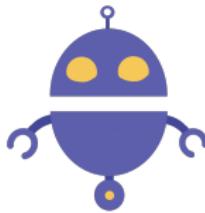


On the syntax and semantics of voice assistants in autonomous vehicles

Warrick Macmillan

6th May 2022



```
data  $\phi$  : Set where
  atom  : Atom  $\rightarrow \phi$ 
   $\perp T$   :  $\phi$ 
   $\neg_-$   :  $\phi \rightarrow \phi$ 
   $\_ \vee \_ \wedge \_ \Rightarrow \_$  :  $\phi \rightarrow \phi \rightarrow \phi$ 
  X F G :  $\phi \rightarrow \phi$ 
   $\_ \cup \_ \cap \_ R \_$  :  $\phi \rightarrow \phi \rightarrow \phi$ 
```

Motivation

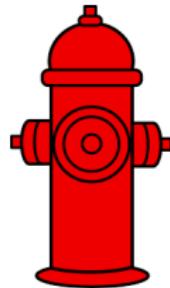
“Go to the grocery store after the next exit, and then go to fuel station, but stop by the fire hydrant so I can take a picture of that crazy sign, first.”

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.”

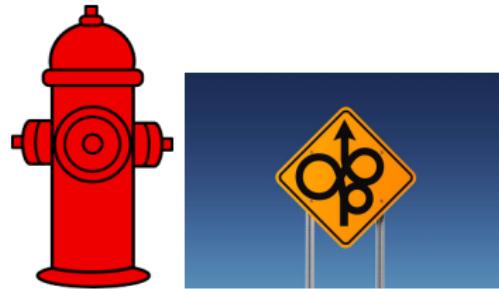
Motivation

"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

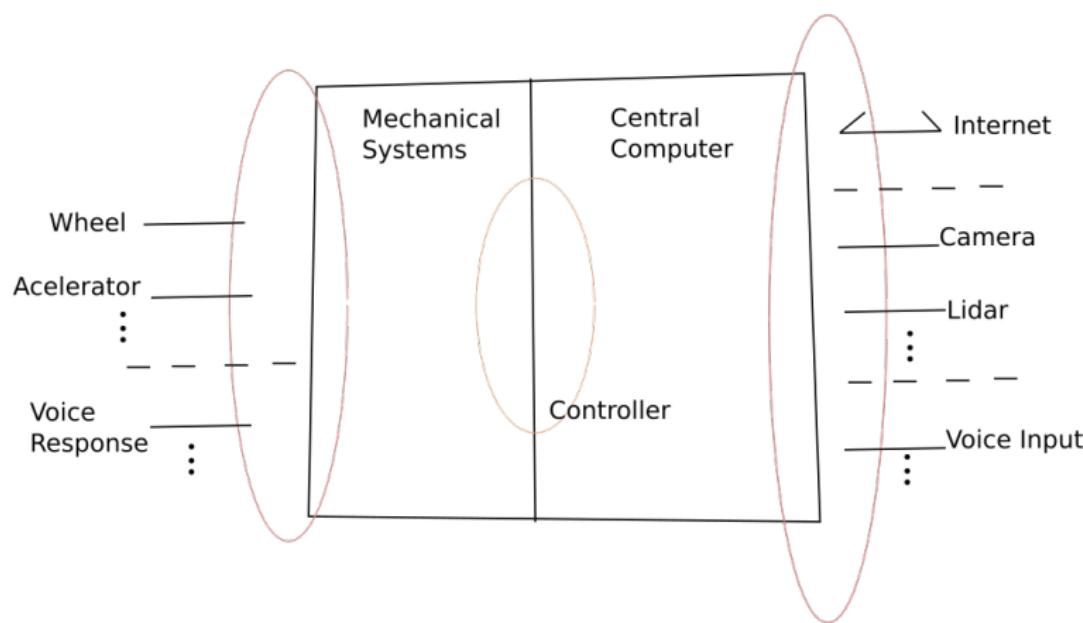


Figure: Self-driving car

Path

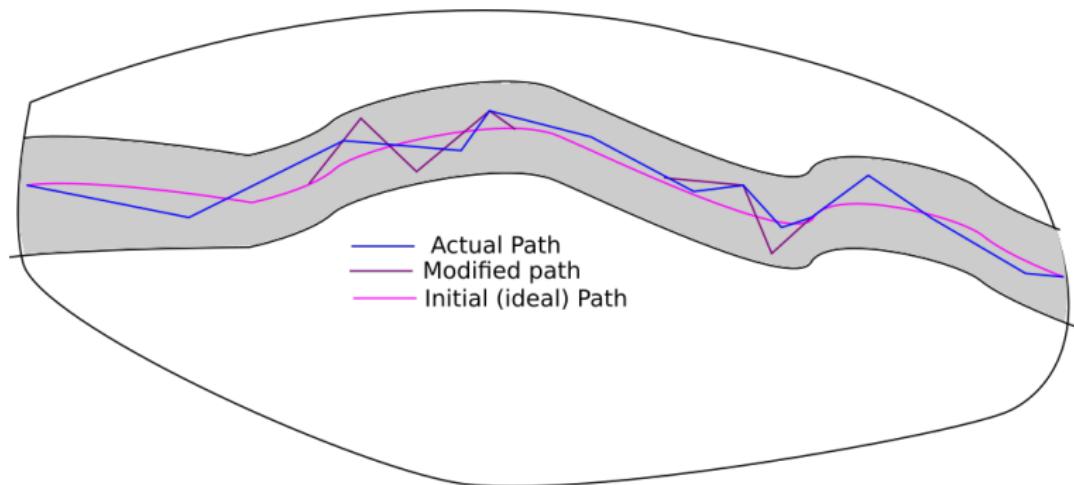


Figure: Initial, modified, and actual paths/routes

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

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

where d is some hypothetical metric which should depend on

- Cost
- Safety

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

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

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

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

