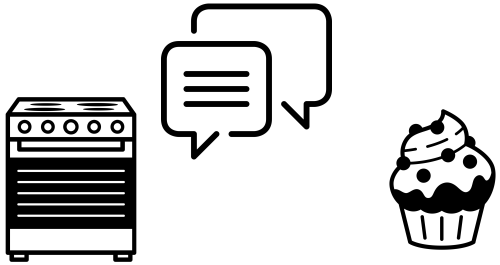
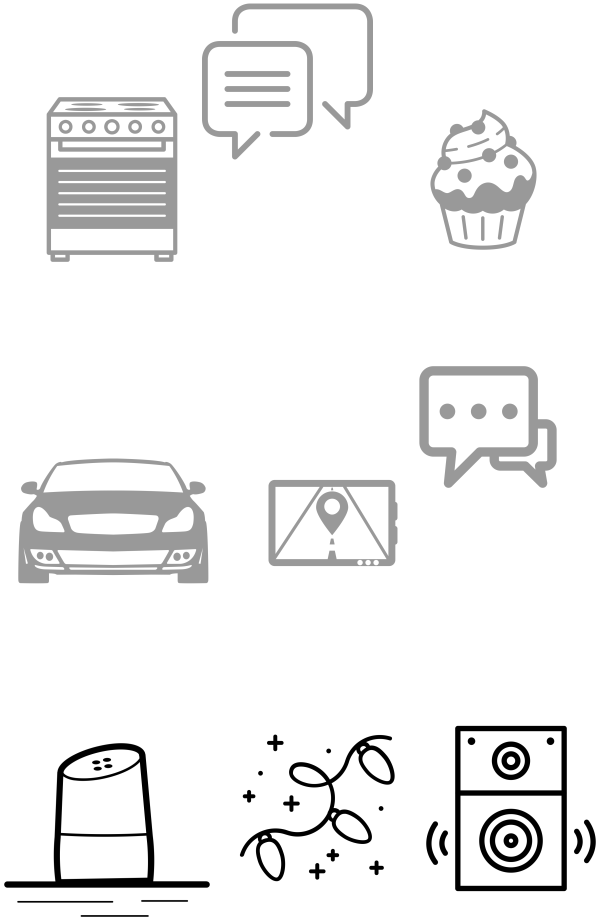
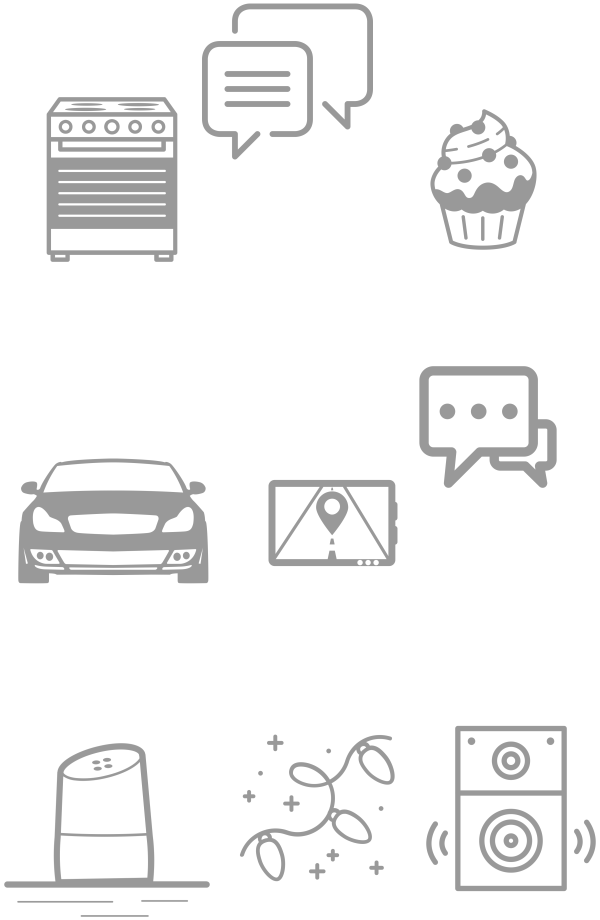
APPLICATION GRAMMARS

