On the syntax and semantics of voice assistants in autonomous vehicles

Warrick Macmillan

6^{th} May 2022

Motivation

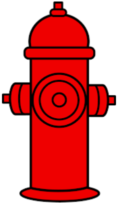
"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."

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."

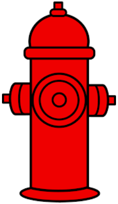
Motivation

"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."



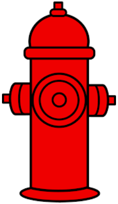
Motivation

"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."



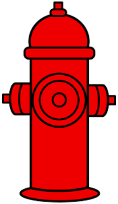
Motivation

"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."



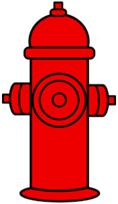
Motivation

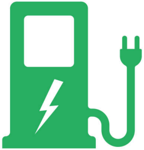
"Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first."



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

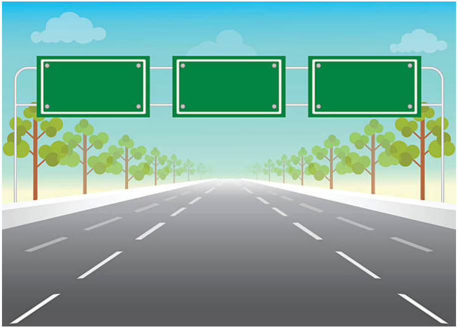
"Go into the other lane"

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

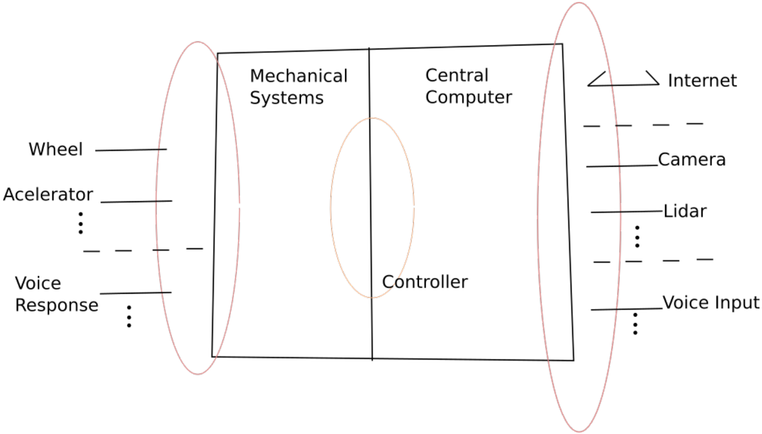


Figure: Self-driving car

Path

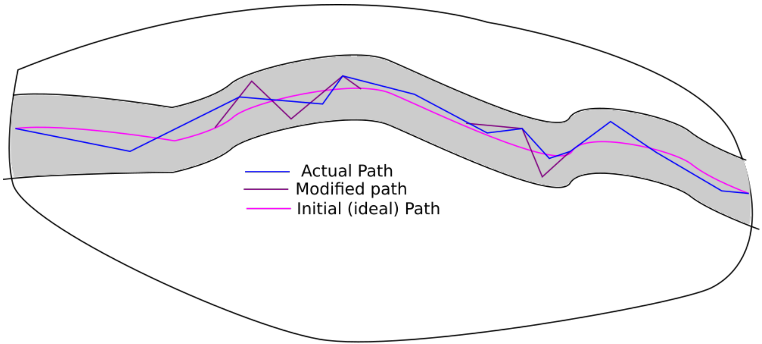


Figure: Initial, modified, and actual paths/routes

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

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

Mathematically Ideal Property

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

where d is some hypothetical metric which should depend on

- Cost
- Safety

Mathematically Ideal Property

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

where d is some hypothetical metric which should depend on

- Cost
- Safety
- Legality

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

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

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

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

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

Mathematically Ideal Property

$\forall u \in$ Utt $\exists r \in$ Routes such that $\forall r' \in$ 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

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

Mathematically Ideal Property

$\forall u \in$ Utt $\exists r \in$ Routes such that $\forall r' \in$ 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

