NLP – Intent Detection (using prolog)

Sandu Cristian

student at Department of Computer Science – UTCluj

Cluj-Napoca, România  
[cristian.sandu@student.utcluj.ro](mailto:cristian.sandu@student.utcluj.ro)

Horea Horea Ștefan  
student at Department of Computer Science – UTClujCluj-Napoca, România  
[horea.horea.stefan@gmail.com](mailto:horea.horea.stefan@gmail.com)

**Beginning in 1950s with Georgetown experiment in which a computer could perform automatic translation, the Natural Language Processing field was in continuing evolution. With the evolution of the complexity, many subfields were created. One of such subfields is Intent Detection.**

# Introduction

An automaton which performs Intent Detection is defined as being able to extract and process spoken language in order to detect the parts which represent an interest for the system. Humanly understandable input should be defined and transformed so that the process by practical for the computer. The conversion is to be done in such way so that the meaning of the sentence is not altered, but only interpreted.

This material is proposing a high-level, general solution for this procedure. The system which is described in the following paragraphs represents only the detection component from a complex pipeline\*.

# The problem of intent detection

Detecting intent digitally has the purpose to influence the way in which a 3rd party tool operates (whether it is a computing device, an application,etc) - therefore an essential limit of any solution to this problem is depending on the nature of the usage of intent. In our case, we understood early that the set of potential intents in not only finite in size, but relatively small - generic intent detection is impractical if there is no automation use for it - which implies no issues in scaling size of intent set.

However, three issues arise at the level of semantics:

First, the cases which are system-incoherent - intents whose behaviour is undefined, which create uncertainty.

Second, the cases which are set-undefined - intents whose behaviour is defined only for cases which do not apply for the the given input structure.

Third, the (possibly high) chance of set-malformed intents - those are well defined, but contain distortions in structure and therefore require high levels of processing in order to attain a decent accuracy level regarding intent certainty.

# our solution in context

We propose a solution for the system-coherent, set-defined but potentially malformed intent problem which is linear in both time and space, which solves the following erroneous traits : (excluded the base cases and inset-coherent ones:

i) Missing words - either one or more as long as the KB\* inset is coherent

ii) Ill-formed words - which are part of the data relevant for the intent outcome, as long as they are structurally compatible and use the right file encoding

iii) Ill-formed words which are part of the grammar logic and structure, in limited ways in the basic module, extensible by KB\* manipulation though flags.

# High level understanding

From a software engineering perspective the tool presented contains multiple layers, therefore some explanation is necessary in order to assure a good understanding of the way in which it operates and interacts with the pipeline.

## Abbreviations and Acronyms

i) The “tool” represents the currently presented program, while the “pipeline” represents the graph of agents which together constitute the NLP project.

ii) The “Admin” is the person that makes the a priori choices for the program logic, which are not defined explicitly by the program and which require a 3rd party in order to function in a practical manner.

iii) When we refer to the knowledge base(KB), we do not refer to a set of data contained in the prolog file - instead, it represents the data construct which we use to contain the abstraction of the potential variations of statements that could be received.

iv) When we address the idea of the “kernel”, we refer to the prolog processing function(s) that is(are) contained inside the python wrapper.

v) “Flags” are bits of text which are added arbitrarily in the knowledge base after the preprocessing phase in order to specify the ways in which the Admin wants the kernel to behave with respect to the knowledge base.

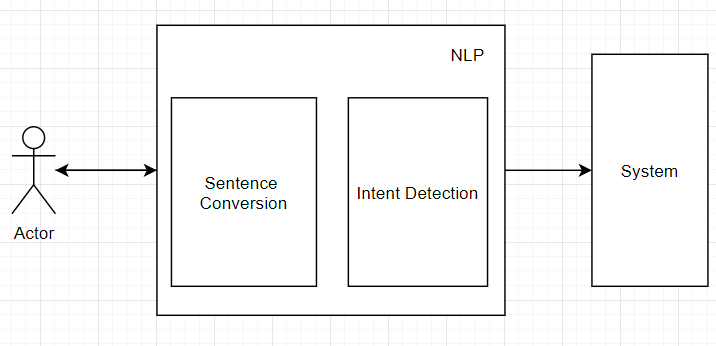
i) The Intent Scenario File (called IFS) is the file which contains the set of all potential intents, which all in turn contain all subsets of wordings which can be relevant for each of the aforementioned intents.