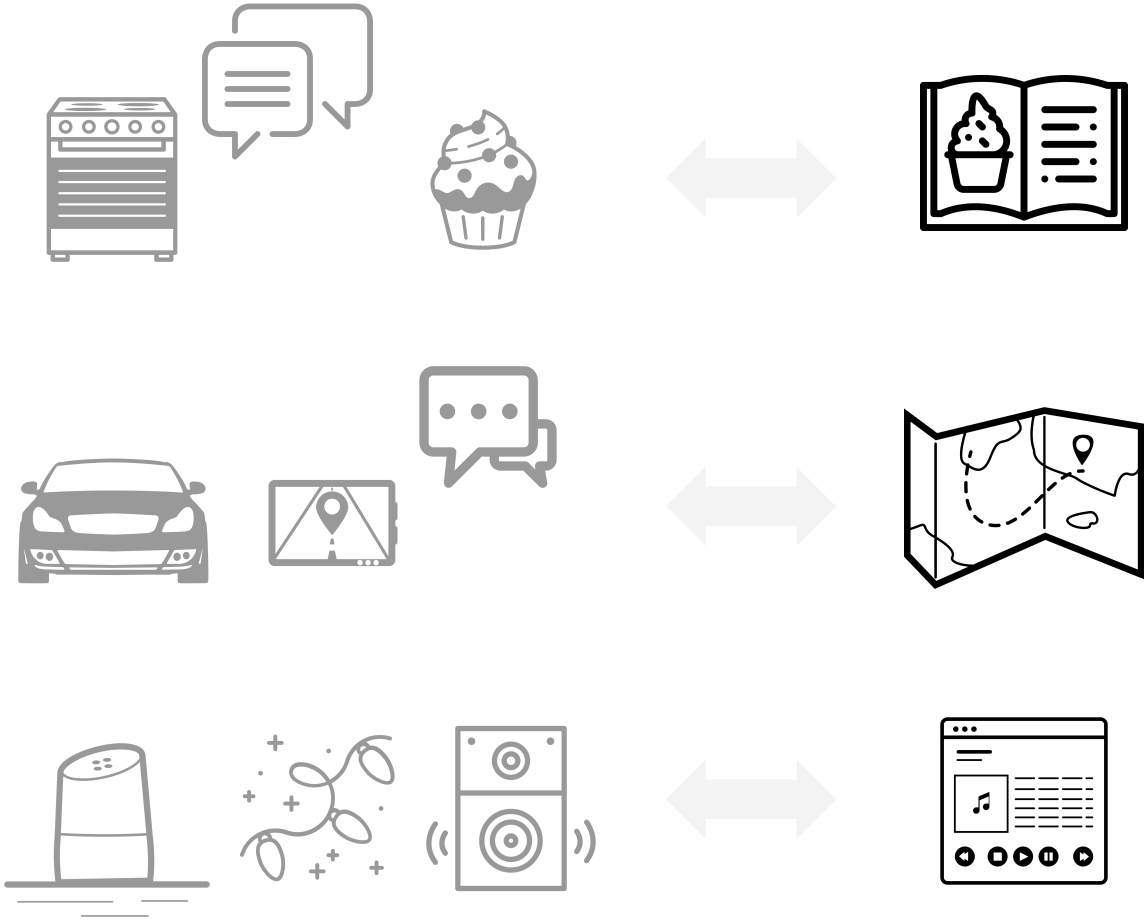
CHRISTINA UNGER
(MERCURY.AI)