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

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

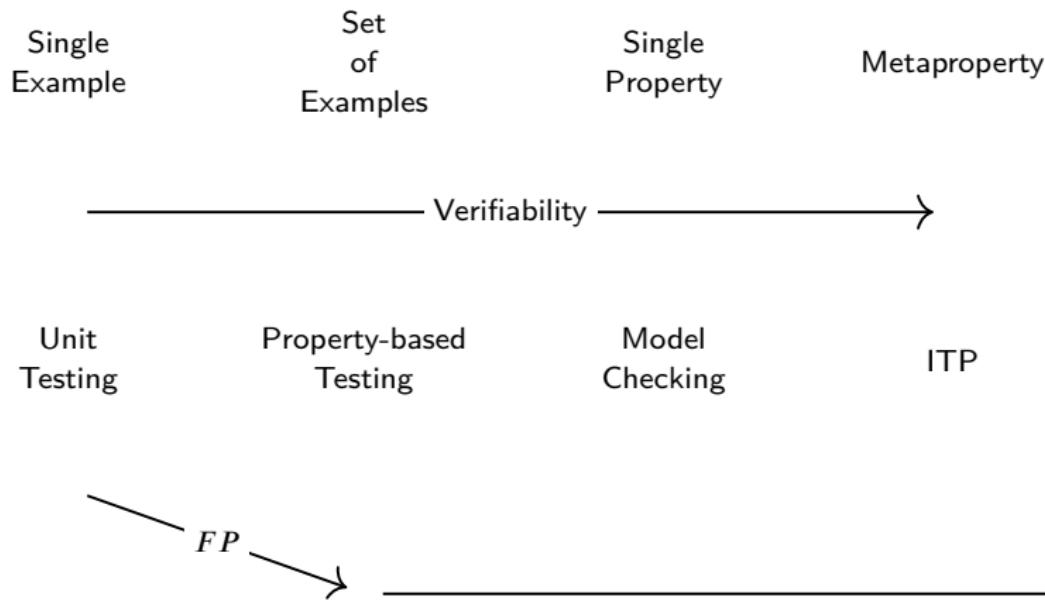
where d is some hypothetical metric which should depend on

- 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
 - physical objects in the case of paths in Euclidean space

Verification Spectrum



Main Idea

- Functional Programming is all about composition of higher order functions

Main Idea

- Functional Programming is all about composition of higher order functions
- Break down system into composable *big-components*, each of which

Main Idea

- Functional Programming is all about composition of higher order functions
- Break down system into composable *big-components*, each of which
 - admits composition of *small-components*

Main Idea

- Functional Programming is all about composition of higher order functions
- Break down system into composable *big-components*, each of which
 - admits composition of *small-components*
 - operates under different verification conditions

Main Idea

- Functional Programming is all about composition of higher order functions
- Break down system into composable *big-components*, each of which
 - admits composition of *small-components*
 - operates under different verification conditions
- Ensure maps between big-components are functorial with respect to small-component composition