## Component hierarchy summary - bottom-up perspective

* The IFS contains the raw input (i.e. the variability in phrasing of potential intents);
* The initial KB, created through offline preprocessing on top of the IFS, contains the raw intent tree;
  + The Admin will specify rules for all flags that can be used - note that given the structure this step can also be done manually if the dataset is small enough.
* Once the KB has been fine-tuned by the Admin, the kernel can coherently process information offline
* The wrapper is the middleman between the online pipeline input and the offline kernel processing

# Architecture & Pipeline

## Pipeline

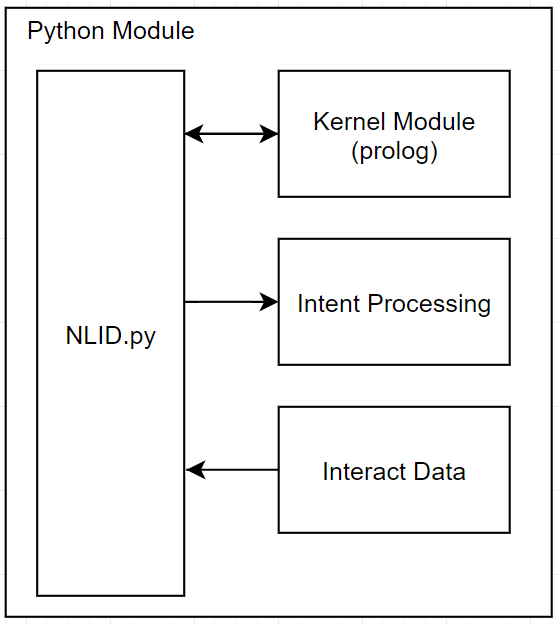
The following schema represents the simplified version of the pipeline which compose the system. An actor, in our case a human, will start the sentence, giving an input to the system.

The *Sentence Conversion* module will lexically analyze the sentence in order to extract the region of interests, this data will be further into the pipeline.

Next into the pipeline is *Indent Detection* module which should be able, from the given input data, to convert it into a format understandable by computer. This is the last part in our pipeline, meaning that the output from this module should go to the system.

*B. Architecture*

In the followings it is a simple representation of the architecture used in Intent Detection module.

**

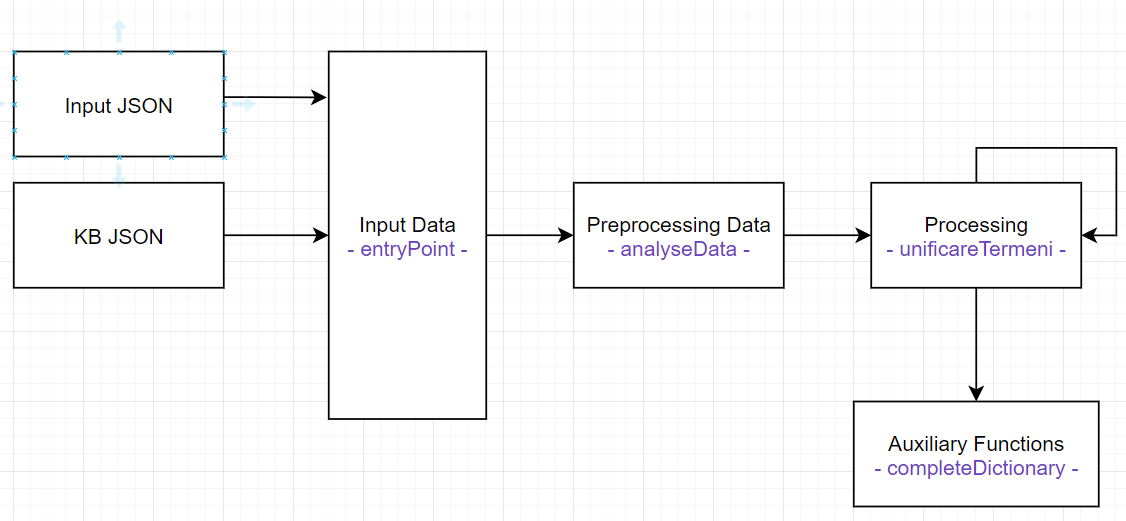
The brain, or kernel, for this module, where the logic is located is named: Kernel Module. This is a written in prolog. Prolog language was chosen because of its ability to force a backtracking mechanism and obtain all of the solutions for a given problem.

NLID is a simple class written in python which acts as a communication medium (as a BUS) for all modules in the system.

Intent Processing & Interact Data are used as auxiliary modules to perform task as: connecting with google account & updating status for different actions.

# Kernel Module

This module contains all the logics which gives to our system the ability to successful transform data. The following schema represents a simplified flow in the prolog code,



All input data is sent in entryPoint which will call a preprocessing function. “analyseData” make the basic transformation in order to convert the data to a format which is understood my system.