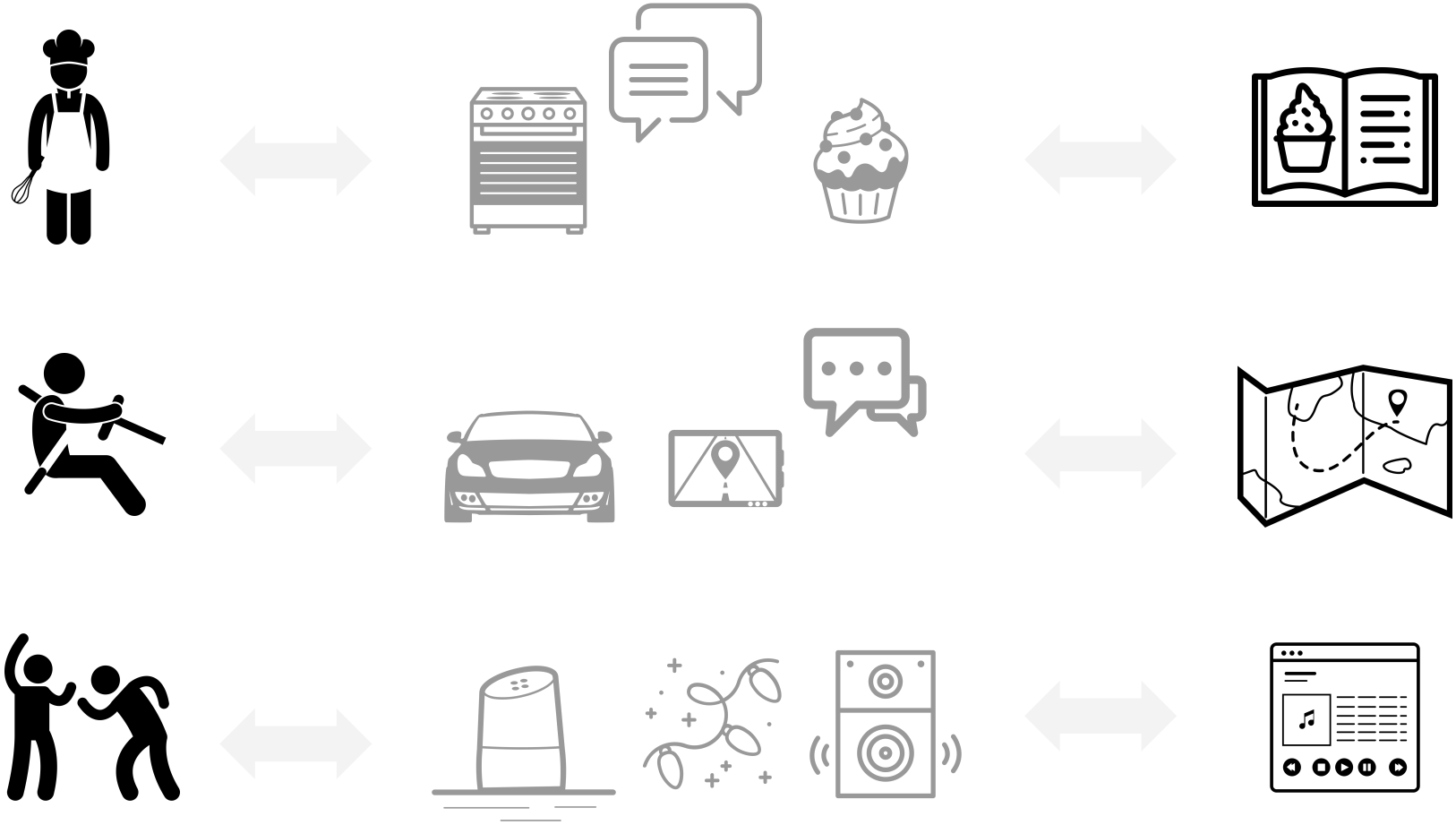
EXAMPLE: NATURAL LANGUAGE INTERFACES



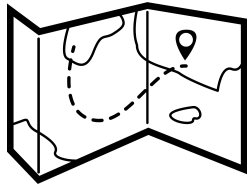








NLI applications are intermediaries between users and data.



They need to speak

- the language of the user
- the language of the data

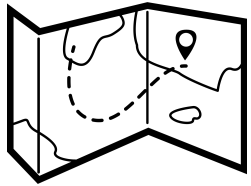
Their job is to translate between both.

NLI applications are intermediaries between users and data.

They need to speak

- the language of the user
- the language of the data

USUALLY EASY (STRUCTURED AND MACHINE-READABLE)



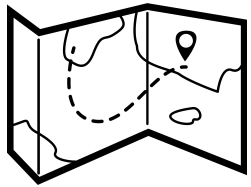
Their job is to translate between both.

NLI applications are intermediaries between users and data.

They need to speak

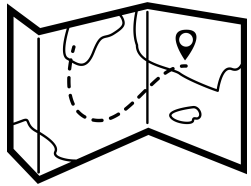
- the language of the user
- the language of the data

MUCH HARDER
(NATURAL LANGUAGE)



Their job is to translate between both.

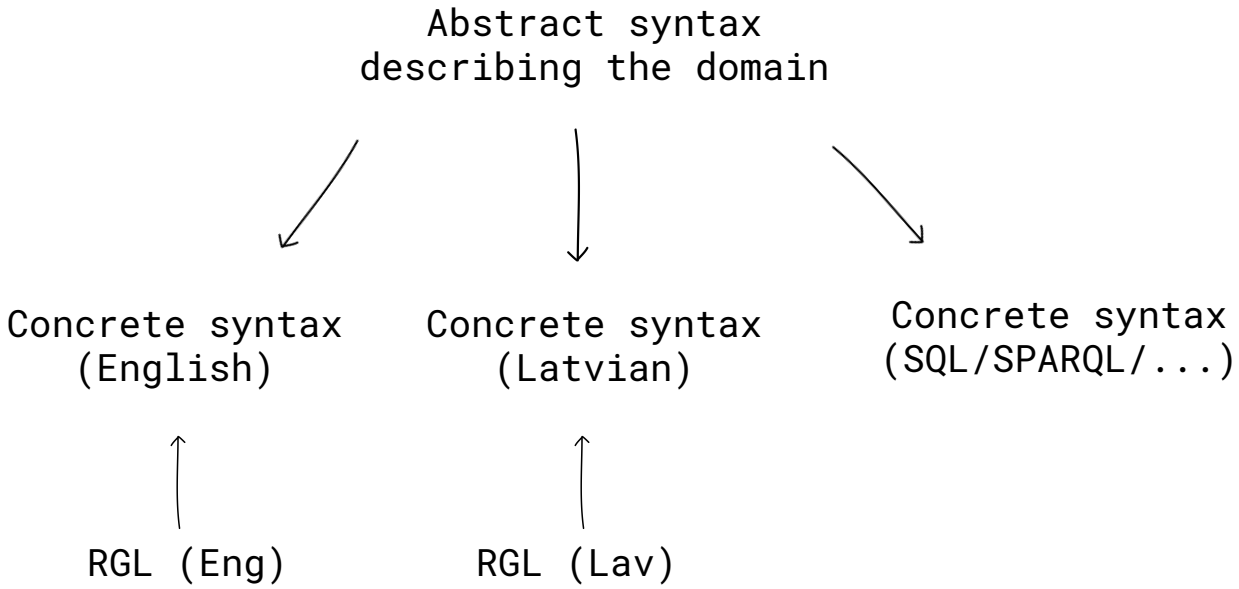
NLI applications are intermediaries between users and data.



