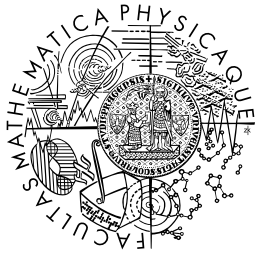


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jan Pavelka

Object layout in a 2D room based on text description

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis: Mgr. Rudolf Rosa, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2024

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I thank my supervisor Rudolf Rosa for his advice, insights, help and undying patience. I thank my friends Petr Kupka and Jan Brokeš for providing example input data for the task. Last but not least, let me thank my family and friends for great support throughout the work on this thesis and my studies.

Title: Object layout in a 2D room based on text description

Author: Jan Pavelka

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Rudolf Rosa, Ph.D., Institute of Formal and Applied Linguistics

Abstract: This thesis presents a solution for generating structured description of a 2D map of a room from a bird's eye view based on textual description in Czech. It focuses on identifying physical objects and their mutual relative positions in the description. It describes linguistic phenomena of the information extraction and their usage in the implementation. It shows how syntactic parsing can be used for this task. Then, it uses a genetic algorithm to find a feasible layout of the extracted objects with respect to spatial constraints constructed from the extracted information.

Keywords: natural language processing, scene layout

Název práce: Rozmístění objektů ve 2D místnosti dle textového popisu

Autor: Jan Pavelka

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Rudolf Rosa, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Tato práce řeší úlohu generování strukturovaného popisu 2D plánu místnosti z ptačí perspektivy na základě textového popisu v češtině. Zaměřuje se na identifikaci fyzických objektů a jejich vzájemných relativních poloh ve vstupním popisu. Dále popisuje lingvistické jevy související s touto úlohou a jejich využití v implementaci. Ukazuje, jak lze při řešení úlohy využít syntaktické parsování. Pomocí genetického algoritmu hledá použitelné rozmístění extrahovaných objektů s ohledem na prostorové omezující podmínky zkonstruované z extrahovaných informací.

Keywords: zpracování přirozeného jazyka, rozmístění objektů

Contents

Introduction

This thesis aims to generate a 2D map of a room based on textual description. It extracts structured information about the shape of the room, the physical objects inside the room and their mutual positions from the text input written in natural Czech language (figure 1). Based on this information it assigns positional coordinates and a rotation angle to the objects. The output is a structured `json` file that describes the layout of the room from a bird's eye view and can be easily used as an input for a graphical module[1] to draw the map automatically. The original idea is to use this map generator for preparing RPG maps but it can be used for any object placement into a 2D scene based on text description.

Existing applications usually focus on creating RPG maps by hand or generating them randomly, some of them allow to set parameters (see section 1.1.2). There is a text-to-image ML model that aims to generate maps from textual description but the results are not satisfying and do not always respect the input completely. Many other works focus on 2D or 3D object layout which is a difficult problem on its own (see section 4.2.2).

The main focus of this thesis is on the linguistic analysis of the problem (section 3) and on the NLP part of the solution (section 5). With the aid of syntactic parser such as UDPipe I try to identify the correspondence of the syntactic relations between the words in the sentence and the spatial relation between real physical objects as well as their types and shapes. The dependency syntactic tree constructed by the syntactic parser can be inspected automatically and useful information (such as nouns refering to real objects or spatial prepositions and their arguments) can be extracted from it. This semantic information can be used to create computer representations of the real objects and placing rules which have to be satisfied.

Based on the extracted placing rules, actual representational geometric shapes in the area of the room are constructed from the identified items. The placement of the shapes is determined by a genetic algorithm which tries to find the best solution according to the evaluation of geometric constraints extracted from the placing rules (section 5.2).

The solution is a console application implemented in Python (chapter 6). It uses several Python libraries that can be installed by `pip` and few online services such as UDPipe for syntactic parsing and other natural language processing. The application design is modular. On the high level, the code can be described as a pipeline of five modules: Importing, Language Processing, information Extraction, Composing and Exporting. The mainstaysof the solution are information Extraction

[1]The graphical module is not part of the thesis. Though, a simple plotting module that is able to plot the result into Cartesian coordinate system is included.

```
Vejdete do čtvercové místnosti. Kolem stolu stojí dvě židle.
```

Figure 1: Example input. [*You enter a square room. There is a table and two chairs around it.*]

```
{
  "types": [
    {
      "name": "chair",
      "terms": ["židle", "křeslo", "sesle"],
      "geometry": [[0.0, 0.0], [0.4, 0.0], [0.4, 0.4], [0.0, 0.4], [0.0, 0.0]]
    },
    {
      "name": "table",
      "terms": ["stůl"],
      "geometry": [[0.0, 0.0], [1.6, 0.0], [1.6, 0.8], [0.0, 0.8], [0.0, 0.0]]
    }
  ],
  "instances": [
    {
      "id": 0,
      "type": "table",
      "position": [1, 2],
      "rotation": 0
    },
    {
      "id": 1,
      "type": "chair",
      "position": [0.6411693601696831, 2.62893973934946],
      "rotation": -0.18199051253175633
    },
    {
      "id": 2,
      "type": "chair",
      "position": [0.6285673132718861, 1.653892778868146],
      "rotation": 0
    },
    {
      "id": -1,
      "type": "default",
      "position": [0, 0],
      "rotation": 0
    }
  ]
}
```


Figure 2: Example output in the `json` format.

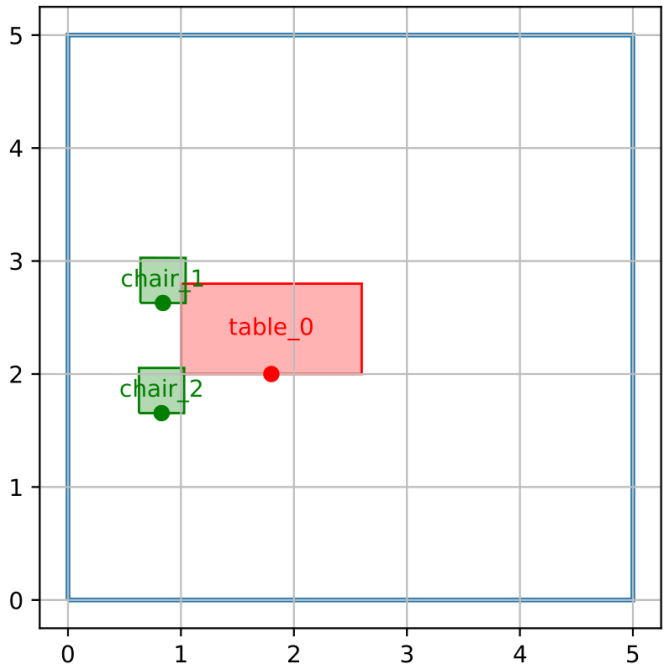


Figure 3: Plot of the output

which extracts the semantic information from a syntactically parsed text and Composing which implements the placing algorithm.

The usage of the application is straightforward (chapter 7). It simply processes the standard input (see figure 1) in form of raw text and generates output in the `json` format (see figure 3). Also, the results can be plotted by the application (see figure)

1. Problem

1.1 Background and Motivation

In tabletop role-playing games (RPGs) such as Dungeons & Dragons (D&D) each player except one assumes the role of a fictional character. The last player, usually called gamemaster (GM), describes the setting and the environment of a scene in which the characters are located. Based on these descriptions the players express their characters' emotions, reasonings and actions. If the actions are complicated (such as combat encounters), a good map of the environment is very handy.

Thus, many pre-made D&D adventure modules include illustrated maps of various locations which are essential to the story. Still, there might occur situations where the party decides to go to an unexpected place which has no map pre-made. Also, many players prefer playing their own campaign rather than running the oficially published modules.

Since the gamemaster often cannot anticipate players' decisions, some environments in which the characters occur are not included in his preparation. In such scenarios, the GM can draw an improvised map on a piece of paper by hand or find a similar one online and use it. But this approach takes some time and slows the game. To prevent this, GM's environmental descriprions could be used to generate the map automatically. After all, the players also imagine the scene based solely on the GM's description.

A tool that generates a map from textual description would be useful for the GM's preparation, too. It could be also combined whith a speech recognition software to produce the result in real time while the GM is speaking to the players.

1.1.1 Text and Map Description

D&D campaign modules include many useful tools for running adventure including maps of important locations (e. g. figure 1.1) and their descriptions (e. g. example 1).

The text is meant to be read aloud to the players while playing the game. It usually describes both the physical layout of the environment and the atmosphere of it. It can also contain further information needed for the game such as the presence of non-player characters or a magic aura, for instance. But these pieces of information are irrelevant for generating the map of the room.

Text example 1 (Example from an official module (DnD [2014])). *Three large stone sarcophagi stand within this dusty crypt, and propped up against each sarcophagus is a human skeleton clad in bits of rusty mail. False columns along the walls are carved in the image of spreading oak trees. The double doors in the southeast corner are sheathed in tarnished copper plate.*

The map depicts usually a room (or multiple rooms) from a bird's eye view. It is supposed to show the positions of the walls, windows and doors as well as

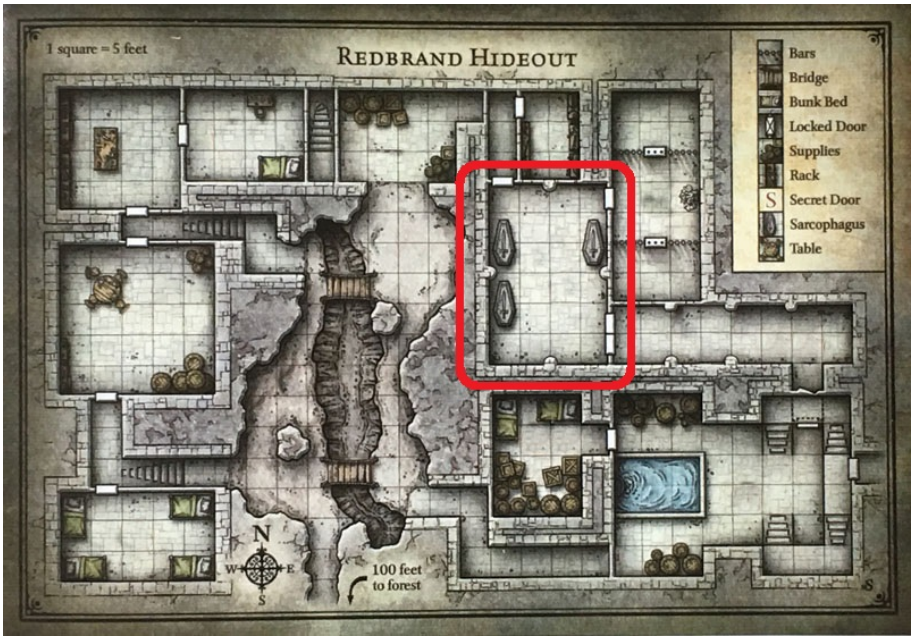


Figure 1.1: Actual map illustation from an oficially published D&D module (DnD [2014]). The red rectangle denotes the room discribed in example 1

the furniture such as beds, tables, chairs, barrels, wardrobes etc. Besides it, it should also contain items that are essential to the game.

The rooms are usually constructed with respect to an 1-inch square maze. The maze is drawn over the map itself which makes easier to place miniatures of the characters on the map while playing the game. The items are preferably aligned to the maze if possible. 1 inch on the map usually represents 5 feet of real distance.

1.1.2 Existing Applications

There are tools that can help the gamemaster prepare the maps. However, the vast majority of them does not accept input in form of textual description.

donjon d20 Random Dungeon Generator [1] is one of random generaotrs that generate a dungeon (multiple rooms interconnected) based on specified parameters.

Kasson House Map Generator [2] generates a house map which is visually very close to the oficially published maps. Nonetheless, it generates it randomly and not many parameters can be specified.

