On the syntax and semantics of voice assistants in autonomous vehicles

Warrick Macmillan

6^{th} May 2022

Motivation

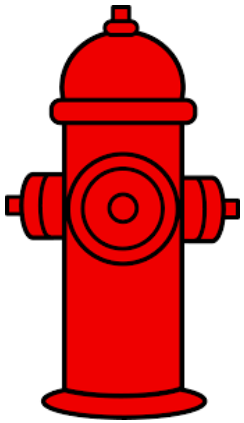
"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."

Motivation

"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."

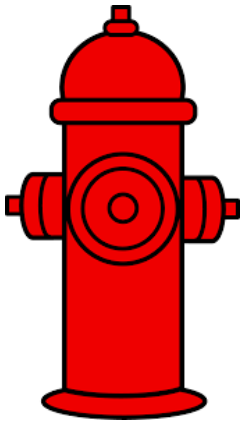
Motivation

"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."



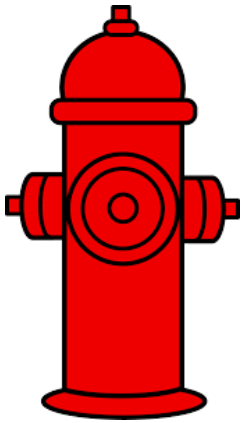
Motivation

"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."



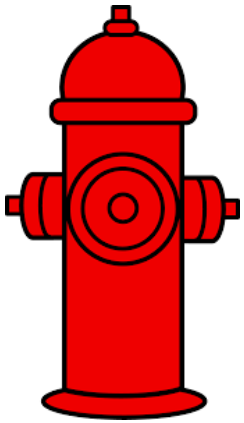
Motivation

"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."



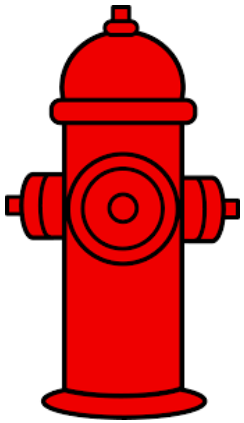
Motivation

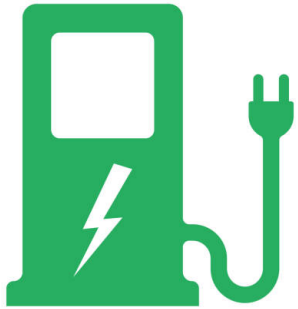
"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."



Motivation

"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."





Ambiguities

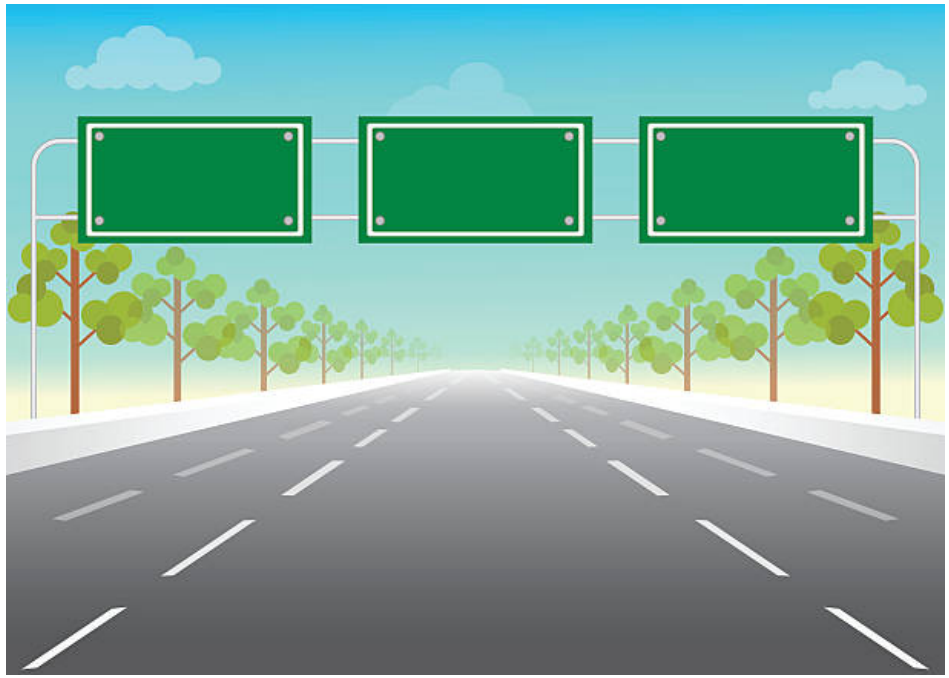
"Go into the other lane"

Ambiguities

"Go into the other lane"

Ambiguities

"Go into the other lane"



Ambiguities (cont.)