They need to speak

- the language of the user
- the language of the data

Their job is to translate between both.


Abstract syntax
describing the domain
Concrete syntax
(English)
Concrete syntax
(Latvian)
Concrete syntax
(SQL/SPARQL/...)
RGL (Eng)
RGL (Lav)

Application grammars are top-level grammars, the RGL is a library.

The **Resource Grammar Library** was developed to take care of **"low-level" linguistic rules** such as inflection, agreement, and word order.

This enables the authors of **application grammars** to **focus on the semantics** when designing the abstract syntax.

`https://www.grammaticalframework.org/lib/doc/translation.html`

THE RGL AS TOP-LEVEL GRAMMAR FOR MACHINE TRANSLATION: TOMORROW

MAIN PROBLEM:

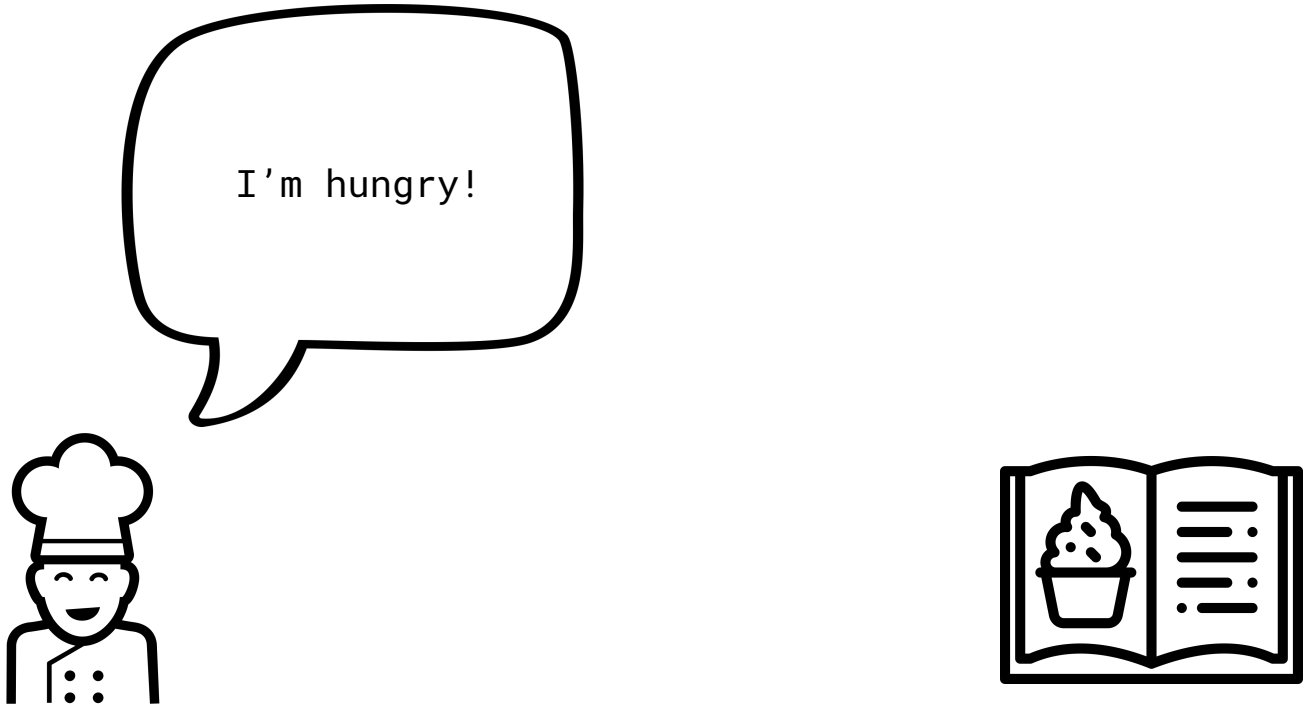
THE RGL IS LOW-LEVEL SYNTAX-ORIENTED.