Dungeon Alchemyst [3] is recently one of the most popular software tools for generating dungeon maps. Though it is very powerful and let the user customize the random generated map until they are satisfied, it does not accept textual description as an input.

[1] `https://donjon.bin.sh/d20/dungeon/`
[2] `https://www.kassoon.com/dnd/house-map-generator/`
[3] `https://www.dungeonalchemist.com/`

AI Map Generator [4] produces large-scale maps of larger areas rather than a single building or room. And many results are not depicted from a bird's eye view. You can be lucky and get a satisfying result, though.

Even though it is the closest solution to what I attempt, it focuses on a global view rather than details such as mentioned objects that should be placed in the environment.

1.2 Problem Specification

Since the RPG map should be easy to use and practical in the first place, I focus on things that are important for the game. Things in the environment that the characters can interact with (e. g. chairs, tables and other objects in the room). My goal is not to draw a map with high aesthetic quality which corresponds to the feeling of the textual description but rather to detect as many physical objects as possible and their mutual relative positions and put them on the map. Thus, the output is in form of structured information instead of an image.

1.2.1 Input

The input for the task is a textual description of a single room in form of a raw text string. The text is in Czech language and has a natural structure. No specific syntactic or morphological constructions are required. To expect a proper result, the text must speak about a room and mention how it looks and what physical objects are inside and where they are placed.

> **Text example 2.** *V místnosti stojí dvě čalouněné židle kolem bohatě prostřeného stolu.*
> *[In the room there are two upholstered chairs around a richly laid table.]*

Language

I chose to work with the Czech inputs because although there are applications that solve similar tasks (see section 1.1.2), they usually work with English. Moreover, Czech is my mother tongue and my sense for Czech is better than my sense for English. On the other hand, I tried to make the solution as language-independent as possible.

1.2.2 Output

The goal of the solution is to produce structured information about the room described in the input. It should be easily processable by a graphical module so it could be used for drawing the map automatically. This means that the output should contain

- shape of the room described by sequence of coordinates of the shape's vertices;

[4] `https://perchance.org/ai-map-generator/`

- list of items that are present in the room including
  - their shape – also described by the coordinates,
  - their position,
  - their rotation.

The output file is formatted as `json`.


```
{
  "types": [
    {
      "name": "table",
      "terms": ["stůl"],
      "geometry": [[0.0, 0.0], [1.6, 0.0], [1.6, 0.8], [0.0, 0.8], [0.0, 0.0]]
    },
    {
      "name": "chair",
      "terms": ["židle", "křeslo", "sesle"],
      "geometry": [[0.0, 0.0], [0.4, 0.0], [0.4, 0.4], [0.0, 0.4], [0.0, 0.0]]
    }
  ],
  "instances": [
    {
      "id": 0,
      "type": "chair",
      "position": [2.5937934771450855, 0.7140583463594873],
      "rotation": 180
    },
    {
      "id": 1,
      "type": "chair",
      "position": [2.6518019915106588, 0.13278771448164994],
      "rotation": 180
    },
    {
      "id": 2,
      "type": "table",
      "position": [1, 0],
      "rotation": 0
    },
    {
      "id": -1,
      "type": "default",
      "position": [0, 0],
      "rotation": 0
    }
  ]
}
```

2. Solution

2.1 Data collection

Although there exist many officially published adventure modules that contain many human-illustrated maps and textual description of places, they are not publically available for using. Also, the vast majority of them are in English and they have no Czech translations. Even if I were able to obtain rights to use them, it would take a huge amount of time to collect them and prepare them for computer processing, since they are usually only available in printed physical books.

There are online services that are able to generate textual description for RPGs.

dScryb [1] is a large database of textual descriptions of places, items, charcters etc. They should be all human-written. You can also pay them to create a new description for you.

ChatGPT [2] or other large language models can be also used for generating text.

These sources provide only texts and no graphical output but they can be used for inspecting linguistic phenomena which must be dealt with. But again, these produce English outputs.

I am not aware of any specialized application capable of generating room descriptions in Czech. machine translation (MT) or general language models could be used but I rather chose to write small amount of high-quality data manually. I also asked my friends who are experienced gamemasters to write some textual room descriptions that could be used while playing RPGs. These descriptions are considered gold data since they are written manually by humans. The texts are in Czech and contain gramatical errors and typos.

I did not put any restrictions on how the descriptions should look like because I wanted them to be as natural and unbiased as possible.

2.2 Approach

The data I have collected are not sufficient for training a machine learning model. I have only few dozens of room textual descriptions and no target data. So, I chose a different approach to the solution. The approach is primarily rule-based but includes the usage of UDPipe[3], an online available ML model (Straka and Straková [2016]).

The solution focuses on finding items which are said to be present in the room and placing them according to the information in the input description. Much of this information is expressed using lexical, morphological and syntactic features of a language (Ursini [2010]). UDPipe is able to extract these features from the text. From these features, structured description of the scene can be constructed.

[1] `https://dscryb.com/`

[2] `https://chat.openai.com/auth/login`

[3] `https://lindat.mff.cuni.cz/services/udpipe/`

In the first place, let me inspect part of speech (PoS) tags. I am especially interested in nouns, prepositions and numerals. Most often, nouns refer to real objects. If nouns are detected in the input text, the corresponding objects can be placed into the scene. Prepositions describe relations between the nouns, including the spatial relations (Mlu [1986]). Based on them, spatial rules can be constructed which express the mutual relative positions between the objects. Finally, the numerals modify the nouns and express the number of the corresponding objects reffered by the nouns.

Other linguistic features, especially the position of the word tokens in a syntactic tree of the input sentence, enable to find the relations between the words, and therefore the relations between the real objects.

If I am able to extract objects and placing rules from the input text, I can use a placing algorithm to get a layout which respects the rules and thus the input textual description. In most cases, there is no single correct solution since one description can correspond to many possible layouts of the room. Some solutions will be probably better or more natural than others. The goal is to choose the best one from the candidates the algorithm produces. This can be achieved by an evolutionary algorithm (Sanchez et al. [2003]) which mutates the candidates over multiple generations and in each generation, it only lets the fittest ones survive. At the end, there should be a solution which is significantly better than the original ones that were generated at random during the initilaization of the evolutionary algorithm.

3. Linguistic Analysis

The input text can be analyzed on various levels. Since the goal is to extract real-world information, it would be ideal to obtain a semantic analysis of the input text. However, this is a very complex problem in general. There are many tools for extracting specific semantic information (such as sentiment analysis, information retrieval etc.). To a certain extent, these tools usually rely on syntactic parsing of the text and this is also the case of this project.

Syntactic parsing is usually easier because it is closer to the surface level of the text. I use UDPipe[1] online service for syntactic parsing. This tool accepts a raw text string and outputs a dependency syntactic tree in Universal Dependencies (UD) format. Each node of the tree corresponds to a single token (word, number, punctuation mark etc.) in a sentence. This tree can be then processed automatically and based on its structure semantic connections between words in the sentence can be detected (e. g. spatial relations between the objects denoted by the words).

Let me now focus on the linguistic aspects of the task. This linguistic knowledge will then be used to solve the problem of extracting information from the text.

3.1 Lemmatization

Definition 1 (Lemma)**.** *Lemma is the canonical form of a word. It is the chosen representant of all inflectional forms of the word.*

Definition 2 (Lemmatization)**.** *Lemmatization is the process of labeling tokens*[2] *in a given text with their lemmata.*

Lemmatization is critical in NLP because it reduces the number of text strings that refer to the same entity. UDPipe performs it automatically and the nodes of the output syntactic trees are annotated with their lemmata.

The canonical form is usually the one that occurs as a dictionary entry. In Czech and many other inflected languages, it is the nominative case singular number for nouns (e. g. *stolu*, *stoly*, *stole* all have lemma *stůl* [*table*] because they differ only in an inflectional suffix) and adjectives (e. g. *dlouhý*, *dlouhá*, *dlouhému* have lemma *dlouhý* [*long*]) and the non-vocalized form of prepositions (*v* and *ve* are forms of *v* [*in*]).

3.2 Nouns

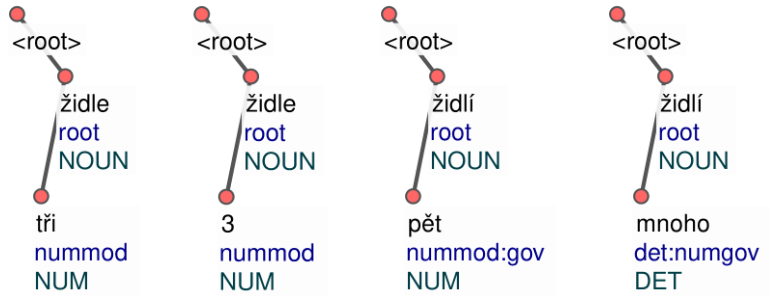
Nouns (n.s) are names for both physical and non-physical entities. Since my input data contain almost exclusively the nouns that denote physical objects, I presume to genralize this and consider all nouns in the input to refer to the physical objects to be placed in the scene.

[1] `https://lindat.mff.cuni.cz/services/udpipe/`

[2] In NLP, usually all tokens are lemmatized, including punctuation marks.

Grammatical number	Actual number	Example
Singular	1	*stůl* [*table*]
Dual	2	*se třema nohama* [*with three*$_{dl}$ *legs*$_{dl}$]
Plural	≥ 2	*vedle postelí* [*next to the beds*]

Table 3.1: Gramatical numbers and examples



(a) *tři židle* [*three chairs*] (b) *3 židle* [*3 chairs*] (c) *pět židlí* [*five chairs*] (d) *mnoho židlí* [*many chairs*]

Figure 3.1: UD representation of numeral. The structure is the same in all cases.

Nouns forms express gramatical number (singular, dual[3] and plural). This can be used to determine number of objects that are denoted by a single noun (see table 3.1).

Noun lemmata can be assigned to the types of the objects (e. g. chairs) they denote. One type of objects can be denoted by multiple synonyms (e. g. *židle, sesle, křeslo, trůn* are all synonyms that denote (various kinds of) a chair). Thanks to the lemmatization, I can only list the synonyms in their canonical form.

3.3 Numerals and quantifiers

Numerals (num.s) express numbers of entities. They modify nouns, especially their count or amount. In the UD (see 3.1), their node is always governed by the node of the noun that they modify, even though in Czech, their syntax is somewhat complicated[4] They can be used to determine the number of the governing noun's objects if the noun is in plural.

There are also other, vague, quantifiers besides the concrete numerals (e. g. *několik* [*several*], *mnoho* [*many*]). In Czech, they are also traditionally considered numerals but in the UD they are labeled as determiners.

[3]Very rare in contemporary Czech. Almost exclusively used only for parts of body that are in pairs (legs, arms, eyes, ears etc.). In metaphorical meanings (such as leg of a table) plural forms are used.

[4]They act as adjectives in the case of 1, 2, 3, 4 and share the case of the governing noun. But in the case of 5, 6, 7 etc. they act as nouns and determine the case of the governing noun.

3.4 Prepositions

The main focus of this theses is on prepositions because most spatial information is expressed by them[5].

Prepositions (prep.s) are adpositions (adp.s) that stand before their complement. In the Czech language, they are an uninflectable, synsemantic part of speech (Mlu [1986]). Syntactically, they are inseparable from their complement, and they determine their complement's case. Rather than having full sense on their own, they express spatial, temporal, genetic, and other relations between other words in a sentence (Biskup [2017]).

3.4.1 Etymological classification

Etymologically, prepositions are divided into proper (primary) and improper (secondary).