Main Idea

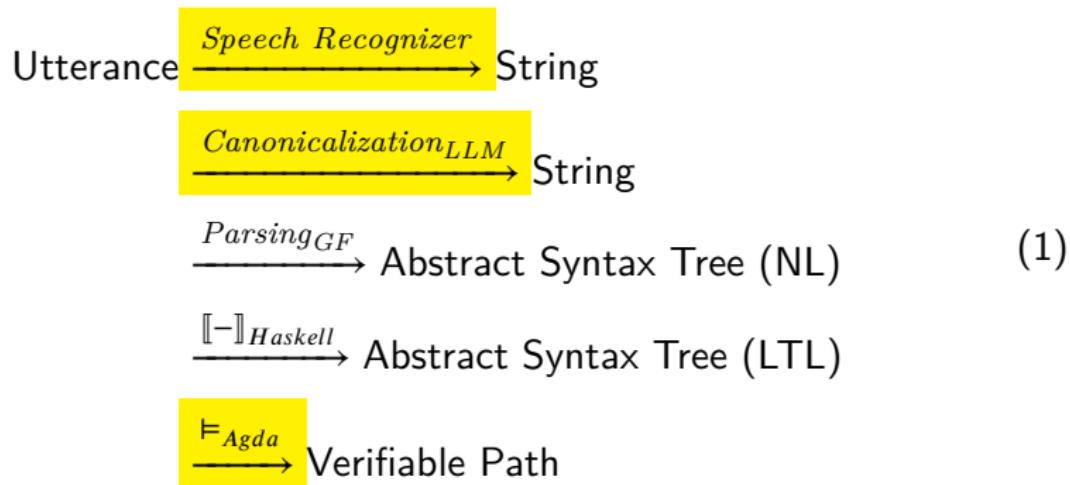
- Functional Programming is all about composition of higher order functions
- Break down system into composable *big-components*, each of which
 - admits composition of *small-components*
 - operates under different verification conditions
- Ensure maps between big-components are functorial with respect to small-component composition
- Verification of system is can similarly be broken into sub-verifications

More Specifically

This project contains bits and pieces of :

- ① Functional Programming Languages : GF (Grammatical Framework), Haskell, Agda
- ② Natural Language Processing : Semantic Parsing, Controlled Natural Languages
- ③ Tools : Grammars, Decidable logics for motion planning and verification

Ideal Pipeline



Grammatical Framework

- Multilingual support

Grammatical Framework

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

Grammatical Framework

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

Grammatical Framework

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

Grammatical Framework

- Multilingual support
- Standard Library : 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 Library : Resource Grammar Library (RGL)
- Embedding in Haskell (as GADTs) : Portable Grammar Format (PGF)
- Simple types, functional programming
- “Parallel multiple CFGs”
- Chomsky Hierarchy : $\text{CFG} < \text{PMCFG} < \text{Context Sensitive Grammar}$

Grammatical Framework

- Multilingual support
- Standard Library : Resource Grammar Library (RGL)
- Embedding in Haskell (as GADTs) : Portable Grammar Format (PGF)
- Simple types, functional programming
- “Parallel multiple CFGs”
- Chomsky Hierarchy : $\text{CFG} < \text{PMCFG} < \text{Context Sensitive Grammar}$
- 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)),["VB"])
((["left"],228),(["came"],59),(["made"],47),(["started"],44)),["VBD"])
((["going"],1684),(["facing"],939),(["moving"],849),(["passing"],617)),["V"]
((["left"],1733),(["parked"],616),(["painted"],307),(["fenced"],263)),["VB"]
((["are"],3554),(["'re"],1185),(["reach"],1100),(["get"],770)),["VBP"])
((["is"],4919),(["has"],795),(["'s"],471),(["ends"],93)),["VBZ"])
```

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
                MkAdvPh
                  * 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.