- It lacks a **level of abstraction**, e.g. to facilitate aligning natural language with data.
- **Semantic distinctions** are assumed to be defined in application grammars. The RGL defines the combinatorics of elements, but doesn't specify which elements can really go together.
- RGL parsing creates **spurious syntactic ambiguities**.

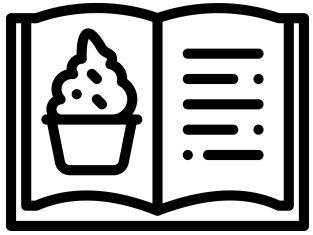
HANDS-ON:
BUILDING AN APPLICATION GRAMMAR

STEP 1: SCOPE

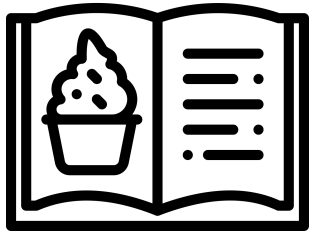
STEP 1: SCOPE



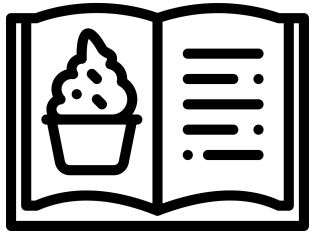
STEP 1: SCOPE



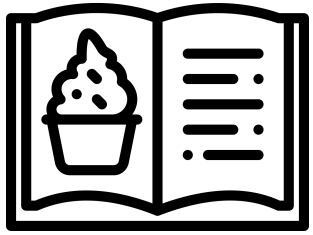
STEP 1: SCOPE



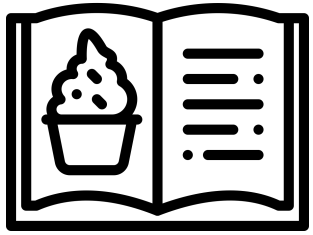
STEP 1: SCOPE



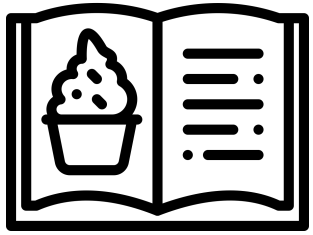
STEP 1: SCOPE



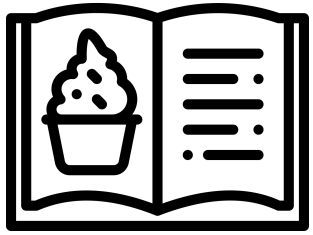
STEP 1: SCOPE



STEP 1: SCOPE



STEP 1: SCOPE



STEP 1: SCOPE

Grouping examples

Recipe Search

I'm hungry.
Any burger recipes?
Fast and healthy please.
What can I do with papayas?
I'm still hungry.

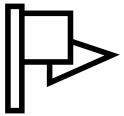
Recipe INFO

Does this contain peanuts?
For how many people is this?
What do I need?
How many carbs does it have?
Is this vegetarian?

User Preferences

I hate garlic.
I'm vegetarian.
I'm allergic to peanuts.
I like cheese.
I try to eat low-carb.

Version 1



Writing application grammars is inherently **domain-driven**: All important choices depend on the scope and requirements of the application.

`https://gist.github.com/cunger/1e5d9e404c6979fc45cdf366b52562e1`

Version 1

+ **good level of abstraction**

- **doesn’t generalize across domains**
  observation: verbalization structures are usually the same across similar domains, it’s mostly the lexical items that differ

Version 2 (ontological heaven)

```
abstract Search = {

  cat

    Kind;

    Term Kind;
    Entity Kind;
    Attribute Kind;
    Relation Kind Kind;

    Search;
    SearchFilter;


+  functions for composition
```

Version 2 (ontological heaven)

```
abstract SearchForRecipes = Search ** {

  fun

    Ingredient, Recipe : Kind;

    pizza, burger, dessert : Term Recipe;
    tomato, cheese, peanut : Term Ingredient;

    spaghetti_bolognese, pizza_hawaii : Entity Recipe;

    vegetarian, fast, easy, healthy : Attribute Recipe;

    with    : Relation Ingredient Recipe;
    without : Relation Ingredient Recipe;
}
```

Version 2 (ontological heaven)

```
abstract SearchForCars = Search ** {

  fun

    Car, Equipment : Kind;

    porsche_cayenne  : Entity Car;
    convertible, suv : Term   Car;

    child_seat, air_conditioning : Term Equipment;

    fast, cheap : Attribute Car;

    with    : Relation Equipment Car;
    without : Relation Equipment Car;
}
```