Proper prepositions

Proper prepositions are original monomorphemic short words, typically nonsyllabic (*k* [*towards*], *s* [*with*], *v* [*in*], *z* [*from*]) or monosyllabic (*o* [*about*], *u* [*at*], *na* [*on*], *od* [*from*], *do* [*into*], etc.). They are not derived. They are a closed category thus all nineteen of them (Biskup [2017]) can be listed:

> **Text example 3** (Proper prepositions)**.** *Bez [without], do [(in)to], k [to(wards)], na [on], nad [above], o [about], ob [every other], od [from], po [after, along the surface of], pod [under], pro [for], před [before], přes [over], při [at, by], s [with], u [at], v [in], z [from], za [behind] (Mlu [1986])*

Excluding *bez* [*without*] and *pro* [*for*], each of them also has at least one spatial sense (Slo [1960-1971]). Since they are a closed category and the most common prepositions in the language, they should be covered by the solution.

Improper prepositions

Improper prepositions are fixed forms of words originally assigned to other parts of speech. They often express more specific or emphasized sense than the primary prepositions.

> *v* [*in*] (prep.) + *prostřed* [*middle*] (n.) → *vprostřed* [*in the middle*] (adv.) → *vprostřed místnosti* [*in the middle of the room*] (prep. + n.)

Usually, a human can identify them easily since they share their form with the original word. But they are an open category and there are many of them with many various senses. I tried to cover at least the most common ones from the localization list in Čermák [1996]:

[5]This section covers prepositional phenomena that are relevant for the task (see attachment A.2 for mor linguistic details about prepositions)

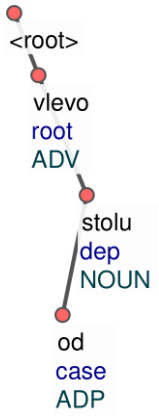


Figure 3.2: *vlevo od* [*on the left of*] is composed of two nodes, *vlevo* [*on the left*] is considered to be an adverb and *od* [*of*] a preposition

.

Text example 4 (Covered prepositions)**.** *blízko, blíže, dle, do, doprostřed, dovnitř, k, kol, kolem, kraj, mezi, mimo, na, nad, naproti, napříč, naspod/u, navrch/u, nedaleko, od, okolo, opodál, po, poblíž, pod, podél, podle, prostředkem, proti, před, přes, při, skrz, stranou, středem, u, uprostřed, uvnitř, v, vedle, v(e)prostřed, vespod/u, vevnitř, via, vně, vprostřed, vstříc, z, za, zespod/a, zevnitř, zeza, zkraje, zpod, zpoza, zprostřed; do blízkosti, na konci, na začátku, směrem do/k/na/proti, ve vzdálenosti, v okruhu, ze vzdálenosti*

Of course, more prepositions can be added to the list. Some of the impoper prepositions are multi-word (freezed constructions originally). In UD, each token has a separate node and the non-prepositional part of the construction is annotated as the former PoS of the word (see figure 3.2).

3.4.2 Syntax

In Czech, the preposition always stands before its complement.[6] The complement is a noun, pronoun, or their equivalent. It must be present unless the preposition is a part of an idiom (*být pro* [*be in favor (of a proposition)*], *nahoře bez* [*topless*]). Furthermore, the complement cannot be shifted across the sentence but can be modified by an attribute (*pod dřevěnou židlí* [*under wooden chair*]) (Biskup [2017]). In extreme cases, this results in multiple prepositions in a row (*žádný z na smrt unavených lidí* [*none of dead tired people*] (Machálek [2014])).

3.4.3 Semantics

Primary prepositions are usually modeled as two-argument predicates that express the relation between the dominating and the dominated argument (Mlu [1986]).

[6]Except for a few exceptional postpositions (postp.s) (e. g. *protestům* (n.) navzdory (postp.) [*despite protests*]).

Definition 3. *The dominated argument is the preposition's complement and is called internal (or **ground**).*

> *hrnek na stole* [a mug on the table] – *stole* [table_L] is the ground

Definition 4. *The dominating argument is a noun, a verb, or the whole proposition and is called external (or **figure**).*

> *hrnek na stole* [a mug on the table] – *hrnek* is the figure; *na stole* is the attribute of *hrnek*
> *Na zahradě se budou slavit narozeniny.* [Birthday will be celebrated in the garden.] – the whole sentence is the figure, *Na zahradě* is an adverbial

General semantic features

Definition 5. *General relational meanings of the prepositions can be defined using these three binary features.*

1. ***Staticity – dynamicity**. Determines whether the figure is in motion (either literally or figuratively). In Czech, this feature is often expressed by the ground's case rather than the preposition itself.*

   - *Sedím na podlaze. [I am sitting on the floor.] (na (L) [on] – static)*
   - *Sednu na podlahu. [I am going to sit down on the floor.] (na (A) [on] – dynamic)*

2. ***Directedness – undirectedness**. Directed prepositions need the ground to be oriented. Undirected prepositions lack the information of direction.*

   - *před domem [in front of the house] (directed)*
   - *u domu [at the house] (undirected)*

3. ***Contact – lack of contact**. Contact prepositions suggest that the figure is in touch with the ground at some point.*

   - *táhnout na Bělehrad [to march on Belgrade] (contactless)*
   - *ležet na stole [to lie on the table] (contact)*

Staticity–dynamicity is irrelevant for the task since the scene is considered to be always static. Even dynamic prepositional constructions can be used when describing a static scene, though (*židle byla postavena ke zdi* [a chair was placed to the wall]). For the purpose of the task, such constructions can be merged with their static counterparts (*židle je postavena u zdi* [a chair is placed at the

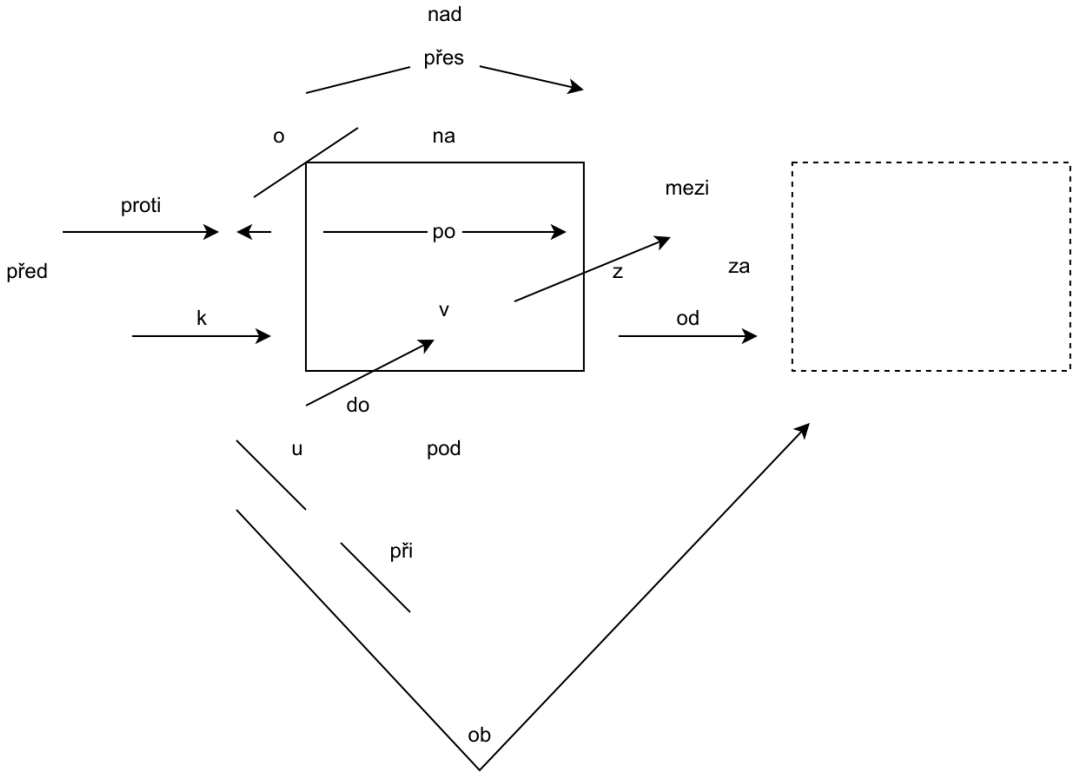


Figure 3.3: Conceptual meaning of the primary (spatial) prepositions. The continuous-line rectangle represents the ground, and the prepositions show the relative position of the figure with respect to the ground. (Mlu [1986])

wall]). After all, in many cases, only the case of the complement differs while the preposition itself remains unchanged.

The latter two features are critical for represnting the meanings of the prepositions. The contact can be easily encoded as physical distance. Undirected prepositions are relatively simple, too, since they do not require the knowledge of the ground's and figure's sptial orientation.

Directed prepositions are more complicated in terms of extracting spatial information (see section 3.4.4).

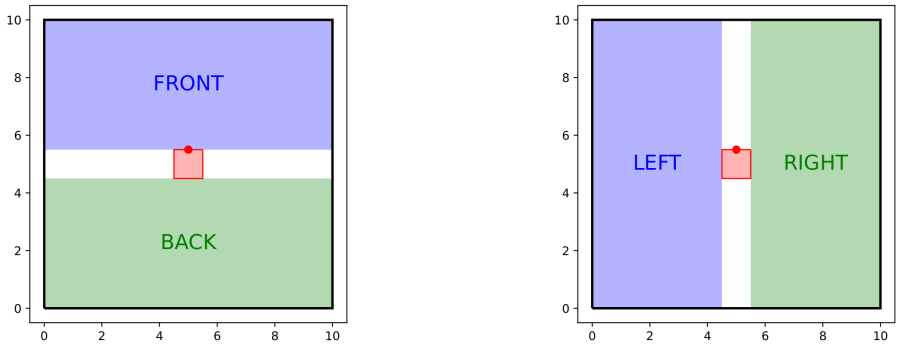
3.4.4 Semantics of Directed Prepositions

Modeling semantic meanings of directed prepositions (such as *před, za, vlevo od, vpravo od* [*in front of, behind, on the left of, on the right of*]) is somewhat complicated.

Since the task is two-dimensional, the prepositions that speak about the vertical position can be reduced to undirected horizontal prepositions.

> *nad, pod, přes* [*above, under, over*] $\rightarrow$ *v* [*in*]

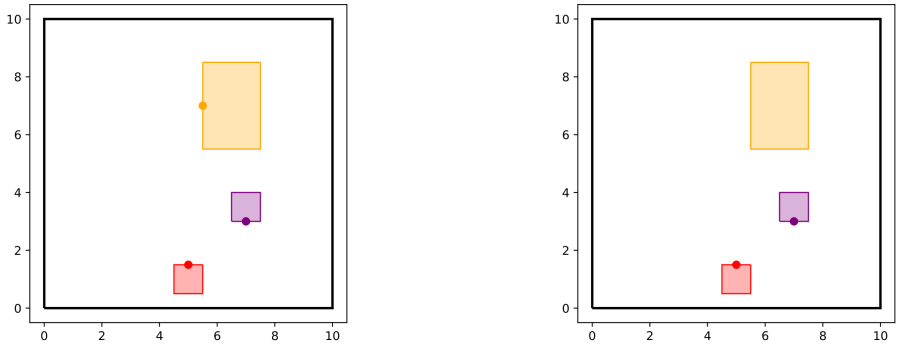
During the implementation of the project, it emerged that the human understanding of horizontally directed prepositions is ambiguous. It depenends on the position of the observer (1st or 2nd person) as wel as the orientation of the



(a) Front and back sides

(b) Left and right side

Figure 3.4: Observer's point of view



(a) If the ground has a natural face (e. g. a wardrobe), the observer says "the chair is on the right side of the wardrobe"

(b) If the ground has no face (e. g. a table), the observer says: "the chair is in front of the table" The observers assumes that the ground is facing them.

Figure 3.5: Directed vs. undirected (faceless) objects. The red suare is an observer, the purple square is a figure (e. g. a chair) and the orange rectangle is a ground. Points denote their faces.

objects.