```
ModAction
  * Go
    MkAdvPh
      * To
        WhichObject
          * The
            Store
BasePosCommand
  * SimpleCom
    * ModAction
      * Turn
        WherePhrase
          Left
SimpleCom
  * ModAction
    Stop
    MkAdvPh
      * At
        WhichObject
          * The
            PhraseModObj
              * Woman
                MkAdvPh
                  * With
                    WhichObject
                      The
                        Dog
OneCommands
  * OneCommand
    * SimpleCom
      * ModAction
        * Go
        MkAdvPh
          * To
            WhichObject
              The
                Bridge
BaseCommands
  * OneCommand
    * Finish
```

The image shows a hierarchical tree structure of a speech act grammar. The root node is 'ModAction'. It branches into 'Go' (via 'MkAdvPh'), 'Turn' (via 'WherePhrase'), 'Stop' (via 'SimpleCom'), and 'Finish' (via 'BaseCommands'). 'Go' and 'Turn' both have 'WhichObject' children, which are circled in pink. 'Turn' also has a 'Left' child, which is circled in pink. 'Stop' has a 'PhraseModObj' child, which has 'Woman' and 'With' children, and a 'Dog' child, all of which are circled in pink. 'Finish' has a 'OneCommand' child, which has a 'SimpleCom' child, which has a 'ModAction' child, which has a 'Go' child, and a 'To' child, which has a 'WhichObject' child, which is circled in pink.

Alternative Interpretation

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

Alternative Interpretation

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

ModAction

- * Stop

MkAdvPh

- * At

WhichObject

- * The

Woman

MkAdvPh

- * With

WhichObject

- * The

Dog

Alternative Interpretation

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

ModAction

- * Stop

MkAdvPh

- * At

WhichObject

- * The

Woman

MkAdvPh

- * With

WhichObject

- * The

Dog

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

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")))))))))
```

Haskell Semantics

Sequence of future states reveal a simple list flattening procedure which amounts to an exceedingly simple denotational semantics

Haskell Semantics

Sequence of future states reveal a simple list flattening procedure which amounts to an exceedingly simple denotational semantics

```
semantics :: GListCommands -> Phi
semantics x =
  let (GListCommands ((GOneCommand y) : _)) = normalizeList x
  in case y of
    q@(GSimpleCom a) -> astToAtom q
    (GCompoundCommand GAnd (GListPosCommand xs)) -> listCommand2LTL xs
```

