} &\rightarrow \texttt{e} \\
\texttt{Class} &\rightarrow \texttt{[e]} \\
\texttt{Relation} &\rightarrow \texttt{[(e,e)]} \\
\texttt{Statement} &\rightarrow \texttt{t}
\end{aligned}
$$

$$
\begin{aligned}
\text{Entity} &\rightarrow \text{e} \\
\text{Class} &\rightarrow [\text{e}] \\
\text{Relation} &\rightarrow [(\text{e},\text{e})] \\
\text{Statement} &\rightarrow \text{t}
\end{aligned}
$$

```
Yuri    = onto:YuriGagarin
Jupiter = onto:Jupiter
Europa  = onto:Europa
...
```

$$
\begin{aligned}
\text{Entity} &\rightarrow e \\
\text{Class} &\rightarrow [e] \\
\text{Relation} &\rightarrow [(e,e)] \\
\text{Statement} &\rightarrow t
\end{aligned}
$$

```
Astronaut = query [x] "?x rdf:type onto:Astronaut ."
          = [onto:Yuri,onto:Neil,...]

Planet    = query [x] "?x rdf:type onto:Planet ."
          = [onto:Earth,onto:Jupiter,...]
```

$$
\begin{aligned}
\text{Entity} &\rightarrow e \\
\text{Class} &\rightarrow [e] \\
\text{Relation} &\rightarrow [(e,e)] \\
\text{Statement} &\rightarrow t
\end{aligned}
$$

```
orbit = query [x,y] "?x onto:orbit ?y ."
      = [(Europa,Jupiter),...]

visit = query [x,y] "?x onto:visit ?y ."
      = []
```

$$
\begin{aligned}
\text{Entity} &\rightarrow e \\
\text{Class} &\rightarrow [e] \\
\text{Relation} &\rightarrow [(e,e)] \\
\text{Statement} &\rightarrow t
\end{aligned}
$$

```
isA : (c : Class) -> Entity c -> Statement;

isA_denotation :: [e] -> e -> t
isA_denotation es e = e `elem` es

app : (c1,c2 : Class) -> Relation c1 c2
   -> Entity c2 -> Entity c2 -> Statement;

app_denotation :: [(e,e)] -> e -> e -> t
app_denotation r e1 e2 = (e1,e2) `elem` r
```

Yuri Gagarin is an astronaut.

$$\downarrow$$

$$[\![(\texttt{isA Astronaut Yuri})]\!]$$
$$=$$
$$(([\![\texttt{isA}]\!]\ [\![\texttt{Astronaut}]\!])\ [\![\texttt{Yuri}]\!])$$
$$=$$
$$[\![\texttt{Yuri}]\!] \in [\![\texttt{Astronaut}]\!]$$
$$=$$
$$\texttt{onto:Yuri} \in \texttt{[onto:Yuri,onto:Neil,...]}$$

$$\downarrow$$

$$\texttt{true}$$

What difference does it make?

Coverage is not restricted to the expressiveness of the query language.

Outline

1. Schema-based grammars

2. From natural language to data
   - Transformation
   - Translation
   - Denotation

3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars

Outline

1. Schema-based grammars
2. From natural language to data
   - Transformation
   - Translation
   - Denotation
3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars

Quering cultural heritage data

Demo: http://museum.ontotext.com

Dana Dannélls, Ramona Enache, Mariana Damova:
A Multilingual SPARQL-Based Retrieval Interface for Cultural Heritage Objects. http://ceur-ws.org/Vol-1272/paper_12.pdf

Querying biomedical linked data

http://cs-gw.utcluj.ro/~anca/GFMed/index.html

Anca Marginean: GFMed: Question answering over biomedical linked data with Grammatical Framework. ceur-ws.org/Vol-1180/CLEF2014wn-QA-Marginean2014.pdf

Anca Marginean: Romanian2SPARQL: A Grammatical Framework approach for querying Linked Data in Romanian. http://dx.doi.org/10.1109/DAAS.2014.6842456