If the ground is the observer themself, it is straightforward (see figure 3.4). The areas of the figure's possible location is a half-plane which can be easily defined based solely on the orientation of the observer.

If the observer uses a directed preposition while describing positions of other figures, things get more complicated (see figure 3.5). The orientation of the objects must be considered as well as the observer's relative position to the objects.

If the ground object has a natural front side (let me call it **face**) such as the door of a wardrobe, the observer usually tend to construct the front and back half-planes with respect to the ground object orientation, but the left and right half-planes are switched (see figure 3.5(a)).

If the ground object has no natural face side and no orientation anchoring in the space (e. g. a tree in a forest), the observer usually describes the layout as if

the ground object's face was facing them (see figure 3.5(b)).

The situation in my task is even more complicated since the room is usually described through the eyes of a (hypothetical) observer who is present in the room and, on the other hand, the output should be constructed from a bird's eye perspective. In most cases, you get into a room through a door, so the input descriptions usually assumes the position of the observer at the entrance. Then, the natural orientation of a map is such that the entrance is at the bottom.

I came up with three possible approaches of processing the directed prepositions (see 3.6).

1. The half-planes around the objects are constructed with respect to the map as a whole (figure 3.6(a)). That means that all objects are facing to the bottom of the map.

   This is the easiest approach to implement. The directed spatial relations are simply based on comparing the horizontal and vertical coordinates on the map. Even though this can be sufficient for describing positions of the objects on a map, it turned out that the results are not very natural because when people read the description, they imagine the actual three-dimensional space of the room as if they were inside rather than a two-dimensional map of the room.

2. The middle of the bottom margin of the map is chosen to be the position of the observer and the half-planes around the objects are constructed with respect to it (figure 3.6(b)).

   This partially fixes the problem of unnatural results. On the other hand, rooms are closed spaces and usually have a significantly bound structure. This approach would work much better in an open space where such rigid borders as walls are not present.

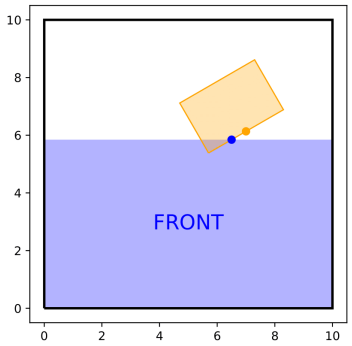
   The problem of the room is that the side of the object that faces the wall is considered as a back regardles of the position of the observer.

3. Each object has a defined face and the half-planes around it are constructed base on its own orientation (figure 3.6(c)). Even the naturally faceless objects are assigned a face. Defaultly, the face faces to the bottom of the map. This means that if there is no reason for change the facing, this approach reduces to the first one.

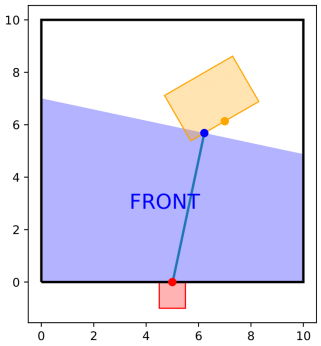
   The default face direction could be also the observer from the approach 2, but since the map is meant to be used with a grid, I chose to perserve the Cartesian orthogonal structure of the object layout as much as possible. This should be the optimal compromise between naturality and practicality.

I performed few real-life experiments to choose the best approach. I pointed at real objects and asked people where the object is placed with respect to them, to me or other objects in the space. The experiments support the choice of the third approach.

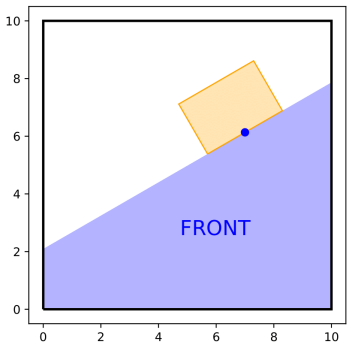
But many descriptions are still ambiguous and it seems there is no way to distinguish them. Usually one sense is more probable than the others. The experiments showed that the approach 3 covers the most expected meanings.



(a) Each object is facing down.



(b) Each object is facing the observer.



(c) Each objects has its original face.

Figure 3.6: Approaches of processing directed prepositions. Only the front half-plane is displayed. The blue point denotes the assumed face of the object based on the approaches. The orange point denotes the real face of the object.

3.4.5 Prepositional structures in UD

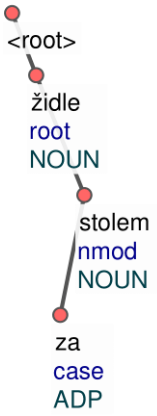
In UD (see figure 3.7), the preposition are governed by their complement and the dependency relation between them is annotated as `case`. The PoS tag is `ADP` (adposition). The Feats of the preposition's node include a key-value pair `AdpType=Prep`. The preposition's complement is the root of the whole prepositional phrase and is governed by a node which corresponds to the external argument of the preposition (see section 3.4.3). The described phenomena such as multi-word prepositions (see section 3.4) can complicate the structure of the UD tree but are quite uncommon.

For my task, I need to consider multiple cases of the tree structure (see figure 3.7). If the parent of the preposition node is a noun (figure 3.7(a)), it is considered to be the figure. In other cases (3.7(b), 3.7(c), 3.7(d)), the external argument is formally the rest of the proposition as a hole. But I need to know which physical object is relevant for the spatial relation expressed by the preposition. Luckily, it seems that the structure is similar in all of the cases. The parent is (part of) a predicate of the sentence. To find the noun relevant for the physical object about which the text speaks, we need to iterate through the children of the predicate nodes (the predicate can be composed of more nodes (figure 3.7(f)), I take all of them into account). Among these children I choose all the nouns except the one that is part of the prepositional phrase as the candidates (so in figure 3.7(e) the candidates are *oko* [*eye*] and *židli* [*chair*], in the other cases it is a single node *židli* [*chair*]). From the candidates, the most probable one is chosen.

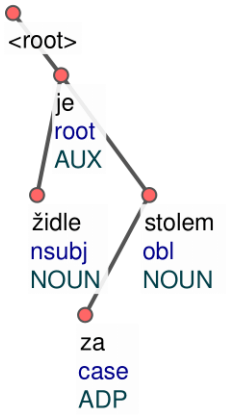
It is not clear which candidate is the most probable one if there are multiple candidates. My choosing strategy is based on what I have seen in the data and my experience. If the sentence has an object (*židli* [*chair*] in figure 3.7(e)), I choose it rather than the subject (*oko* [*eye*]) since the subject is usually an actor (the one who looks at the scene and describes it). If there is no object among the candidates, the subject is chosen. As a backup solution, if no objects nor subjects are found, the first candidate is considered to be the most probable. This should be a rare case and I was not able to come up with an example of this.

3.5 Coordination

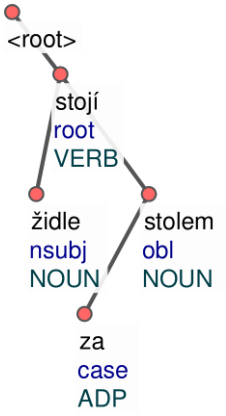
Both prepositional arguments (ground and figure) can be coordinated. In UD (figure 3.8), coordination is denoted by `conj` dependency relation, thus it is easy to extract it from the tree. When processing, the coordinated figures are distributed to the grounds, so in the end there is an instance of `Preposition` class for each pair of a figure and a ground. Prepositions that have multiple semmantic arguments (*mezi* [*between, among*]) are not currently supported.



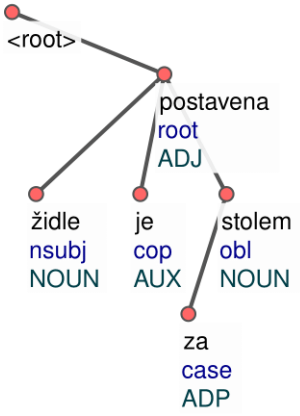
(a) *židle za stolem* [*chair behind table*]



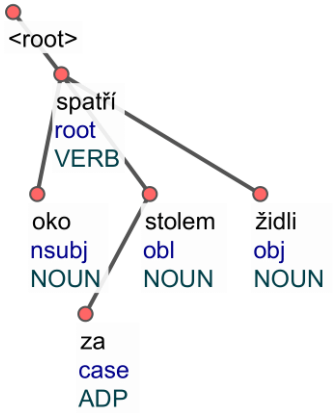
(b) *židle je za stolem* [*chair is behind table*]



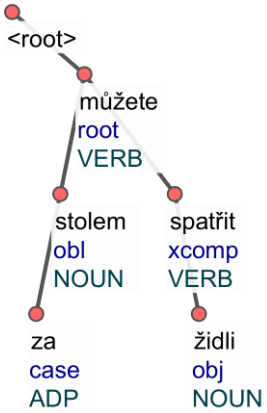
(c) *židle stojí za stolem* [*chair stands behind table*]



(d) *židle je postavena za stolem* [*chair is placed behind table*]



(e) *židli vaše oči spatří za stolem* [*your eyes can see a chair behind the table*]



(f) *oko spatří za stolem židli* [*an eye can see a chair behind the table*]

Figure 3.7: UD prepositional structure

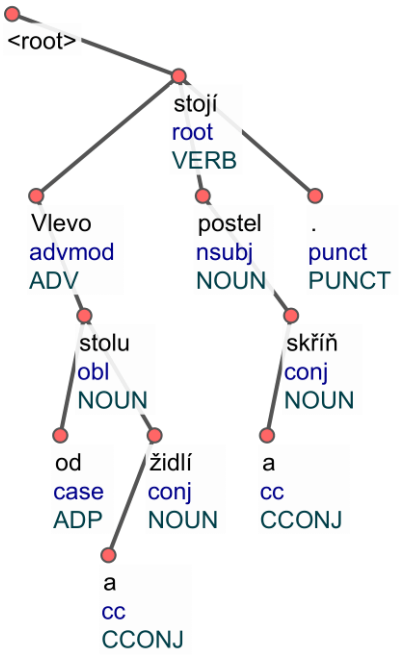


Figure 3.8: Coordination in UD

4. Layout Problem

This part of the solution aims to place given physical objects in a specified space based on given constraints. Since the result should be a two-dimensional (2D) map of a room, I focues on a 2D version of the problem. Even though it is simpler that the three-dimensional (3D) version of this problem, it is still a difficult and time consuming task (Xu et al. [2002]).

4.1 Specification

Constraind-based object placement problem is a task of finding such object placement that the constraints are satisfied. The positions are restricted by given area.

Input The input of the problem consists of:

- a list $\mathcal{O}$ of objects to be placed

  Each object is defined by its shape. It is a polygon, a line segment string or a point.

- an area A of the scene

  The area is a 2D polygon. Each object should be placed inside the area completely.

- a set $\mathcal{C}$ of abstract constraints to be satisfied

  A constraint C is a mapping $C : \mathcal{O}^2 \to \{0, 1\}$ which assigns an evaluation to the pair of objects

Output The output is a list $\mathcal{P}$ of so-called placings. Each placing is assigned to a single object and each object has a single placing. A placing P of object O consists of

- position

  Coordinates where the object O is placed.

- rotation

  Angle of the rotation of the object O.

The task is to find placings $\mathcal{P}$ of the objects such that:

1. all constraints $\mathcal{C}$ are satisfied (hard constraints, basically deicision problem),
2. as many constraints are satisfied as possible (hard constraints, optimization problem),
3. or constraints are satisfied as much as possible (soft constraints, optimization problem).

   In this case the constraints are fuzzy and the range of the evaluation is $[0, 1]$).

Since the constraints are not guaranteed to be entirely correct (they are extracted automatically from a text) and the objects are not guaranteed to fit into the area, the problem does not necesseraly have a solution for hard constraints. For this reason, I chose the third version of the problem.