"Drive to the person with the dog"

Ambiguities (cont.)

"Drive to the person with the dog"

Ambiguities (cont.)

"Drive to the person with the dog"

Simplified Autonomous Vehicle

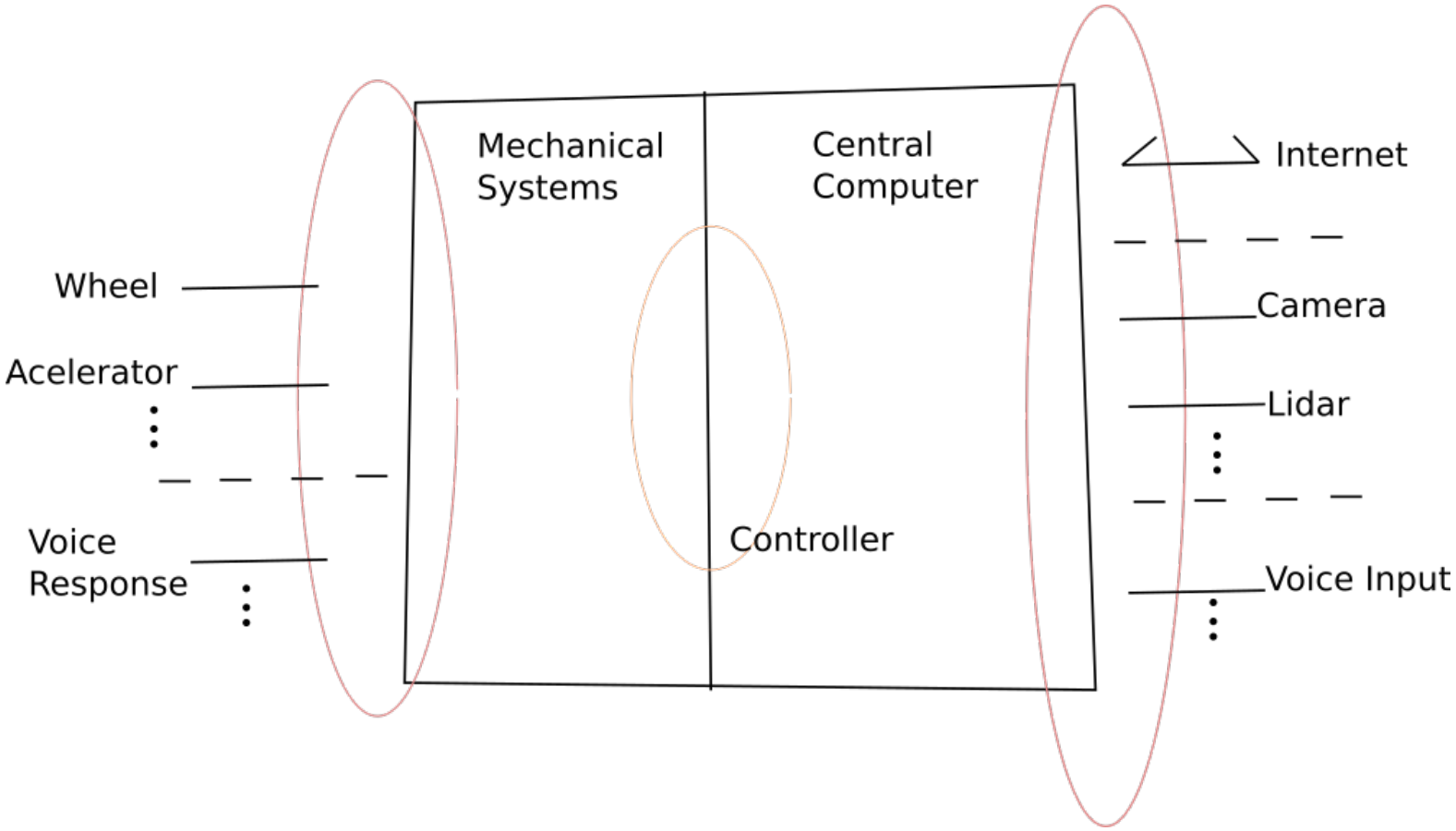


Figure: Self-driving car

Path

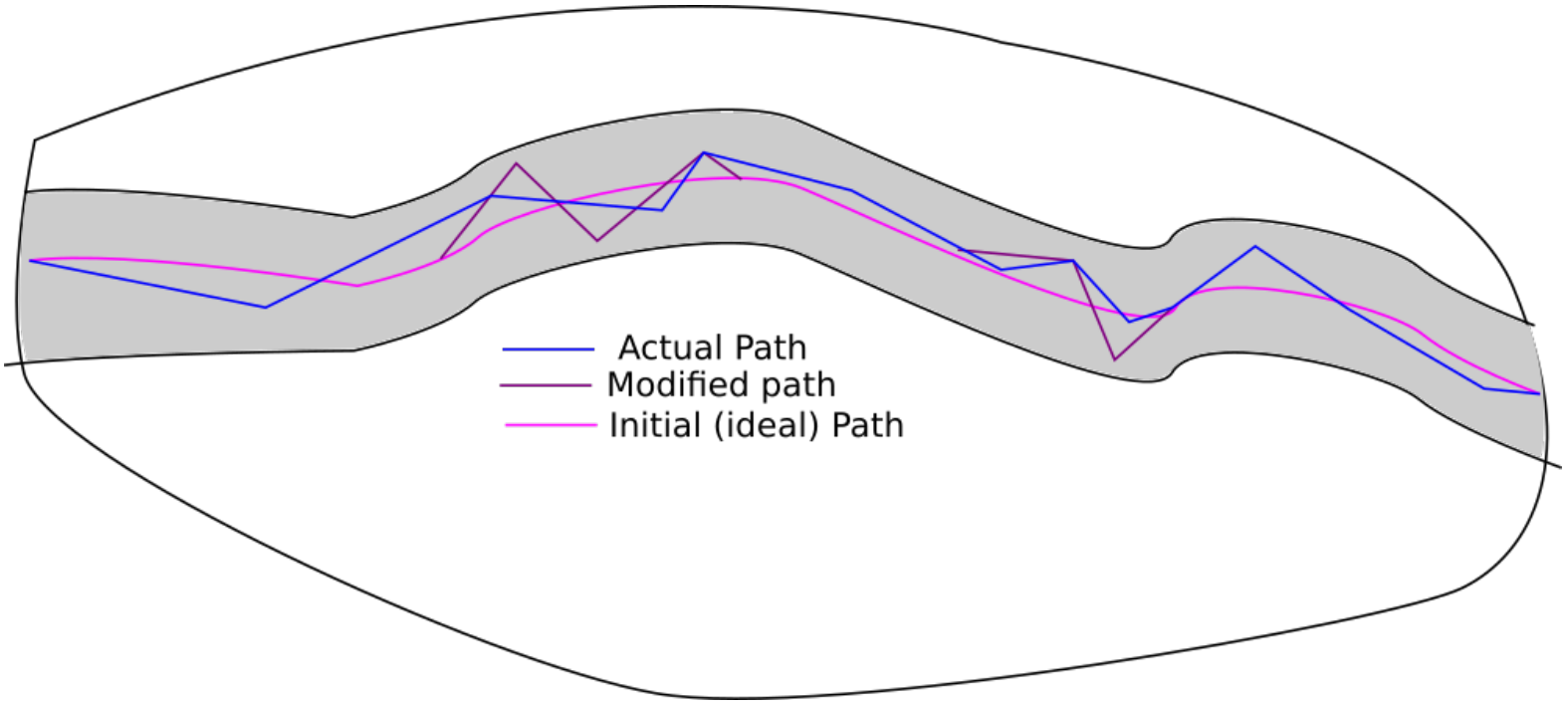


Figure: Initial, modified, and actual paths/routes

Mathematically Ideal Property

$\forall u \in \text{Utt}\ \exists r \in \text{Routes}$ such that $\forall r' \in \text{Routes}\ d(u, r) \leq d(u, r')$

Mathematically Ideal Property

$\forall u \in \text{Utt}\ \exists r \in \text{Routes}$ such that $\forall r' \in \text{Routes}\ d(u,r) \leq d(u,r')$

where d is some hypothetical metric which should depend on

- Cost
- Safety
- Legality
- Adversaries

Mathematically Ideal Property

$\forall u \in \text{Utt}\ \exists r \in \text{Routes}$ such that $\forall r' \in \text{Routes}\ d(u, r) \leq d(u, r')$