Grammatical Framework

- Multilingual support

Grammatical Framework

- Multilingual support
- Standard Libarary : Resource Grammar Library (RGL)

Grammatical Framework

- Multilingual support
- Standard Libarary : Resource Grammar Library (RGL)
- Embedding in Haskell (as GADTs) : Portable Grammar Format (PGF)

Grammatical Framework

- Multilingual support
- Standard Libarary : Resource Grammar Library (RGL)
- Embedding in Haskell (as GADTs) : Portable Grammar Format (PGF)
- Simple types, functional programming

Grammatical Framework

- Multilingual support
- Standard Libarary : Resource Grammar Library (RGL)
- Embedding in Haskell (as GADTs) : Portable Grammar Format (PGF)
- Simple types, functional programming
- “Parallel multiple CFGs”

Grammatical Framework

- Multilingual support
- Standard Libarary : Resource Grammar Library (RGL)
- Embedding in Haskell (as GADTs) : Portable Grammar Format (PGF)
- Simple types, functional programming
- "Parallel multiple CFGs"
- Chomsky Hierarchy : CFG < PMCFG < Context Sensitive Grammar

Grammatical Framework

- Multilingual support
- Standard Libarary : Resource Grammar Library (RGL)
- Embedding in Haskell (as GADTs) : Portable Grammar Format (PGF)
- Simple types, functional programming
- "Parallel multiple CFGs"
- Chomsky Hierarchy : CFG < PMCFG < Context Sensitive Grammar
- Separation of abstract (semantic and syntactic) and concrete (syntactic and morphological) considerations

Abstract Syntax

- Semantic considerations

Abstract Syntax

- Semantic considerations
- Coarse meaning space

Abstract Syntax

- Semantic considerations
- Coarse meaning space
- Tree formation

Abstract Syntax

- Semantic considerations
- Coarse meaning space
- Tree formation
- Categories cat (basic types)

Abstract Syntax

- Semantic considerations
- Coarse meaning space
- Tree formation
- Categories cat (basic types)
- Named function fun of arbitrary arities over them

Concrete Syntax

- Lexical Details

Concrete Syntax

- Lexical Details
- Function body, how strings are formed

Concrete Syntax

- Lexical Details
- Function body, how strings are formed
-

TOUCHDOWN Dataset

```
([(["Go"],5527),(["be"],5300),(["turn"],4154),(["Turn"],4154)]
([(["left"],228),(["came"],59),(["made"],47),(["started"],44)]
([(["going"],1684),(["facing"],939),(["moving"],849),(["passin
([(["left"],1733),(["parked"],616),(["painted"],307),(["fencec
([(["are"],3554),(["'re"],1185),(["reach"],1100),(["get"],770)
([(["is"],4919),(["has"],795),(["'s"],471),(["ends"],93)],["VE
```

n-grams : the 9-gram “so you are moving with the flow of traffic” occurs 311

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
```

GF Functions

```
fun
  -- Explicit Temporality
  DoTil      : Action -> Time -> PosCommand ; go in one minute

  -- Modified action
  ModAction : Action -> AdvPh -> Action ; -- go to the store

  -- Adverbial Phrases
  MkAdvPh      : Way   -> Object -> AdvPh ; -- to the store

  -- Noun Phrases
  WhichObject : Determ -> UndetObj -> Object ; -- the red dog

  -- Modified Noun
  ModObj : Descript -> UndetObj -> UndetObj ; -- black dog
```

Base Ingredients

These represent the tree leaves to be grounded!

```
fun
  Quickly : How      ;
  Left    : Where    ;
  To      : Way      ;
  After   : Way      ;
  Store   : UndetObj ;
  Traffic : UndetObj ;
  London  : Place    ;
  Drive   : Action   ;
  Turn    : Action   ;
  Big     : Descript ;
  A       : Determ   ;
```

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.