4.2 Previous Works

Most of the papers I was abel to find solve 3D version of the problem. I work with 2D environment but the ideas usually work for me.

The classical version of the problem is known to be NP-complete which means that the computation time is expected to be long. The computation time can be reduced by restricting the task to isothetic[1] systems (Sanchez et al. [2003]).

4.2.1 Techniques and ideas

Let me now briefly mention some relevant placing techniques and which ideas they ispire (see Xu et al. [2002] for more details, they focus on a slightly different task but the ideas can be reused):

Reducing the number of degrees of freedom (DOF) Objects can be snapped to grids or auxiliary helper geometry. Their placement can also be restricted to a smaller area.

My objects should be snapped to the grid if possible, although this requirement is not very strict. So, their initial position is snapped to the grid but during the placement process, they can be moved and rotated.

The idea of restricting placement area can be used for `inside` constraints. This can be combined with the *devide-and-conquer* technique. Once smaller objects are placed inside a larger one which is their placement area, they are done. Then, only the larger object must be then placed into its area and so on.

Pseudo-physical techniques Objects are placed to physically stable positions. Objects are moved according to the defined pseudo-physical laws until they find their stable position.

This technique is used indirectly. I only add default physical constraints that must be satisfied always. These constraint represent the general physical properties of the objects (they must not overlap) and the spatial restriction of their placing (they must be inside the placement area).

Semantic techniques Objects are placed according to given semantic constraints.

These are in fact what is described throughout this chapter.

[1] Paralel and/or perpendicular to the coordinate system.

4.2.2 Algorithms

Various approaches have been tried to solve the constraint-based placement problem. Let me list some common techniques, with notes on their adequacy in my setting (Sanchez et al. [2003]).

constraint satisfaction programing (CSP) works well for under-constrained problems which is a typical scenario in 3D as well as in my case. However, the results of such algorithm is often unrealistic and artificial hard constraints must be added to improve them. The traditional CSP solves the first variant of the problem – the constraints are hard and the solution must satisfy all of them.

Numerical optimization (linear or non-linear programing) works well but only if strict restrictions are set to the constraint systems.

Metaheuristics (local search techniques, evolutionist strategies)

Genetic algorithm (GA) works with soft constraints naturally and it does not need the constraints to be differntiable. On the other hand, the right choice of hyperparameters and genetic operators is highly dependent on the problem and is not easy.

Machine learning (ML) techniques might be powerful but there are not enough available data for training.

4.3 My Approach: Genetic Algorithm

For my project, I chose the genetic algorithm solution for the following reasons:

- It works well with the soft constraints which I prefer. The fitness function can be based on the evaluation of the constraints.
- Since it is an optimizing algorithm and each individual encodes a possible solution, it can be stopped at any time and the provided solution will always be valid (although it may not be optimal).
- It works well even with over-constrained problems. Since the solution is always valid, the computation can be stopped at any point and the result will be sufficient.
- It can be used interactively. During the evloution, new objects can be added without invalidating the solution.
- It is easy to implement and modify. Genetic operators can be changed and improved independently and they can take advantage of knowledge of the problem.
- My knowledge of GA is greater then the knowledge of other techniques that I considered.

The GA is an optimizing algorithm. It is based on the idea of Darwinian evolution. It is initialized with a population of individuals which encode possible solutions. From these individuals, the fittest ones are selected into a mating pool. Then, genetic operators (crossover and mutation) are applied on the individuals in the mating pool. In the end, the offsprings are combined with the population to form a new generation. This process repeats until an ending criterion is met.

5. Algorithmic Overview

The solution of the task can be divided into two main parts. The first part is the information extraction and is based on theory in chapter 3. The other part is the placement problem which was described in chapter 4.

5.1 Information extraction

The information extraction is performed in several steps (see figure 5.1). The steps transduce the data from one form to another. It turned out that these forms of data naturally correspond to the levels of language description (at least the first three of them).

5.1.1 Syntactic Parsing

First of all, we need to syntactically parse the input text (figures 5.1(a) $\rightarrow$ 5.1(b)). The text can be composed of multiple sentences. The sentences are tokenized, lemmatized and then parsed to a syntactic dependency tree. This is done by UDPipe. The result is not guaranteed to be always correct but it usually works fine.

To improve the results of the parsing, the text is automatically corrected using Korektor at first. This helps to get rid of typos, ortographical, and grammatical errors that could be present in the input.

5.1.2 Semantic Information Extraction

Using the linguistic knowledge in section 3, the syntactic tree is processed (figures 5.1(b) $\rightarrow$ 5.1(c)). This part is strictly rule-based and the rules are derived from the observed language phenomena (see 3). This means that the solution can be improved by adding more rules or rewriting the current ones.[1]

The current version of the program inspects the tree and look for mentions of physical objects, information about their spatial layout and global information about the room.

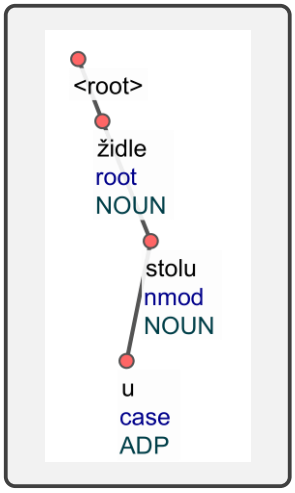
The goal is to get a list of item types and counts that can be found in the described room, a list of prepositional relations between the items that can be easily processed by computer and at least slight insight into the room's shape and size.

From the linguistic point of view, this part of the solution basically corresponds to the step from the syntactic to the semantic layer of language description, although the semantic description is very narrow and informal. From the syntactic trees we get a list of semantic entities and their relations, even though we are interested only in very specific features.

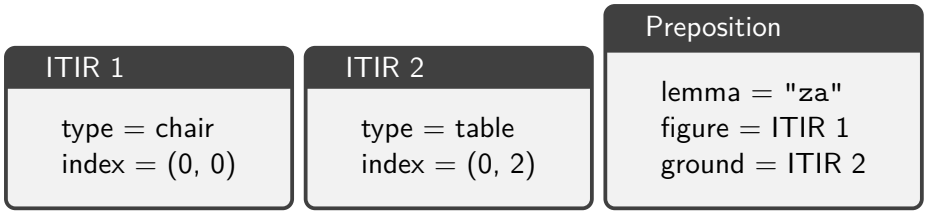
[1] This was experienced during the work many times. The original version was only able to find physical objects mentioned in the text and place them randomly.

[*chair at table*]

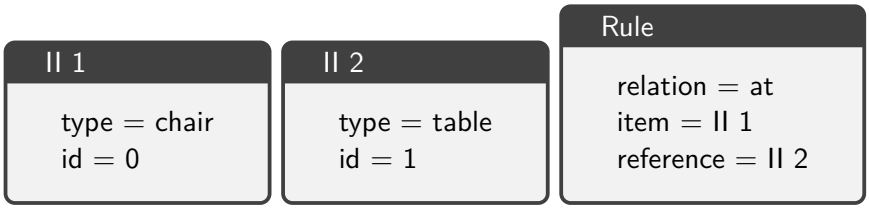
(a) Surface level. Raw string.



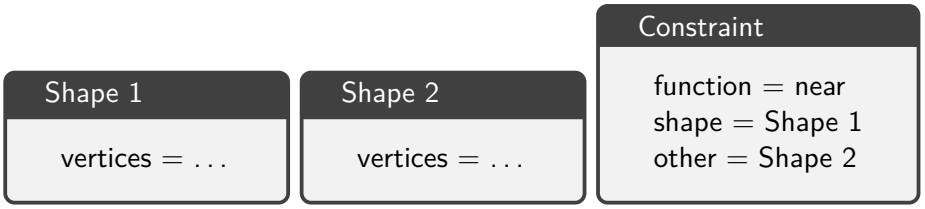
(b) Syntax level. Dependency tree representation.



(c) Semantic level. Item type index references represent the reference between the sentence tokens and real world entities. Prepositions represent semantic relations between them.



(d) Real object representation. Item instances represent real physical objects. Rules represent their mutual relative placements mentioned in the text.



(e) Geometric level. Shapes are extracted from the items and they are defined by concrete vertices (they are placed in the environment). The constraints are soft and they evaluate the quality of the placement of the shapes.

Figure 5.1: Extraction structure. The figure shows how the information is extracted from the input. The corresponding boxes are placed respectively.

Item Extraction

As described in section 3.2, each noun in the tree is considered to be a candidate for items. Firstly, I need to distinguish them, which means I have to determine their so-called **type**.

The solution requires a list of supported item types. Each type defines the shape of all items of that type. The type also lists all possible synonymous terms that can be used to refer to an item of that type. For example, type `chair` defines the shape[2] of square 0.4×0.4 and lists Czech synonyms that denote (various kinds of) chairs: *židle, sesle, křeslo, trůn*. The terms should be unique among the types – no term is allowed to be assigned to more than one type.

The candidates are iterated through and they are asigned the types based on their lemmata. If a lemma of a candidate is found among terms of a type, this type is assigned to a candidate and the candidate is inserted into the output list. If the lemma is not found in any type, a special type `unsupported` is assigned to it. From the point of view of the rest of the program, `unsupported` is considered to be a regular type with a predefined shape. The idea is that an unrecognized noun denotes also an item with a very high probability. These unsupported items can be set to be ignored, too.

The output of this step is a list of so-called **item type index references**. They combine the type object with the index of the corresponding node in the tree so that both the node and the type could be easily accesible.

Correferences Multiple words in the text may refer both to the same item or different items of the same time. This is called **correference** and it is a difficult task of NLP. This thesis does not attempt to solve this problem perfectly.

The most common case is pronoun which refer to entities already mentioned. This solution ignores pronouns completely. In the case of nouns, only first mention is taken into accounta and the latter nouns with the same lemma are considered to refer to the same item (or group of items if the nouns are in plural number).

Determining Number of Items For all the item type index references, the number of their instances must be determined. Again, the program inspects the tree. The noun nodes contains a morphological tag specifying its grammatical number. If it is singular, the number of the instances which are going to be created is set to 1. In the case of plural the closest preceding numeral node is seeked. If there is one, the number is extracted from the numeral. Otherwise, the number is chosen uniformly at random from $\{2, 3, 4, 5, 6\}$. The idea is that if the number is not mentioned in the description, it is not that important and the usual counts of things tend to be small.

Room Information Extraction

Room type is a special kind of item type in terms of extracting information. If there is a node with a lemma present in the room type's terms, regular item type index reference is constructed. In such case, there is a node in the tree that refers to the room. Otherwise, the index to the tree is set to an invalid value.

[2]Shape will be used later in section 5.2 for placing.

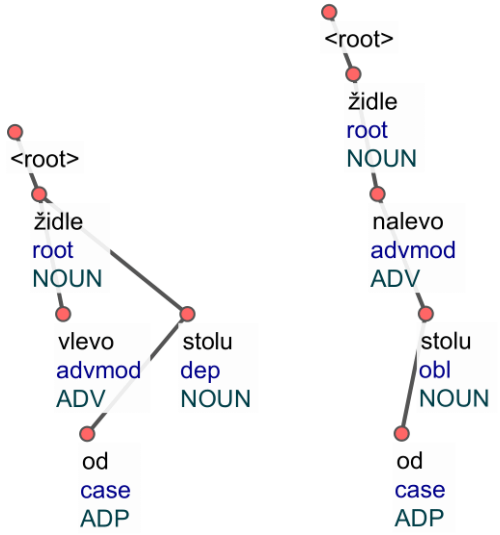


Figure 5.2: Results of parsing multi-word prepositions by UDPipe [*a chair on the left of the table*].