where d is some hypothetical metric which should depend on

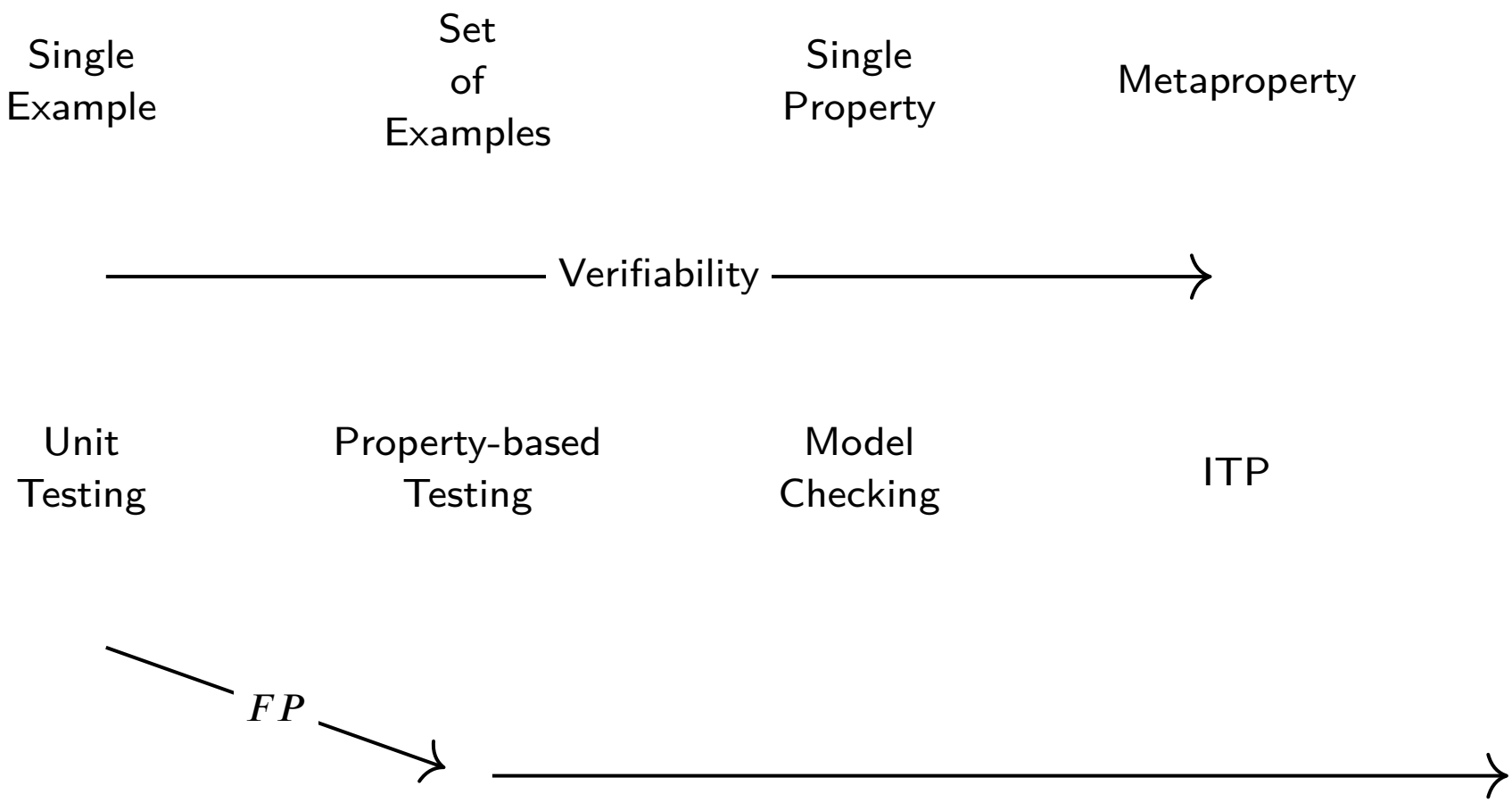
- Cost
- Safety
- Legality
- Adversaries

and where

- The set of utterances is a set of strings over a standard alphabet
- The set of routes is some discretized set of paths in Euclidean 2-space

subject to constraints on the sets imposed by, for example

- grammars in the case of strings
- or physical objects in the case of paths in Euclidean space


Single Example
Set of Examples
Single Property
Metaproperty
Verifiability
Unit Testing
Property-based Testing
Model Checking
ITP
FP

Main Idea

Functional Programming is all about composition of higher order functions
Break down system into composable components, each of which can operate under different verification conditions

Trade-offs

- Depth versus breadth

Trade-offs

- Depth versus breadth
- Specificity and generality

Trade-offs

- Depth versus breadth
- Specificity and generality
- Formality and formalizability

Trade-offs

- Depth versus breadth
- Specificity and generality
- Formality and formalizability
- Verifiability and validity