Multimodal querying

```
 cat
    Query → { nlang : Str }
    Click → { click : Str }
    Input → { nlang : Str ; click : Str }
fun
    ...
    PClick : Click -> Place
    PClick c = { nlang = "here" ; c.s }
```

Björn Bringert, Peter Ljunglöf, Aarne Ranta, Robin Cooper: Multimodal Dialogue Systems Grammar. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.119.7396

Demo: https://www.youtube.com/watch?v=1bfaYHWS6zU

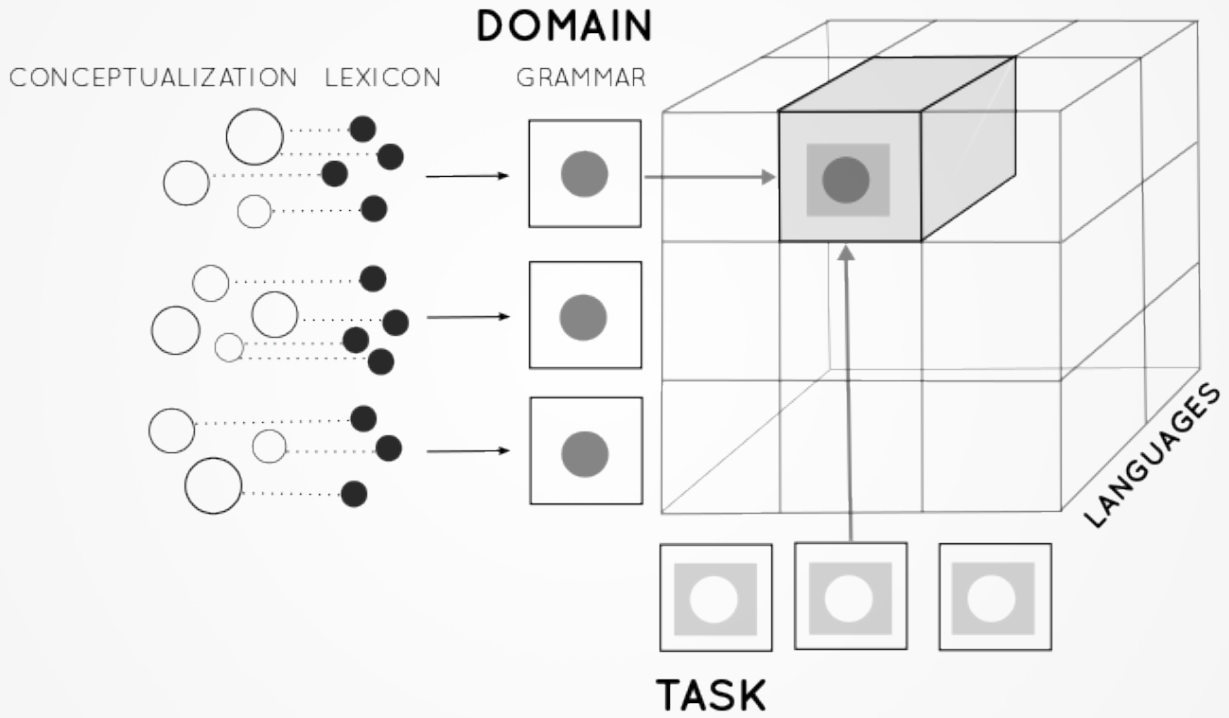
Outline

1. Schema-based grammars
2. From natural language to data
   - Transformation
   - Translation
   - Denotation
3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars

YAQL (Yet Another Query Language)

```
svn checkout svn://molto-project.eu/wp4/YAQL
```

Modular application grammars



Application = (Core +) Domain + Task + Language

Outline

1. Schema-based grammars
2. From natural language to data
   - Transformation
   - Translation
   - Denotation
3. Examples and applications
   - GF-based query systems
   - Application grammars for querying
   - Generating application grammars

Generating application grammars from ontology lexica

https://github.com/cunger/lemongrass