The interesting part in the “unificareTermeni”. The five predicates which compose this function were designed to handle all of the possible states of a sentence. (defined above). This is the main body of the execution tree created by prolog.

Auxiliary functions, named completeDictionary, are used for basic, non important to mention tasks. (the functionality of this function can be seen in the comments in code).

The code is well documented, so there is no reason to explain some of the decisions here.

# Knowledge base

The power of the this implementation is given by the presence of the knowledgebase. It contains the information which is used by prolog module to unify the terms.

## Format of KB

The structure of KB is represented as a tree, for this reason we used the JSON file format to store it.

The following segment of code is a simplified version of the actual KB, (flags were eliminated for simplicity)



In this example the “adaugaEvent” & “interabaEvent” are the direct children of the root of the tree.

The @data is a subtree which is present in both the upper level.

By writing the common branches only one and adding the the next level the reference of the branche we dealt with the problem of inclusion levels.

## Flags

There are three types of flags that were defined:

final flag marks that a sentence has ended. Also this flags is useful when dealing with malformated sentences.

default flag was used to indicate that a value is missing and can be substituted with another value

canAbsent indicates that the current word can be absent. It will force a bruteforce to a lower level in tree

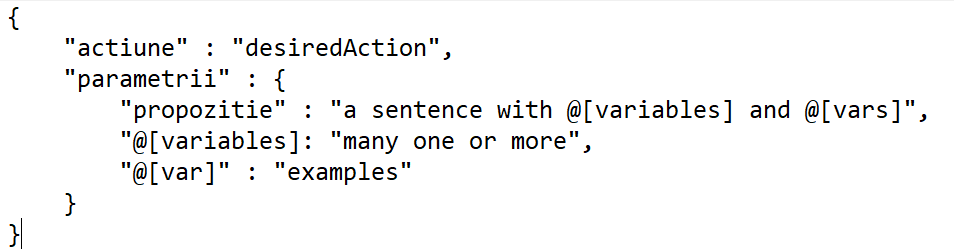
Please note that flag definition is an essential part of the tool and should not be underestimated - intent logic can vary and special cases can occur, therefore using this feature is essential in order to maximize the limits of the kernel processing.

## Algorithm used for unification

The unification is made by “unificareTermeni” predicate. Mostly the algorithm is based on DFS. It will search in the input “parametrii” can going down in KB according to rules & flags defined above.

# Input Format

The input for the Intent Detection module from NLP is given by Sentence Conversion. Here is presented a simplified version of this input (which is a JSON format):



The keys from the json are having the following meaning:

First one, “actiune” is the principal action which was detected in the sentence. This word is the master key for interpreting the input.

Second key is “parametrii” which describes, at a high level, the sentence form as a JSON. First argument of this json is “propozitie” which describes the sentence which was given as input. The words which begin with @ are called variables. They have a value which should be use by ID. Their values can be present, in this case they appear from the second row in json, or can be missing.

# Cases for a sentence

Will researching this subject, there were identified, for a general solution, four principal cases as follow:

Valid Forms: The sentences which are valid from the sintactic point of view.

* Complete Form: a well defined sentence. All data is present in the input & no need for default.
* Incomplete Form: a sentence where the topologic & semantics are correct, but there are missing values for some variables:
  + One Default: one value missing, using a default from KB - ask for confirmation
  + Two or More defaults: repeat the sentence

Invalid Forms: The sentence was malformated. It could not be understand from the current data in knowledge base.

* Premature Ending: there is not a final flag detected in sentence - repeat it
* Malformed Word:
* if a final was detected: a sentence can be composed with the given values, the repetition begins from the invalid word detected
  + otherwise: repeat the sentence..

# Efficiency

Given that the process in which we determine our solution is basically a form of iterative tree traversal, the proposed tool works in amortized linear time with respect to the size of the knowledge base for the detected intent, with a medium constant.

However, with a good flag definition, the complexity can be reduced to the order of the intent subset subtree(an inset of the tree) due to advantageous flag restrictions(i.e. with a low level of branching, in the best case it can be reduced to depth(inset) from size(inset) )

On the other hand, an adversarial flag placement can create circumstances in which the complexity reaches the size of the whole knowledge base - yet such circumstances are to be avoided.

Please note that the aforementioned worst case complexity is equal in magnitude to the brute force approach - so, in essence, the tool has a heuristic aspect : it can and does perform better on average than a bruteforce method, although in the worst case they are similar.