Ontological Categories

```
cat
  PosCommand    ; -- go to the store
  Place         ; -- the store
  Time          ; -- in 5 minutes
  Action        ; -- drive
  Way           ; -- to
  How           ; -- quickly
  Where         ; -- left
  AdvPh         ; -- to the store
  UndetObj      ; -- store
  Determ        ; -- the
  Object        ; -- the store
  Number        ; -- a
  Conjunct      ; -- and
  Condition     ; -- there is a museum
  Descript      ; -- big
```

go to the store, turn left and stop at the woman with the dog. go to the bridge. finish.

```
p " go to the store , turn left  and stop at the woman with the dog . go to the bridge . Finish ." | tt
* ConsCommands
   * OneCommand
      * CompoundCommand
         * And
           ConsPosCommand
            * SimpleCom
               * ModAction
                  * Go
                    MkAdvPh
                     * To
                       WhichObject
                        * The
                          Store
             BasePosCommand
              * SimpleCom
                 * ModAction
                    * Turn
                      WherePhrase
                       * Left
               SimpleCom
                * ModAction
                   Stop
                   MkAdvPh
                    * At
                      WhichObject
                       * The
                         PhraseModObj
                          * Woman
                            MkAdjPh
                             * With
                               WhichObject
                                * The
                                  Dog
    ConsCommands
     * OneCommand
        * SimpleCom
           * ModAction
              * Go
                MkAdvPh
                 * To
                   WhichObject
                    * The
                      Bridge
      BaseCommands
       * OneCommand
          * Finish
```

go to the store, turn left and stop at the woman with the dog. go to the bridge. finish.

go to the store, turn left and stop at the woman with the dog. go to the bridge. finish.

GF AST

GF AST

```
WhichObject
  * The
    Store

WherePhrase
  * Left

PhraseModObj
  * Woman
    MkAdjPh
      * With
        WhichObject
          * The
            Dog

WhichObject
```

This project is quite multifaceted, still in a somewhat primordial state (and therefore may be taken in many directions).

Goal : Design a controlled natural language which is

- Suitable as an "approximation" for a voice assistant for an autonomous vehicle
- Has a well defined semantics in temporal logic
- Seeking to balance breadth and depth of our system

Trade-offs

- Breadth : Wide coverage, usable by non-experts (where ML comes in)
- Depth : Well-behaved, verifiable (where FP comes in)

Ideal Pipeline

$$
\begin{aligned}
\text{Utterance} &\xrightarrow{Speech\ Recognizer} \text{String} \\
&\xrightarrow{Canonicalization_{BERT}} \text{String} \\
&\xrightarrow{Parsing\ (GF)} \text{Abstract Syntax Tree (NL)} \\
&\xrightarrow{[\![-]\!]_{Haskell}} \text{Abstract Syntax Tree (LTL)} \\
&\xrightarrow{\vDash} \text{Verifiable Path}
\end{aligned}
\tag{1}
$$

What we have so far...

$$\begin{aligned} \text{String} &\xrightarrow{Parsing\ (GF)} \text{Abstract Syntax Tree (NL)} \\ &\xrightarrow{[\![-]\!]_{Haskell}} \text{Abstract Syntax Tree (LTL)} \\ &\xrightarrow[Agda]{\models} \text{“Standard Semantics”} \end{aligned} \tag{2}$$

Example AST

"go to the store and turn left" ↦

```
CompoundCommand
    * And
      BasePosCommand
        * SimpleCom
            * ModAction
                * Go
                  MkAdvPh
                    * To
                      WhichObject
                        * The
                          Store
          SimpleCom
            * ModAction
                * Turn
                  WherePhrase (Left)
```

Example Cont..

$[\![$"go to the store and turn left . Finish ."$_{AST}]\!]_{Haskell}$

```
F
  (Meet
    (Atom "the_store")
    (X
      (Meet
        (Atom "left")
        (G (Atom "FINISHED")))))
```

Interpretation

Eventually be in a state where you are at the store. Directly after, you are in a state where you are left. From thereon, you are forever finished.

Functional Programming

Types, and type-checking is a fundamental way to seperate the process of programming into two phases

- Static : Specification, Types
- Dynamic : Implentation, Programs

- Correct-by-construction mentality - “verification technique”
- Constraints are baked into user-defined types.
- Specification guides implementation. Functional programmers spend almost all their time thinking about types.
- Focus energy on high-level (abstract) details
- It is convenient to implement programming languages (i.e. write interpreters and compilers)

Grammatical Framework

- Functional programming language
- Implementing natural language parsers and linearizes
- Separation of abstract syntax and concrete syntax.
- Intended for translation, multilingual support
- One designs a grammar, Parallel Multiple CFGs (PMCFG)
- Chomsky Hierarchy : CFG $<$ PMCFG $<$ Context Sensitive Grammar
- Standard Libarary : Resource Grammar Library (RGL)
- Embedding in Haskell : Portable Grammar Format (PGF)

GF Example

BNFC (traditional treatment of fixity)

```
Lam. Exp  ::= "\\" [PTele] "->" Exp ;
App. Exp2 ::= Exp2 Exp3 ;
```