The concrete type of the room is determined by the attributes of the noun reffering to the room. These attributes can be found among the children of the noun's node. They are compared with the terms of supported room types and if there is a match, the appropriate room type is chosen. Otherwise, room type `default` is chosen.

Preposition Extraction

The goal is to construct semantic representation of a preposition. Such representation is defined as a binary relation between the figure and the ground (see section 3.4.3) where the figure and the ground are item type index references (i. e. typed tokens – they represent the in-between step between the sentence tokens on the syntactic level and the real-world object representation on the other side). The type of the relation is represented by the preposition's lemma.

Firstly, the prepositional nodes are filtered from the tree. This is achieved by inspecting PoS tag of the node and checking if the lemma of the preposition is present among the supported prepositions.

Since multi-word prepositions are very common in room descriptions and I want them covered, before the searching the ground and figure nodes, all parts of multi-word prepositions must be put together. However, UD treats each token separately and the tree structures vary (see figure 5.2). But I can take advantage of the collection of the supported prepositions. The collection also includes multi-word prepositions which can be easily distinguished because they contain a space. Then, the neighbors of the node can be checked if they correspond to one of the multi-word prepositions.

For each prepositial node (or couple of nodes in case of multi-word prepositions) the program tries to find the nodes that correspond to the figure and the ground. The search is based on the findings in section 3.4.5. It can find multiple nodes for both the figure and the ground if they are coordinated. In this case, both lists of the found nodes are distributed and in the output, a copy of the

prepositional relation is present for each figure-ground pair.

5.1.3 From Semantics to the Placing Problem

In this phase, I have a room representing item type index reference, a collection of item type index references, counts of their instances and a set of prepositional relations with their arguments. I want to get actual item instances and spatial rules (figures 5.1(c) $\rightarrow$ 5.1(d)).

Creating Item Instances

Creating actual instances of item types is straightforward. For each item type index reference, the given number of item instances of the corresponding type is created. The same holds for the instance of the room.

Each of the resulting item instances now represents a single real-world object (or room itself). These instances now will be placed according the rules constructed from the prepositions.

Constructing Spatial Rules

All supported prepositional relation types are defined by a single rule which represents the spatial restriction they express. The rules are binary relations between the item instances. They should be respected while placing the instances in order to construct an output corresponding to the input text.

The type of the rule is determined by the type of the preposition. If the preposition's ground or figure represent more then one instance, the rule is constracted for each figure-ground pair. The first argument of the rule is the item that corresponds to the figure and the second is the item that corresponds to the ground of the preposition.

There is a list of simple rules. There are also complex rules that combines restrictions of multiple simple rules. The idea is that some complex spatial relation between objects can be decomposed to multiple simpler ones that must all be satisfied at the same time.

5.1.4 Evaluating Placing using Geometric Constraints

The placing algorithm needs to be able to evaluate the potential positions and rotations of the item instances based on the rules (figures 5.1(d) $\rightarrow$ 5.1(e)). This is achieved by constructing actual geometric shapes from the instances, placing them according to the given positions and rotations and measuring how well they satisfy geometric constraints which are extracted from the rules.

Getting Shapes of the Instances

A shape is defined by its vertices. It can be 1D (point), 2D (line segment string) or 3D (polygon). The shapes of the item instances is defined by the type of the item. It is usually a polygon. The item type shape have the first vertex in the origin (coordinates $(0, 0)$). The actual shape of the instance is then constructed from the type's shape but is translated and rotated according to the given parameters.

During the constraint extraction, there might be created auxiliary shapes that do not represent any actual instance. They are critical for evaluating some constraints.

Constraints

The constraints are constructed from the simple rules. They are relations between two geometric shapes (points, line segment strings or polygons) and the area where the shapes are being placed. They must be evaluatable. Thus they must be as simple as possible and rely on the geometry of the shapes and the area.

In our approach, they are based on fuzzy logic which means that the result of their evaluation is not a binary value but a real number from $[0, 1]$. This value represents how well the constraint is satisfied. The higher the value is, the more satisfied the constrained is. This dependence should the same for all types of constraint if possible. I do not want to favor any constraint at the expense of the others. If the evaluation result is 1, it means that the result is completely satisfied and the evaluation cannot be improved.

It turns out, that all the supported rules can be in the end reduced to combination of four geometric constraints, although some auxiliary shapes that do not represent any real item instance are sometimes needed. The idea is based on the binary general semantic features of the prepositions (see 3.4.3).

Decomposing undirected rules (rules that were created from undirected prepositions) is relatively straightforward. They mostly correspond to a single constraint. Directed rules are more complicated. They need auxiliary shapes to express the proper location.

Since the construction of the constraints is more complicated, each simple rule needs a constructing function which produces constraints from the rule.

5.2 Placing

The input of the placing algorithm is a list of item instances, a set of rules that are expected to be satisfied and a room instance which specifies the area where the instances should be placed. Besides the rules extracted from the text, there are also natural physical constraints such as the items should not overlap (if it is not explicitely stated) and that all items should be placed inside the room.

The goal is to place each instance so that the rules are satisfied as much as possible. So, the output is list of positions and rotations for each item instance.

5.2.1 My Solution

My solution is based on the idea of a simple genetic algorithm. In the simple genetic algorithm, a population of individuals that encode the solution evolves. In each generation, some individuals are mutated and some mate and produce crossovered offsprings of themselves. Then the fittest individuals are selected for the next generation.

In my case, the individuals are placings of the item instances and the fitness function is based on the constraint evaluation.

Individuals Each individuals encodes a solution candidate. It contains placing of each item instance. Placings are composed of position and rotation. The position is a vector of two real numbers – coordinates of the item. The rotation is an angle (in degrees) between the face of the instance and a vertical ray directed down.

Population Initilization All individuals are initialized randomly. Since the layout should be as orthogonal as possible, the rotation initial value is chosen from $\{0, 90, 180, 270\}$ uniformly at random. The position coordinates are generated in such way that the whole item fits into the area of the room.

Genetic Operators The current solution includes only the mutation operator. If an individual is selected for the mutation, some of its placings are modified by adding a random value from normal distribution to the position or the rotation.

Selection I use $(\mu + \lambda)$ strategy which selects μ individuals from the combined population of μ parents and λ offsprings (mutants). Tournament selections is performed which iteratively selects two individuals and the fitter of the two is added to the new generation.

Fitness Function The fitness function is defined as the sum of all evaluations of the geometric constraints.

This means that in each generation of the algorithm new shapes and constraints must be generated because the shapes might have been shifed and therefore the auxiliary shapes are totally different. This slows the algorithm down significantly.

Result After specified number of generations, the fittest individual is returned as the best placing.

6. Implementation

The solution is implemented in Python. It uses online services using REST API and several non-standard libraries.

6.1 External online services

For the purpose of language processing, several web services are used. All of them are accessed using REST API and the response is in form of a `json` file.

6.1.1 Korektor

Korektor[1] is a statistical spellchecker and (occasional) grammar checker. It is released under 2-Clause BSD license[2] license.

It is used to hopefully improve the results of the syntactic parsing because users' inputs might contain typos and various errors. It outputs a corrected version of uploaded text.

6.1.2 UDPipe

UDPipe[3] is the essential service for this solution. It is a trainable pipeline for tokenization, tagging and dependency parsing. It can be downloaded and trained (Straka and Straková [2016]). I use the UDPipe web service that is publically available under the Mozilla Public License 2.0[4]. There are many linguistic models to choose from. They are distributed under the CC BY-NC-SA[5]. I use the default model which is guaranteed to be the latest Czech model (Straka and Straková [2016]).

UDPipe parses the input text and returns a dependency tree. The output is in CONLL-U[6] format and respects the UD formalisms (section 3.4.5).

6.1.3 MorphoDiTa

MorphoDiTa[7] is a tool for morphological analysis of natural language texts. It is licensed under Mozilla Public License 2.0[8] an the models are distributed under CC BY-NC-SA[9].

In the solution, MorphoDiTa can be used as an alternative lemmatizer instead of the UDPipe.

[1] `http://lindat.mff.cuni.cz/services/korektor/info.php`
[2] `http://opensource.org/licenses/BSD-2-Clause`
[3] `https://lindat.mff.cuni.cz/services/udpipe/info.php`
[4] `http://www.mozilla.org/MPL/2.0/`
[5] `https://creativecommons.org/licenses/by-nc-sa/4.0/`
[6] `https://universaldependencies.org/format.html`
[7] `http://lindat.mff.cuni.cz/services/morphodita/info.php`
[8] `http://www.mozilla.org/MPL/2.0/`
[9] `https://creativecommons.org/licenses/by-nc-sa/4.0/`

6.2 Libraries

The solution uses few Python libraries which can be installed by `pip`.

`numpy`[10] (v. 1.22.3) is a well-known library for numerical calculations.

`requests`[11] (v. 2.26.0) allows to send HTTP requests easily.

`conllu`[12] (v. 4.4.2) parses the CONLL-U files which contain the syntactically parsed text.

`udapi`[13] (v. 0.3.0) is a framework for processin UD data.

`shapely`[14] (v. 2.0.2) is a geometric library that allows to create and manipulate geometric objects.

`matplotlib`[15] is able to make plots.

6.3 Application Design

The high level structure is a pipline of five blocks (see figure 6.1). The Input block loads the input and parses it to a string (`str`). This string is then processed by the Language Processing block which outputs an `NLPObject`. It contains the syntactically parsed text and is used by the Information Extraction block to find information about the room, items, and their position. Based on this information, information Extraction block creates `ItemInstance`s, `Rule`s and an `ItemInstance` of the room. These collections are used as an input for the Composition block which composes them together to form a scene with all `ItemInstance`s placed according to the `Rule`s. The output of this block are `ItemInstance`s, their `Placing`s, and the instance of the room. These objects are then exported by the Output block.

Only Language Processing and Information Extraction are language-dependent. The other do not process the natural language at all so they could be reused for another languages.

6.3.1 Modules

The design is modular on two levels. The high-level structure is a pipeline of five conceptual modules, let me call them **blocks**. Each of the blocks is composed of several low-level modules.

These low-level modules are defined only by their interfaces so that they can be easily replaced by another module with the same interface. This is useful for comparing results generated using different approaches and for improving the implementation of a certain module in the future.

Since Python is a duck-typed language, I simulate the interface semantics using
`NotImplementedError()` exception. Each low-level module has its own directory containing `interface.py` file and various implementations. The `interface.py` file contains a regular class whose name starts with `I` prefix which indicates that the class is used as an interface. None method in the class is implemented, they just raise the `NotImplementedError()` exception. Each implementation of a module is in a separate file.