Version 2 (ontological heaven)

```
abstract SearchForMusic = Search ** {

  fun

    Song, Album, Artist : Kind;

    freddy_mercury : Entity Artist;
    made_in_heaven : Entity Album;
    bicycle_race   : Entity Song;

    relaxed, fast, heavy : Attribute Song;

    song_by  : Relation Song  Artist;
    album_by : Relation Album Artist;
    contains : Relation Song  Album;
}
```

Yet Another Query Language (YAQL)

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

Version 2 (ontological heaven)

+ **generalizes across domains**
+ thus easy to re-use grammar parts
+ **tailored towards alignment with data**

- strong semantic orientation leads to a **cat/lincat mismatch**

Version 2 (ontological heaven)

```
abstract Search = {

  cat

    Kind;

    Term Kind;
    Entity Kind;
    Attribute Kind;
    Relation Kind Kind;

    Search;
    SearchFilter;
```

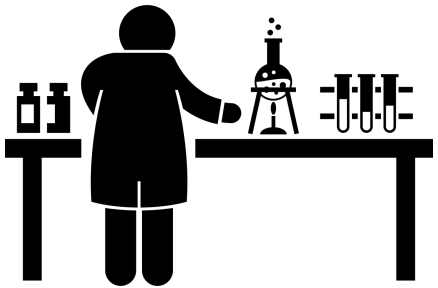
Version 2 (ontological heaven)

```
abstract Search = {

  cat

    Kind;

    Term Kind;            -- CN
    Entity Kind;          -- NP
    Attribute Kind;       -- AP, Adv, RCl
    Relation Kind Kind;   -- V2, N2, A2

    Search;
    SearchFilter;
```

Version 2 (ontological heaven)

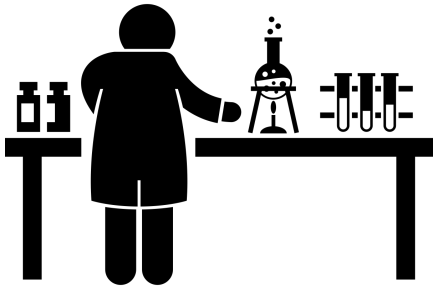
```
abstract Search = {

  cat

    Kind;

    Term Kind;            -- CN
    Entity Kind;          -- NP
    Attribute Kind;       -- { ap : AP, adv : Adv, rcl : RCl }
    Relation Kind Kind;   -- { v2 : V2, n2  : N2,  a2  : A2 }

    Search;
    SearchFilter;
```

Version 2 (ontological heaven)

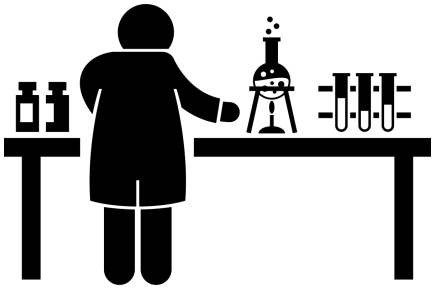


English

Version 2 (ontological heaven)



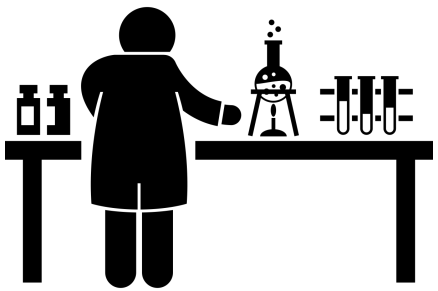
English



German

(or pick your favourite morphologically rich language)

Version 2 (ontological heaven)



English

German

(or pick your favourite morphologically rich language)

Version 2 (ontological heaven)

```
abstract Search = {

  cat

    Kind;

    Term Kind;            -- CN
    Entity Kind;          -- NP
    Attribute Kind;       -- { ap : AP, adv : Adv, rcl : RCl }
    Relation Kind Kind;   -- { v2 : V2, n2  : N2,  a2  : A2 }

    Search;
    SearchFilter;
```

Version 2 (ontological heaven)

```
abstract Search = {

  cat

    Attribute_AP  Kind;      -- AP
    Attribute_Adv Kind;      -- Adv
    Attribute_RCl Kind;      -- RCl

    Relation_V2 Kind Kind;   -- V2
    Relation_N2 Kind Kind;   -- N2
    Relation_A2 Kind Kind;   -- A2
```

Version 2 (ontological heaven)

```
abstract Search = {

  cat

    Attribute_AP  Kind;      -- AP
    Attribute_Adv Kind;      -- Adv
    Attribute_RCl Kind;      -- RCl

    Relation_V2 Kind Kind;   -- V2
    Relation_N2 Kind Kind;   -- N2
    Relation_A2 Kind Kind;   -- A2
```

\+ **flat**, no explosion

\- **duplication of composition rules**
(imagine you have several *_AP and *_CN categories and want to have AP-CN-modification)