```
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.

```
            * 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
nsCommands
* 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.

Alternative Interpretation

go (to the store, turn left and stop (at the woman)) with the dog.

Need a temporal until operator, U, to construct a semantically justifiable interpretation

```
ModAction
  * Stop
    MkAdvPh
      * At
        WhichObject
          * The
            Woman
MkAdvPh
  * With
    WhichObject
      * The
        Dog
```

Haskell LTL

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

```
F (Meet
    (Atom "the_store")
    (F (Meet
        (Atom "turn_left")
        (F (Meet
             (Atom "the_woman_with_the_dog")
             (F (Meet
                  (Atom "the_bridge")
                  (G (Atom "FINISHED"))))))))
```

Lists under the hood

```
normalizeList ::  GListCommands -> GListCommands
where
normalizeNestedLists :: GListCommands -> GListPosCommand
  where
    normalizeListPosCommand :: GListPosCommand -> GListPosCo
      where
        unSentence :: [GCommands] -> [GPosCommand]
        flattenSublist :: GPosCommand -> [GPosCommand]
        where
          getListPosCommands :: GListPosCommand -> [GPosComm
```

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

Ideal Pipeline

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

What we have so far...

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

Trade-offs

- Depth versus breadth

Trade-offs

- Depth versus breadth
  - Breadth : Wide coverage, usable by non-experts (where ML comes in)

Trade-offs

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

Trade-offs

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

Trade-offs

- Depth versus breadth
  - Breadth : Wide coverage, usable by non-experts (where ML comes in)
  - Depth : Well-behaved, verifiable (where FP comes in)
- Specificity and generality
- Formality and formalizability

Trade-offs

- Depth versus breadth
  - Breadth : Wide coverage, usable by non-experts (where ML comes in)
  - Depth : Well-behaved, verifiable (where FP comes in)
- Specificity and generality
- Formality and formalizability
- Verifiability and validity

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

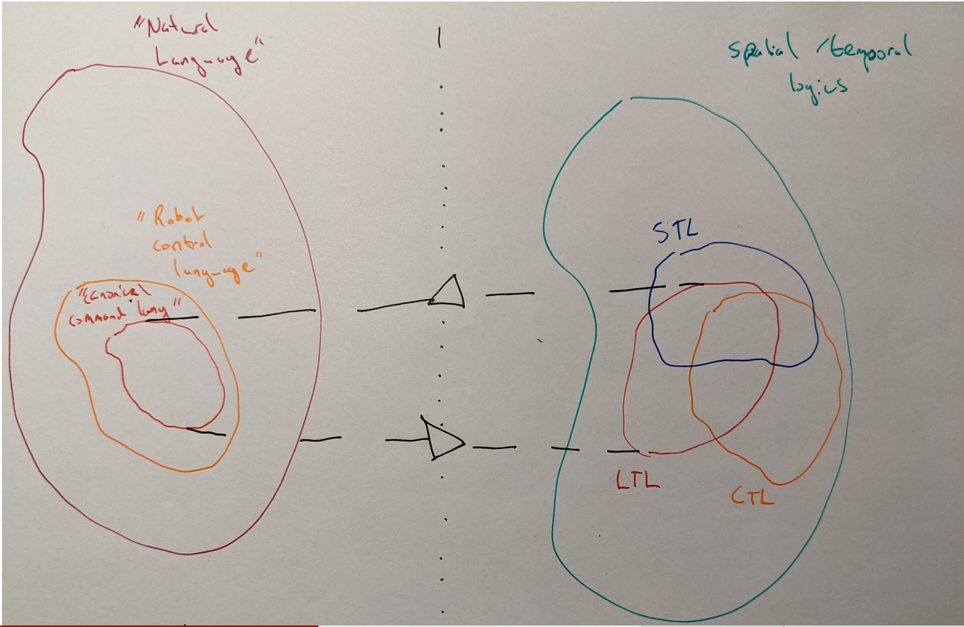
- Foundation models - future of NLP?
- BERT, GPT3, ...
- Pretrain and then fine-tune

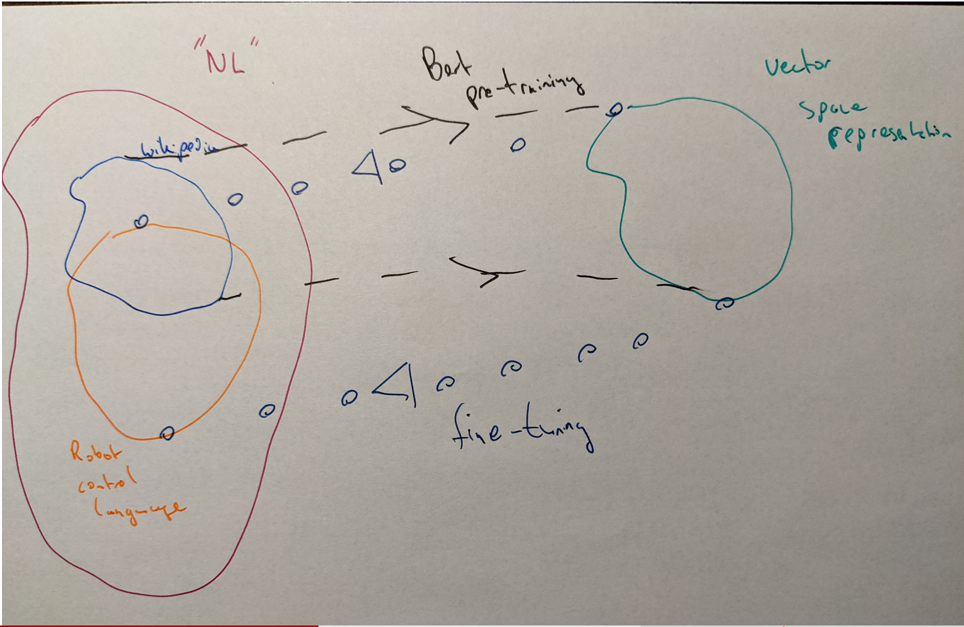
Ingredients

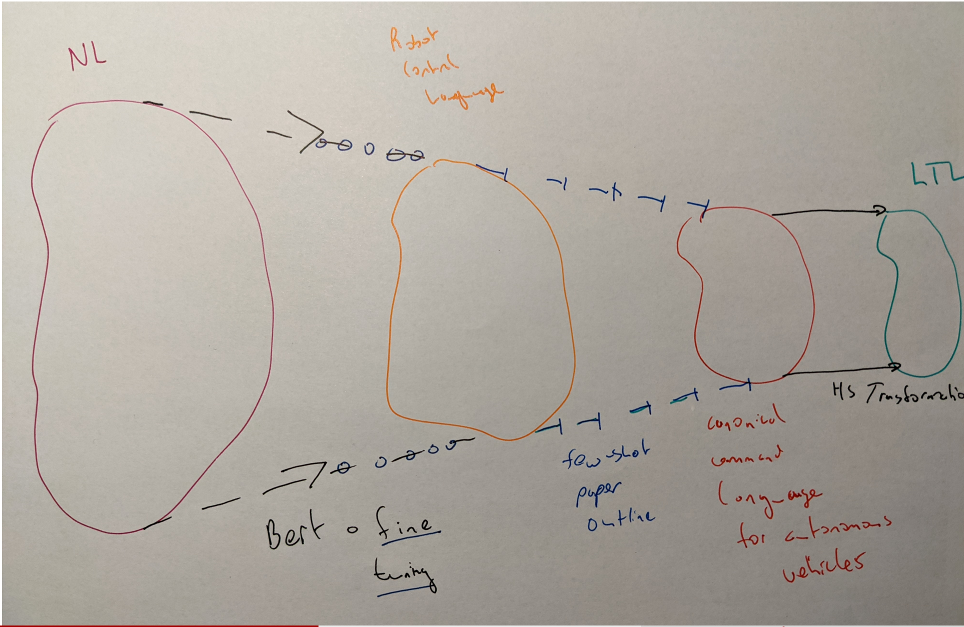
- Semantic Parser : NL Utterance -> Formal Language form
- Touchdown Data Set : 13000 sequences
- Few-shot semantic parsers paper (Microsoft Research) - open source Pytorch code

Idea

Integrate their technique and code with our parser and dataset


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


"NL"
wikipedia
Robot
control
language
Bert
pre-training
fine-tuning
Vector
Space
representation


NL
Robot
Control
Language
LTL
few shot
paper
outline
Canonical
command
language
for autonomous
vehicles
HS Transformatio
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.