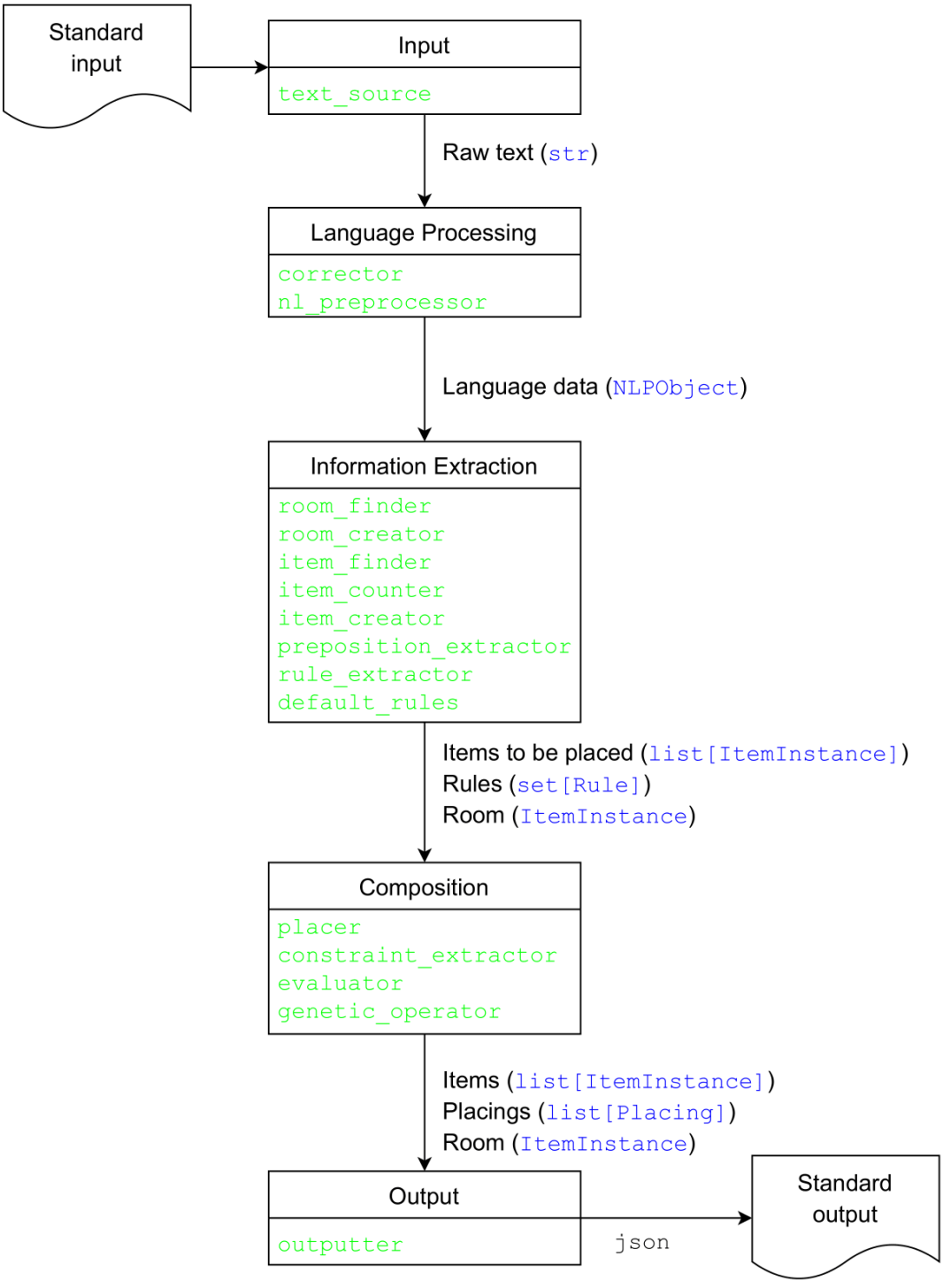


Figure 6.1: High-level blocks in a pipeline

information Extraction Structure

Figure 6.2: Structure of the Information Extraction block

6.3.2 Code Organization

The code is implemented in `src` directory. It is organized to directories by low-level modules (see 6.3.1).

In the `data` directory, `json` configuration files are stored.

The `test` directory contains testing files that were used throughout the work as well as scripts performing certain experiments with libraries etc.

In the `old` directory, no-longer used code is stored.

The `main.py` is the entry-point which is used for executing the scripts.

6.3.3 Input

This block contains a single module `text_source`. This module simply loads the input and parses it to a string (`str`).

6.3.4 Language Processing

This block is composed of two modules: `corrector` which tries to correct the typos and other grammatical and orthographical errors and `nl_preprocessor` which performs syntactic parsing on the input string. The result of this parsing is an instance of `NLPObject` which is a central class that stores linguistic information about the input text in form of both the `conllu` and the `udapi` tree representation. It is also the output of this block.

The modules of this block are implemented using the web services UDPipe and Korektor (see section 6.1). Formerly, even MorphoDiTa was used for another module `lemmatizer` but since the lemmata can be easily extracted from the `NLPObject`, it is no longer needed.

6.3.5 Information Extraction

This block implements the extraction of physical object representation and their positional relations (see section 3). Since it is the focus of this thesis, it has the most complex structure (see). The input of this block is the `NLPObject`. It is used by many modules of this block since it is the central object containing the linguistic information.

The extraction can be devided to three branches. The simplest of the tree extracts information about the room and is composed of `room_finder` which determines the `RoomType` based on the `NLPObject` and `room_creator` which creates an `ItemInstance` of the room.

The second branch extracts items. Firstly, `item_finder` determines the types of the items present in the `NLPObject`. Then, the `item_counter` determines the counts of instances of each object. Finally, the `item_creator` creates the given number of `ItemInstance`s of each `ItemType`.

The third branch is the most complicated. It uses outputs of the modules in the previous branches as well as the central `NLPObject` to extract the `Rule`

from the text. Firstly, the preposition_extractor extracts Prepositions which represent the semantic prepositional relations between items (see 5.1.2). From these Prepositions, Rules are then extracted using rule_extractor. Besides these extracted rules, default_rules module constructs additional rules (see pseudophysic techniques in section 4.2.1).

The output of this block is composed of:

1. list of ItemInstances
2. set of Rules,
3. an ItemInstance of a room

6.3.6 Composition

The Composition block attempts to find Placings for each ItemInstance which respect the Rules. It is done by the placer which is implemented using genetic algorithm (see 5.2).

The other modules of this block are constraint_extractor which extracts Constraints, evaluator which evaluates the individuals of the genetic algorithm (lists of Placings) and genetic_operator which includes SelectionOperator and MutationOperator which are used during the evolution by the placer.

The output of this block consists of:

1. list of ItemInstances
2. list of Placings which respect the order of the ItemInstances,
3. an ItemInstance of a room

6.3.7 Output

This last block contains only one module outputter which outputs the solution for the user. JsonOutputter dumps the solution to a json file. PlottingOutputter uses `matplotlib` to plot the solution. It opens a window with a plot.

7. User Manual

7.1 Requirements and installation

You need to have Python 3.9.4[1] installed on your system. The required packages can be easily installed using `pip`.

```
python3 -m pip install -r requirements.txt
```

7.2 Usage

The usage is pretty simple. It just accepts the data from the standard input and prints the result on the standard output in the `json` format.

```
python3 main.py
```

The input must be a raw Czech text in UTF-8. If it is entered by hand in the terminal, it must be terminated with Ctrl+D on Linux or Ctrl+Z on Windows. Of course, the script can also be pipelined.

```
python3 main.py <input_file
```

7.3 Configuration files

There are `json` configuration files that can be modified. Simply edit those files to add support for new types of items, types of rooms, composed rules or prepositions. The structure of the files must be preserved but it is relatively simple and selfexplanatory.

For example, in `config/item_types.json` the item types are defined. Each type must have the following structure.

```
{
        "name" : "bed",
        "terms" : ["postel", "lůžko", "lože"],
        "geometry" : [[0, 0], [1.6, 0], [1.6, 0.8], [0, 0.8], [0, 0]]
},
```

The `"name"` defines the name of the item type. It must be unique. The `"terms"` are Czech synonyms which can be used for the item type. They must be unique. The `geometry` is a list of vertices of the polygon which defines the shape of the item type. It should be a convex polygon.

[1] Other versions of Python 3 may work as well.

8. Results and Discussion

Let me now presents some results and comment them.

Counts of the items are recognized solidly (see figure 8.1(a)). It does not matter whether the items are reffered by an object or a subject. The only problem with the counts is the coreference (see figure 8.3(b)). If one item is reffered by multiple words, they are not connected and thus, a new instance for each mention is added.

It seems that in terms of extracting prepositions, the solution works solidly. The prepositions are recognized and the rules based on them are respected (see figures 8.1(b)), even the directed rules are respected (see figures 8.2(a), 8.3(a), 8.3(b)).

The quality of the layout depends partially on luck. It was quite common that some objects flied away from the room (see figures 8.2(b), 8.3(a)) but this problem should be fixed in the very last version of the program. The mutation is now restricted such that the position must remains within the area of the room. Still, the rotation is not restricted, so the items could partially penetrate the walls.

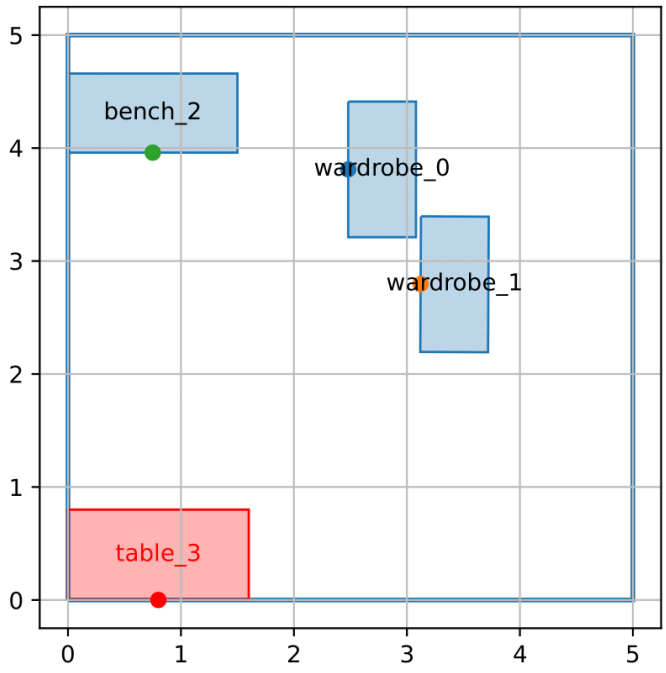
Unsupported nouns are recognized correctly and they are treated as a 1×1 square (see figures 8.3(a), 8.3(b)[1]).

Both distance-based and intersection-based constraints seems to work (see figures 8.1(b), 8.2(b)) but the distance-based are significantly better. The reason probably is that the intersection-based constraints are evaluated as zero almost all the time. For instance, the `within` constraint is non-zero only if its shapes intersects. But they are not lead to any direction so the probability that they meet is not very high.

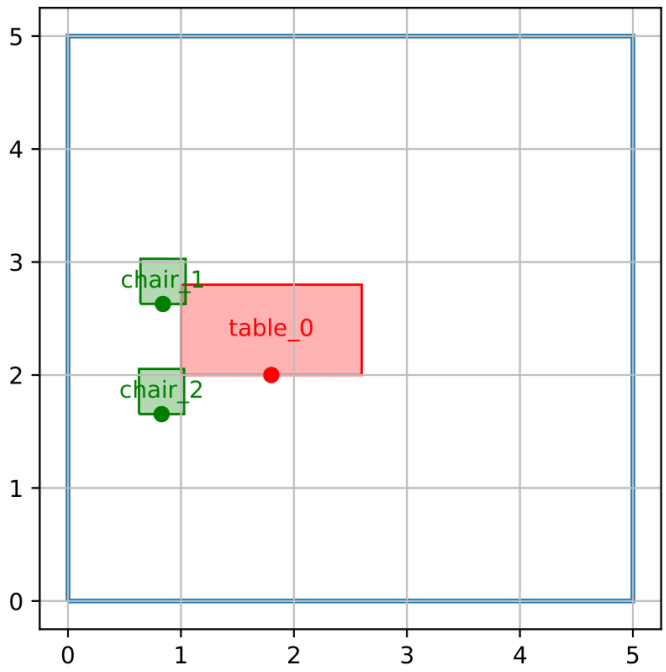
If a phenomenon that is not covered (adverbs, pronouns, etc.) by the solution occurs in the text, the output might respect it only by a coincidence. There are many ways how the information extraction can be improved (see chapter 9).

The performance of the script is acceptable on small inputs. The most time-consuming part of the solution is the genetic algorithm. The time increases with the number of identified items dramatically. For inputs with a larger number of items, the genetic algorithm can easily take several minutes to complete. One of the reason is that for every pair of items some default rules are constructed. So not only the count of items increases but also the count of rules and thus the time of the evaluation which is done in every iteration of the genetic algorithm.