Version 3 (Syntax-Oriented)

```
abstract Search = {

  cat

    Noun;              -- CN
    NounPhrase;        -- NP

    AdjectivePhrase;   -- AP

    VerbPhrase;        -- VP
    Adverb;            -- Adv

    Clause;            -- Cl
```

Version 3 (Syntax-Oriented)

```
abstract Search = {

  cat

    Noun;              -- CN
    NounPhrase;        -- NP

    AdjectivePhrase;   -- AP

    VerbPhrase;        -- VP
    Adverb;            -- Adv

    Clause;            -- Cl
```

\+ **perfect correspondence** between cats and lincats

Version 3 (Syntax-Oriented)

```
abstract Search = {

  cat

    Noun;              -- CN
    NounPhrase;        -- NP

    AdjectivePhrase;   -- AP

    VerbPhrase;        -- VP
    Adverb;            -- Adv

    Clause;            -- Cl
```

\+ **perfect correspondence** between cats and lincats

\- plain duplication of the API
\- and where did the **semantics** go?? (syntax-orientation is not bad, but it's also not enough)

Version 4

```
abstract RecipeSearch = {

  cat

    IngredientMassNoun;
    IngredientCountNoun;

    NounPhrase;
    NounPhrase_Neg;
    NounPhrase_NPI;
    NounPhrase_PPI;

    ...
```

```
abstract RecipeSearch = {

  cat

    IngredientMassNoun;
    IngredientCountNoun;

    NounPhrase;
    NounPhrase_Neg;
    NounPhrase_NPI;
    NounPhrase_PPI;

    ...
```

〉flat

〉syntax-oriented

〉grammatic and semantic distinctions as needed

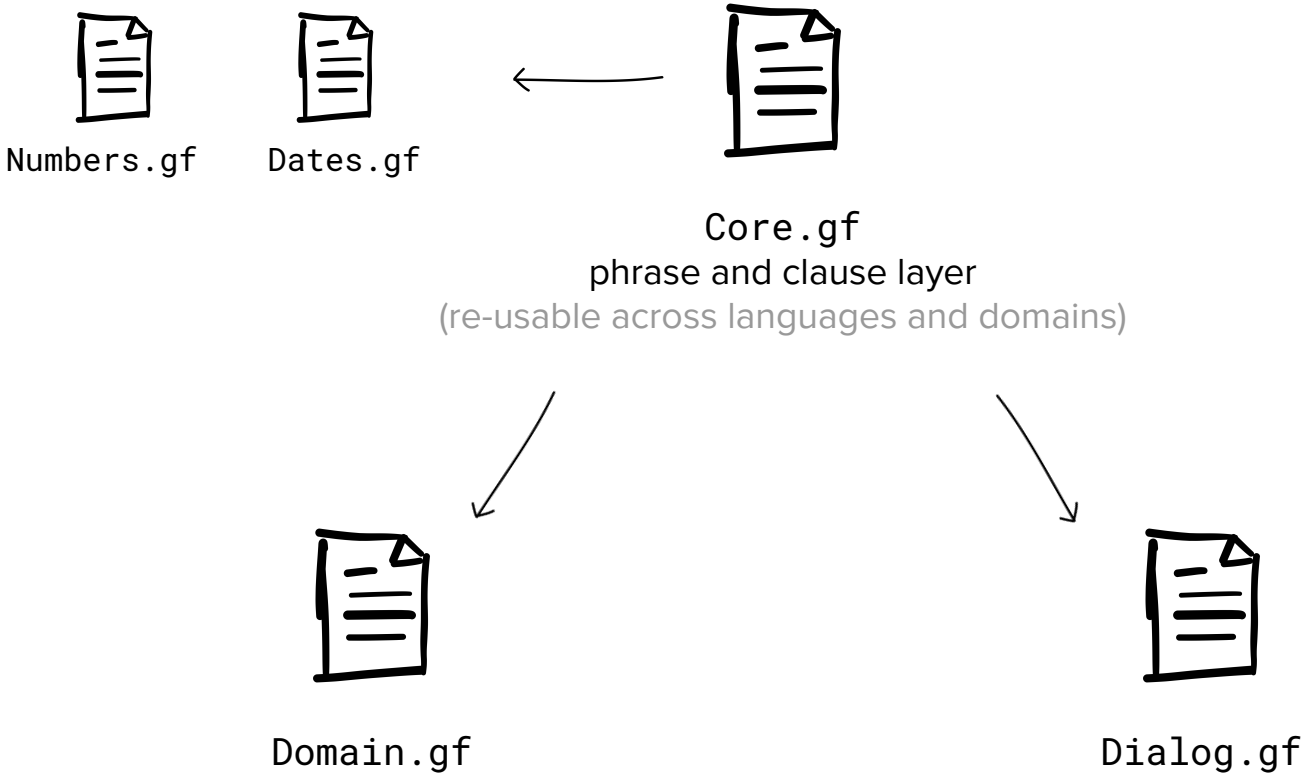
```
abstract RecipeSearch = {

  cat

    IngredientMassNoun;
    IngredientCountNoun;

    NounPhrase;
    NounPhrase_Neg;
    NounPhrase_NPI;
    NounPhrase_PPI;

    ...
```