Abstract :

```
fun
  Lam : [Tele] -> Exp -> Exp ;
  App : Exp -> Exp -> Exp ;
```

Concrete :

```
lin
  Lam pt e = mkPrec 0 ("\\" ++ pt ++ "->" ++ usePrec 0 e) ;
  App = infixl 2 "" ;
```

Haskell and Agda

Haskell

- One of the main FP languages - deep ties in Scotland and Sweden
- GADTs and pattern matching make for first-class tree transformations
- Defining and reasoning about logics and programming languages

Agda

- "Dependently typed Haskell"
- Interactive Theorem Prover
- Programs as proofs, propositions as types

Linear Temporal Logic

- Modal logic (temporal modality)
- Allows one to reason about sequential actions
- Verification for robotics systems
- Objective in reinforcement learning
- Other temporal logics (Signal TL, Computation Tree Logic, ...)
- Decidable

Complexity (and expressivity)

Propositional Logic $<$ Temporal Logic $<_{undecidable}$ First Order Logic

Temporal Operators

- $X\phi$: in the next state, phi holds
- $\diamond\phi$: exists a future state such that ϕ holds $(F\phi)$
- $\Box\phi$: ϕ holds for every future state $(G\phi)$

Where does ML come in?

Language Models

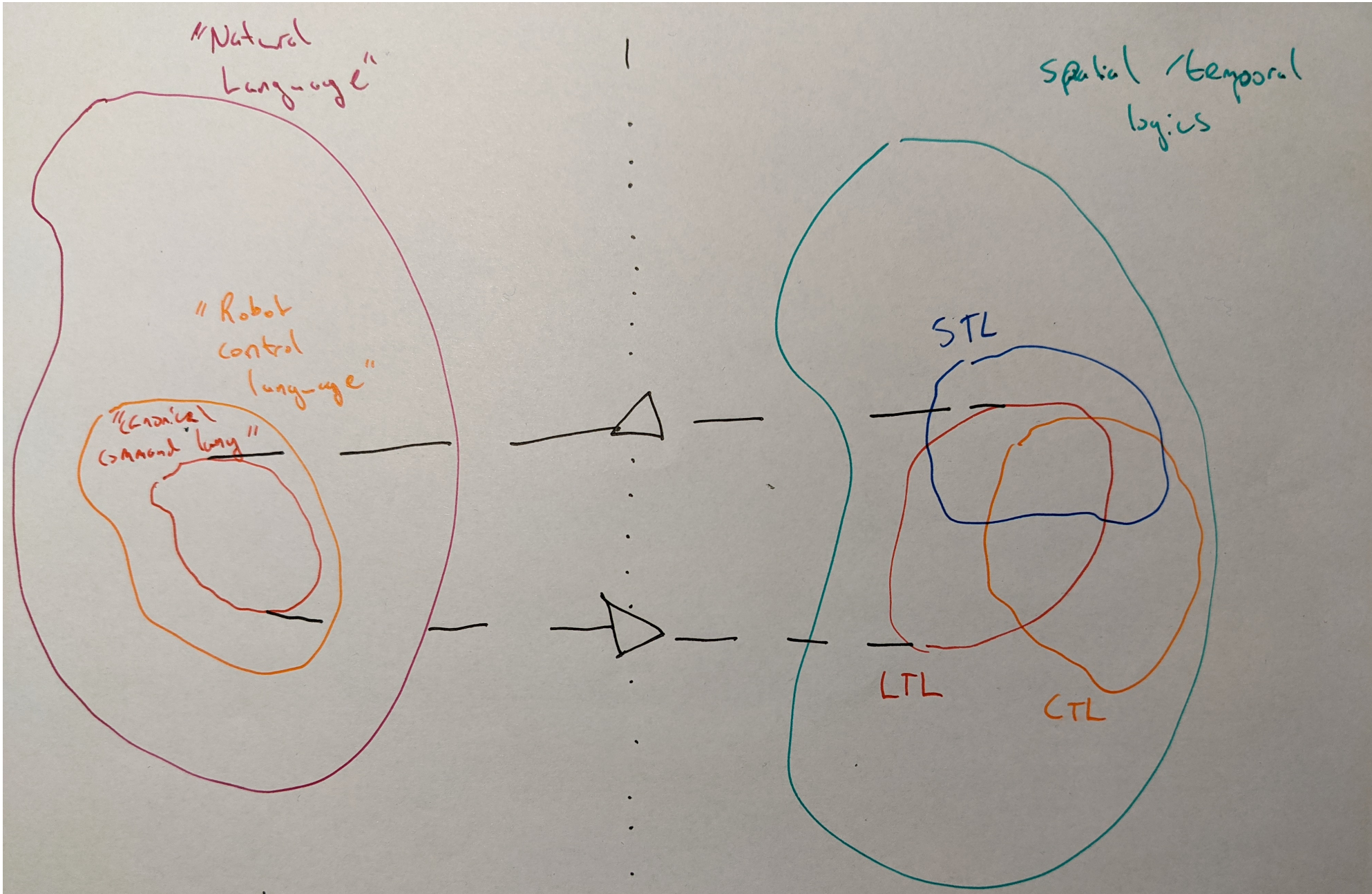
- Foundation models - future of NLP?
- BERT, GPT3, ...
- Pretrain and then fine-tune