Haskell Semantics

Sequence of future states reveal a simple list flattening procedure which amounts to an exceedingly simple denotational semantics

```
semantics :: GListCommands -> Phi
semantics x =
  let (GListCommands ((GOneCommand y) : _)) = normalizeList x
  in case y of
    q@(GSimpleCom a) -> astToAtom q
    (GCompoundCommand GAnd (GListPosCommand xs)) -> listCommand2LTL xs

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

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 ($G\phi$)

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

- 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

Where does ML come in?

Language Models

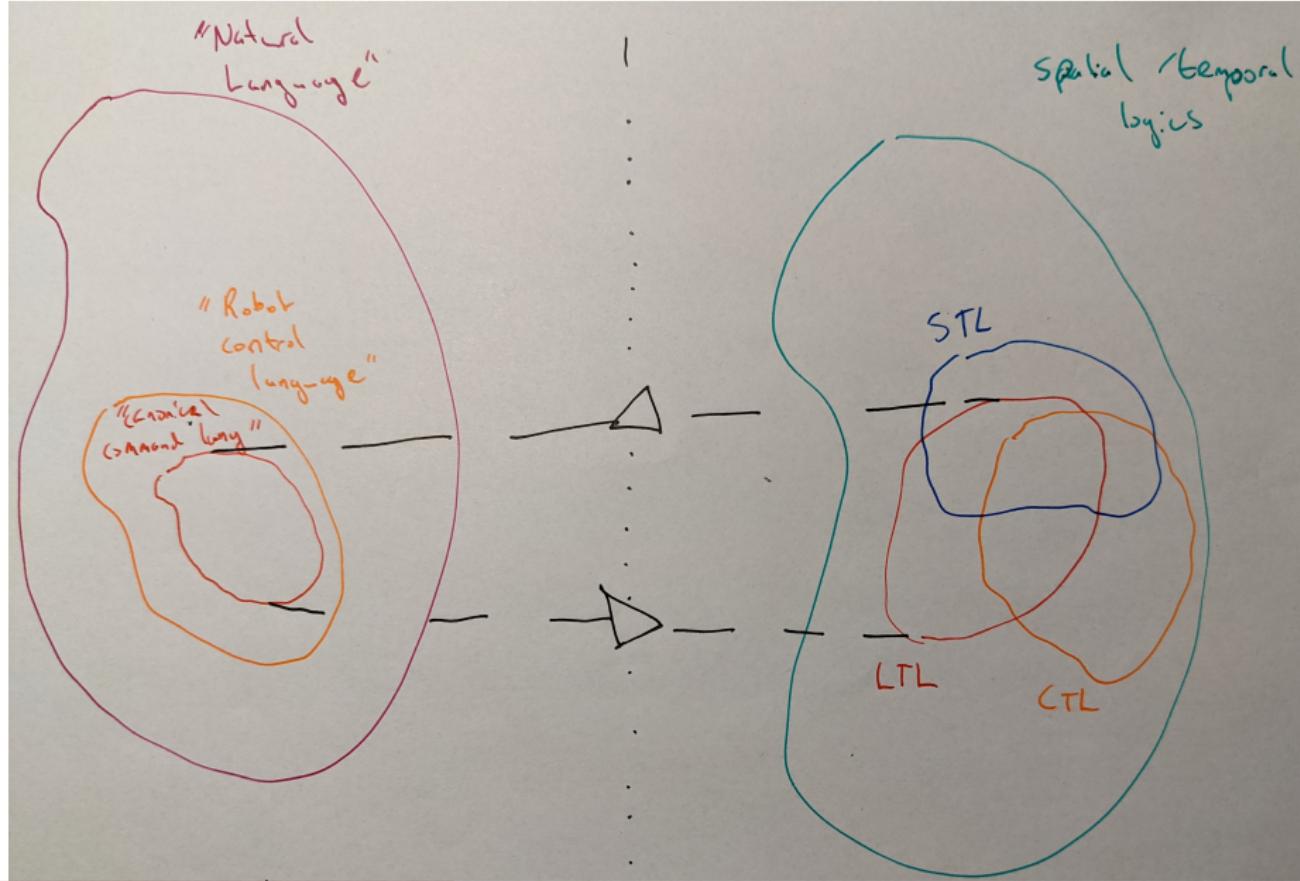
- Foundation models - future of NLP?
- BERT, GPT3, ...
- Pretrain and then fine-tune

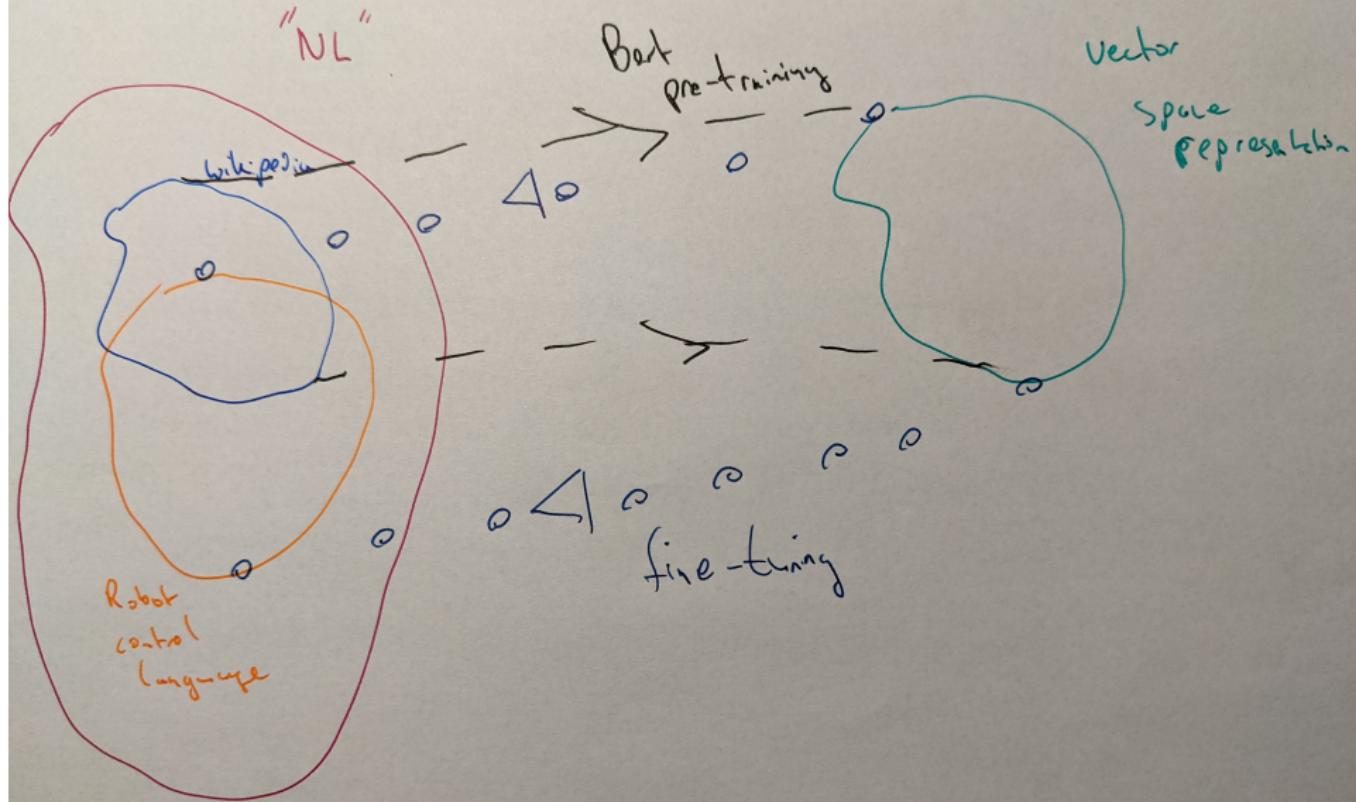
Ingredients

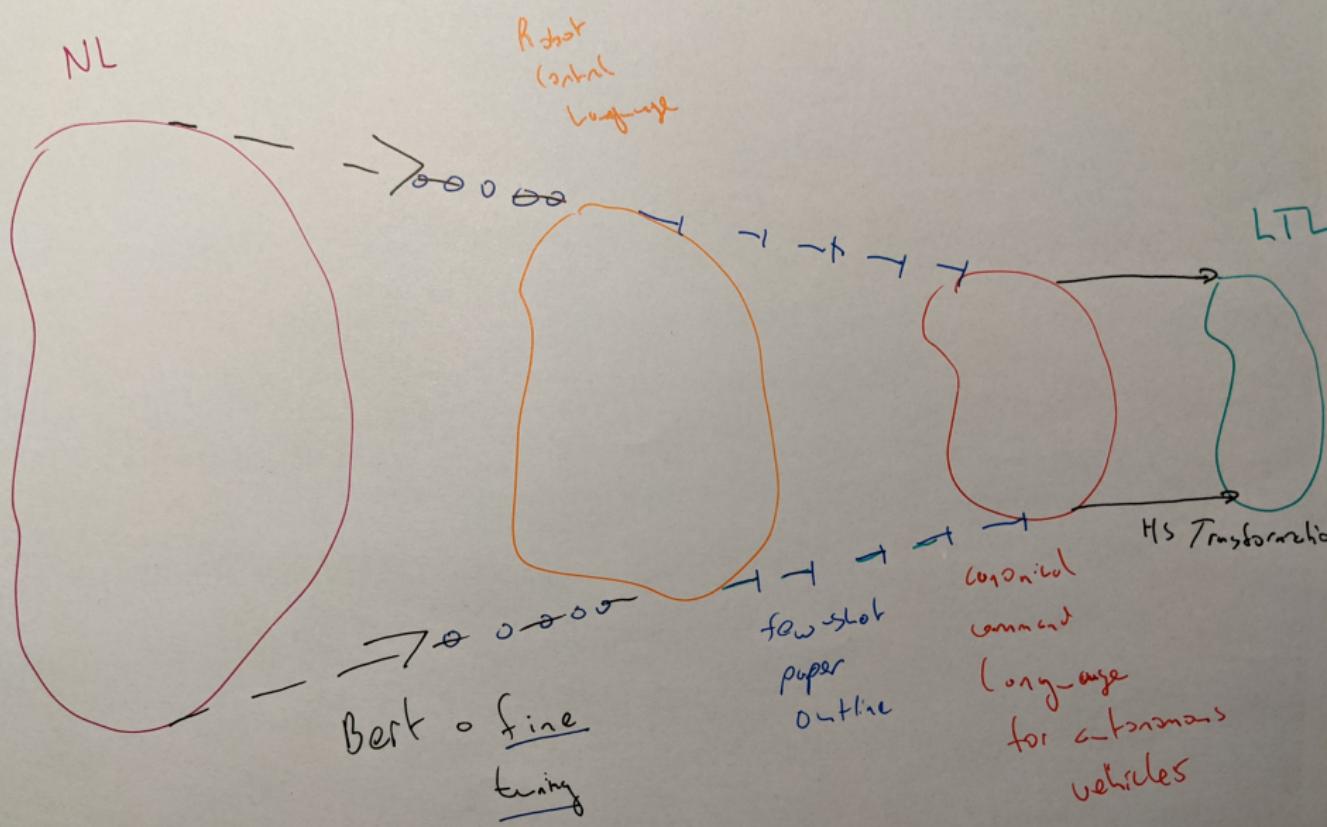
- Semantic Parser : NL Utterance -> Formal Language form
- Touchdown Data Set : 13000 sequences
- Few-shot semantic parsers paper (Microsoft Research) - open source Pytorch code

Idea

Integrate their technique and code with our parser and dataset







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.

Acknolwedgements