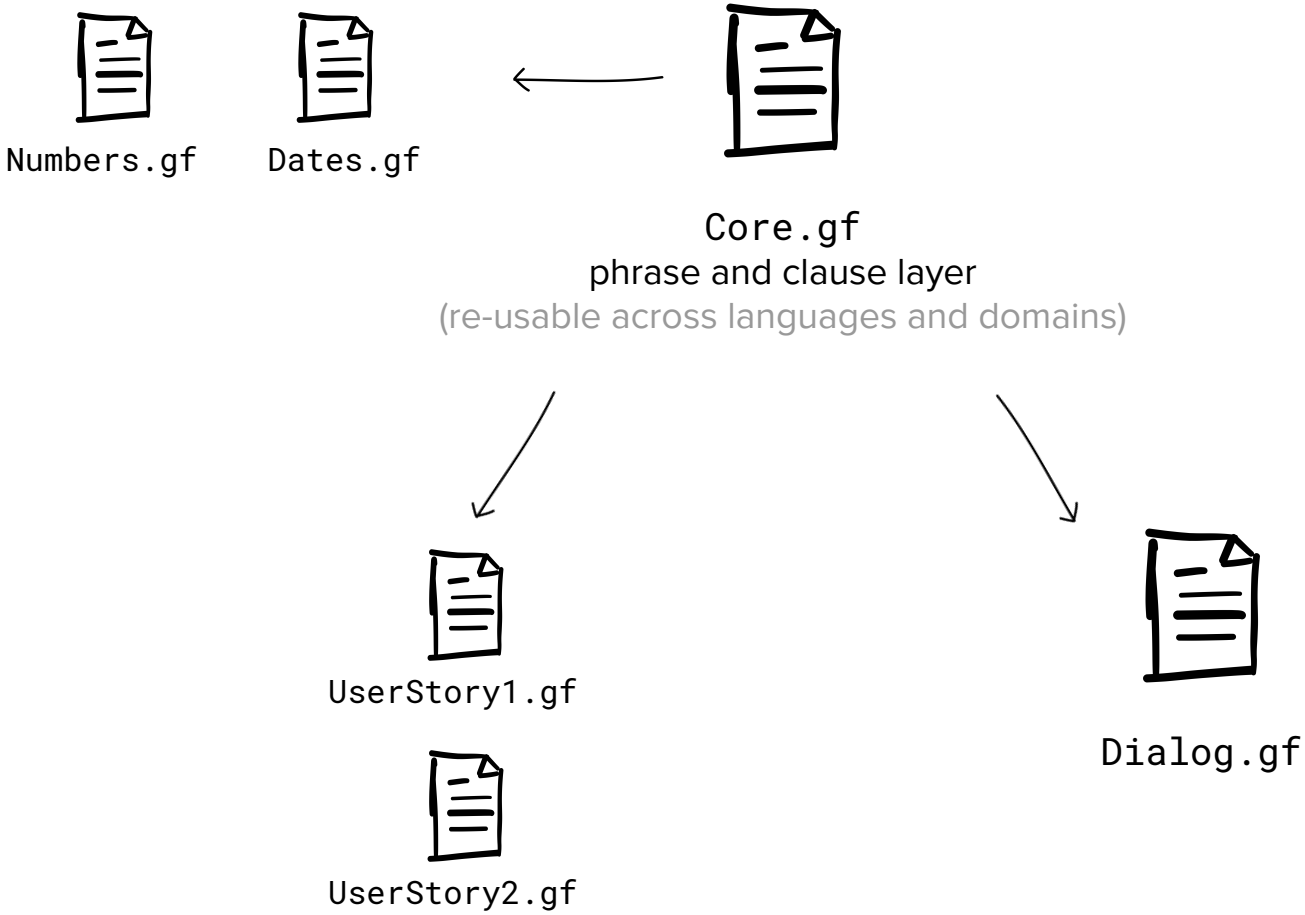
〉flat

〉syntax-oriented

〉grammatic and semantic distinctions as needed

〉modular


Numbers.gf
Dates.gf
Core.gf
phrase and clause layer
(re-usable across languages and domains)
Domain.gf
lexical items and constructrions
Dialog.gf


Numbers.gf
Dates.gf
Core.gf
phrase and clause layer
(re-usable across languages and domains)
UserStory1.gf
UserStory2.gf
Dialog.gf