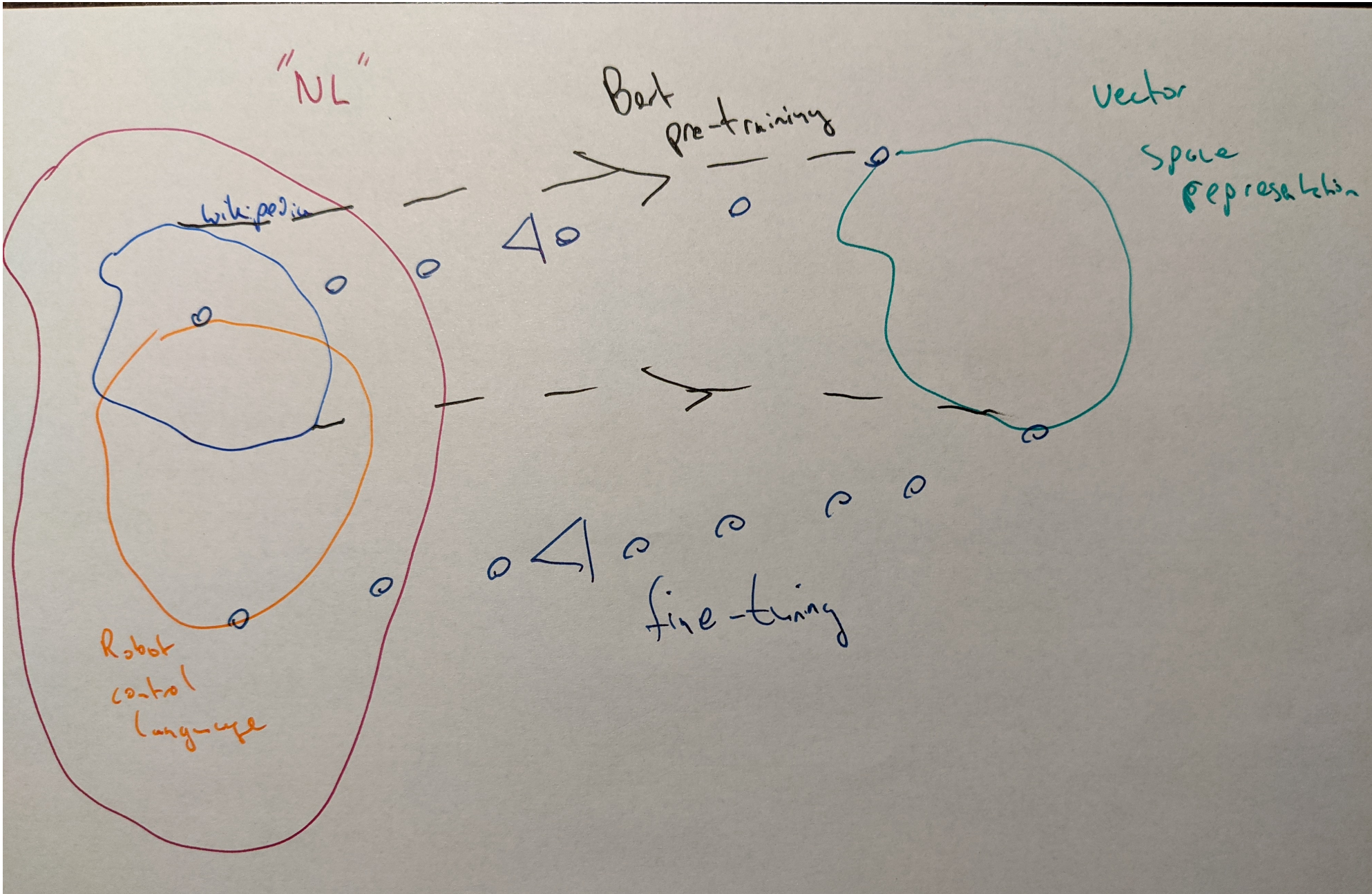
Ingredients

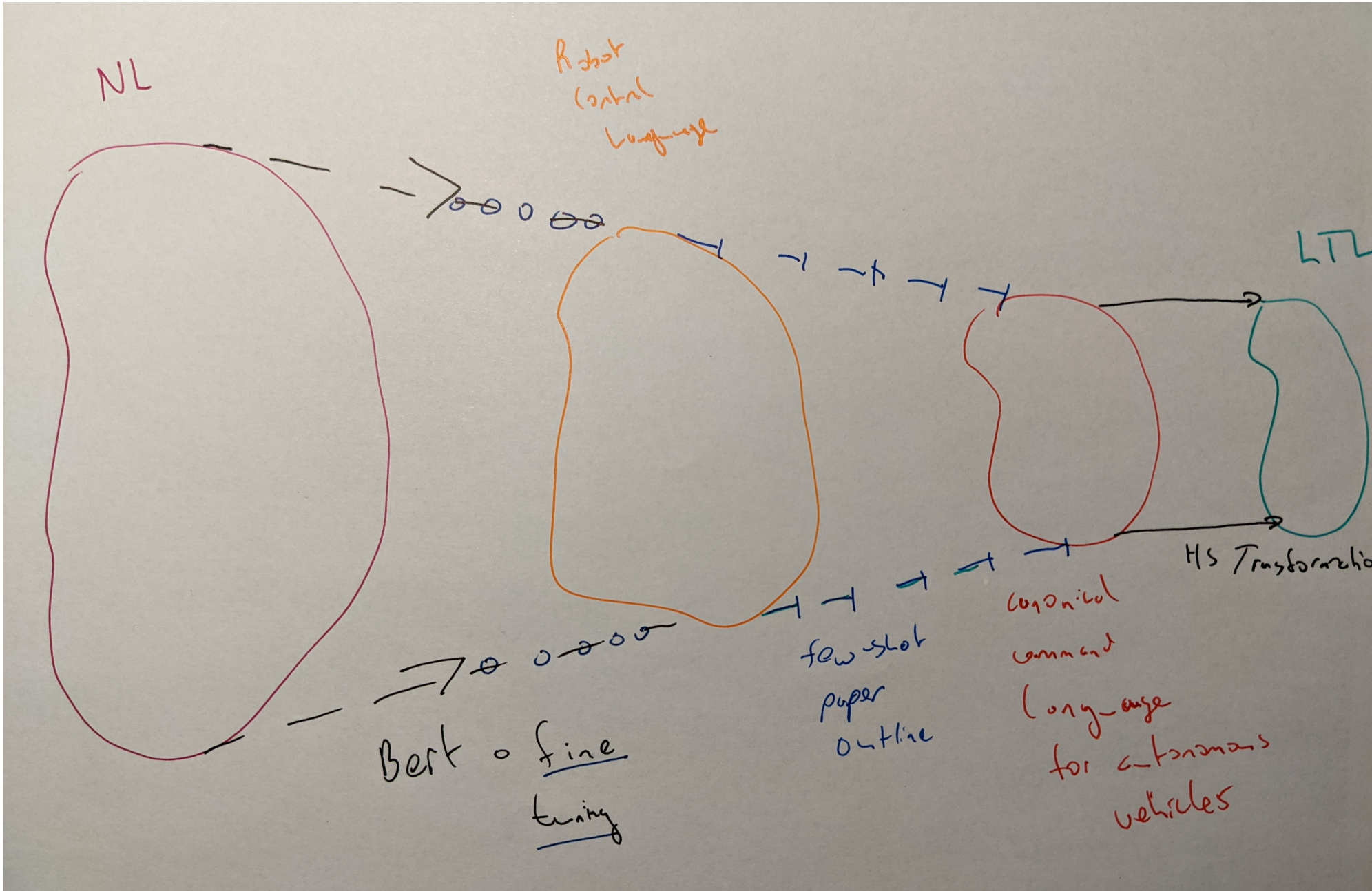
- Semantic Parser : NL Utterance -> Formal Language form
- Touchdown Data Set : 13000 sequences
- Few-shot semantic parsers paper (Microsoft Research) - open source Pytorch code

Idea

Integrate their technique and code with our parser and dataset


"Natural Language"
"Robot control language"
"Logical command language"
Spatial / temporal logics
STL
LTL
CTL


"NL"
Bert
pre-training
Vector
space
representation
Wikipedia
Robot
control
language
fine-tuning


NL
Robot
Control
Language
LTL
HS Transformatio
few shot
paper
outline
canonical
command
language
for autonomous
vehicles
Bert - fine
tuning

Future

- Expand parser itself
- Work on semantics , more expressive logic, etc
- Ground semantics to Touchdown location map
- Better data set?
- Working on a draft document, with references, of everything shown here.