[1] I accidently deleted bed from the types for this test.

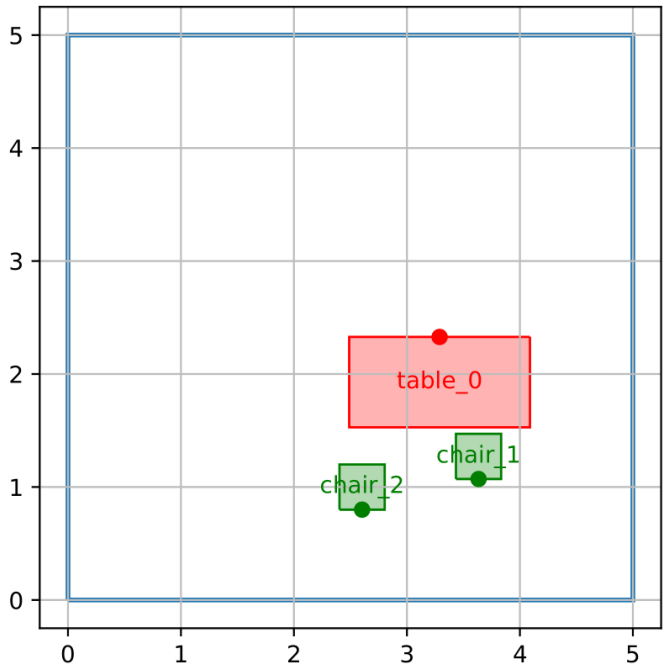


(a) *Vidíte dvě skříně, jednu lavici a jeden stůl.* [*You can see two wardrobes, one bench and one table.*]

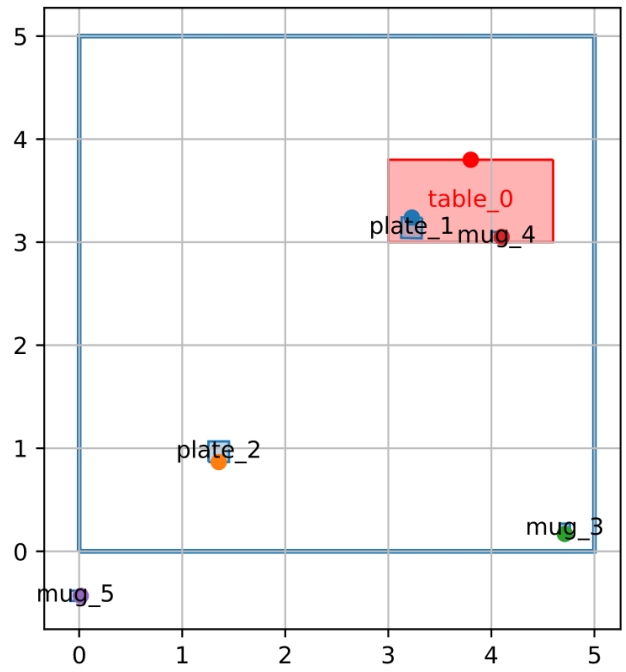


(b) *Vejdete do čtvercové místnosti. Kolem stolu stojí dvě židle.* [*You enter a square room. There is a table and two chairs around it.*]

Figure 8.1: Results

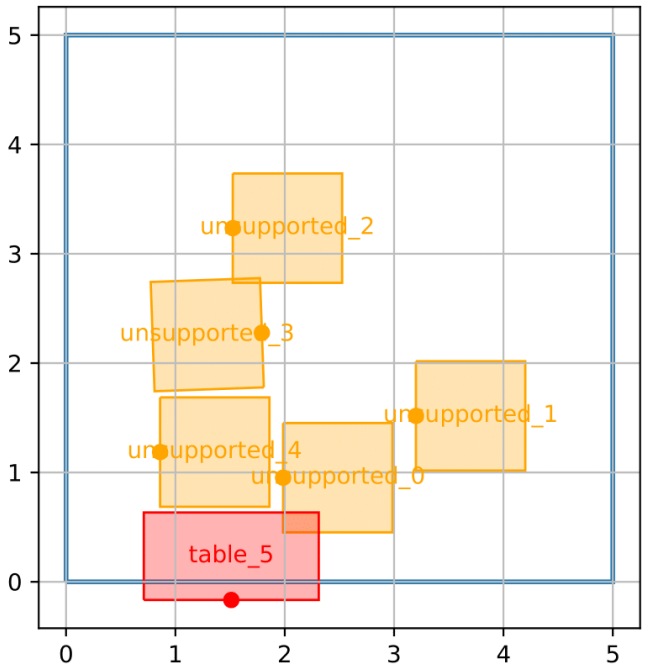


(a) *Za stolem jsou dvě židle.* [*There are two chairs behind the table.*]

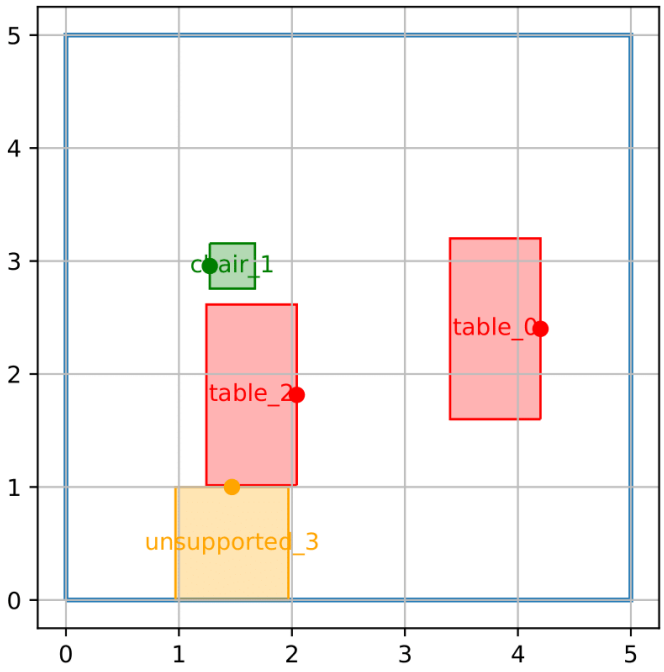


(b) *Na stole leží dva talíře a tři hrnky.* [*There are two plates and three mugs on the table.*]

Figure 8.2: Results



(a) *V místnosti je pět kaktusů za stolem.* [*There are five cacti behind the table.*]



(b) *V místnosti je stůl a židle. Vlevo od stolu je postel.* [*There is a table and a chair/chairs. On the left of the table there is a bed.*]

Figure 8.3: Results

9. Future Works

Since the task is very complex and have many aspects, the solution could be improved endlessly. The application is designed in a modular way with interfaces which means that it can be easily extended in various ways. The main goals of the task was achieved to a great extent (see 8) but there are many opportunities for future works.

9.1 Lexical Support

The easiest way to improve the results without changing the code is to expand the sets of supported item types, room types and prepositions or adding more synonyms for the supported ones. It can be done by editing the `json` configuration files. Prepositions are a relatively closed category, so their coverage is already solid.

Nouns, on the other hand, are a large and open category so it is impossible to cover them completely, because new nouns arise every day. But expanding the set of the supported ones would still improve the results.

9.2 Improving Extraction

Many observed linguistic phenomena could be taken into account in order to improve the results of the information extraction.

9.2.1 Adverbs

Prepositions are not the only part of speech that is able to express spatial relation. There are also adverbs (adv.s). Many secondary prepositions can be also adverbs (*uprostřed* [*in the middle (of)*]). The solution process adverbs only if they are part of multi-word prepositions (see section 5.1.2) otherwise they are ignored. From the semantic point of view, they act like prepositions, only the ground is the surrounding environment. So they could be processed as prepositional relation between a figure and the room.

> ***Uprostřed*** *je stůl.* [*There is a table* ***in the middle****.*] is semantically identical to ***Uprostřed*** *místnosti je stůl.* [*There is a table* ***in the middle*** *of the room.*]

9.2.2 Non-numerical quantifiers

Besides the numerals, there are also determiners in the Universal Dependencies which act as quantifiers (see section 3.3). They could be extracted in the same way as the numerals and modify the choice of the number. Currently, they are ignored and the number is based on the fact that the modified noun is in plural.

mnoho židlí [*many chairs*], *pár židlí* [*few chairs*], *židle (pl)* [*chairs*] are all processed in the same way

9.2.3 Coreference

One of the biggest complication for the extraction is the coreference. During the preposition extraction, pronouns are currently ignored which could be a big flaw since pronouns can in fact occur as arguments of the prepositions. Only nouns are considered. If at least one of the arguments of the preposition is a pronoun, it is not found by the extraction algorithm. So, the extracted preposition either has an incorrectly assigned argument, or is incomplete and discarded.

V místnosti je stůl a před ním židle. [*In the room, there is a table and a chair behind it.*]

Furthermore, the solution is not able to distinguish whether two nouns with the same lemma refer to the same physical object, or not. In general, finding correferences is a difficult task but if the solution were able to deal with them, it would increase the quality of the extraction dramatically.

9.3 Improving Placement Algorithm

The placement algorithm could be improved in two ways – either by improving the genetic operators or changing the hyperparameters, and by adding more artificial rules to be satisfied.

The change of the operators and hyperparameters should lead to higher fitness in the same generation which means that the constraints are more satisfied. Also other strategies like the rule of 1/5 or adaptive σ could be included.

Adding more artificial constraints should make the result look more natural. For example, a plate is expected to be on the table by default, so adding a rule `inside(plate, table)` should help to produce a more natural output.

9.4 Optimizing the implementation

The most time-consuming part of the implementation is the placement algorithm. It could be optimized by using `numpy` arrays instead of Python lists and vectorizing the operators.

The Language Processing block connects to the internet and waits for the response. So, if the connection is slow or the input data too big, it could take some time to obtain the results. The models used by the online service could be downloaded and used locally. This would also reduce the risk of failure of the task.

9.5 Other Potential Extensions

Much more information can be extracted from the text. From the nouns' modifiers the program could deduce properties of the reffered physical object such as color, material, size etc. These properties could then be used to modify the instances of the physical object.

In many cases, the input descriptions specify dimensions of the objects, but the current solution does not allow the shape of the instances to change. It would be nice if the solution were able to adjust the shapes of the instances based on what is mentioned in the input.

Conclusion

In this thesis, I implemented a console application that attempts to generate a structured description of layout of a room described in the natural-language input. The main goal was to extract information about real physical objects and their mutual spatial relations. Based on this retrieved information, geometric constraints are constructed and the objects are placed into the scene of the room so that the constraints are satisfied as much as possible.

I primarily focused on describing linguistic phenomena which are relevant for the task (chapter 3) as well as the application of these ideas in the rule-based implementation of the information extraction (chapter 5). The linguistics analysis brought some surprising results such as the modeling directed prepositions (section 3.4.4) or the fact that prepositions can be reduced, level by level, to a simple language of four geometric constraints (sections 5.1.2, 5.1.3, 5.1.4). Another interesting finding is that the language description levels in linguistics correspond well to the levels of processing the natural language input.

To solve the placement problem, I used a genetic algorithm. The fitness function is based on evaluation of the extracted constraints (section 5.2.1). The constraints are soft and based on an idea of fuzzy logic, so the range of their evaluation is potentially $[0, 1]$. The algorithm tries to maximize the sum of these values. The main advantage of the genetic algorithm is that even if the constraints are not fully satisfied, it still produces a feasible solution (section 4.3).

The results of the solution are acceptable, though much work could be done to improve them (section 4).

Bibliography

Slovník spisovného jazyka českého. Academia, 1960-1971. URL https://ssjc.ujc.cas.cz/.

Mluvnice češtiny 2 – Tvarosloví. Academia, 1986.

Dungeons & Dragons: Lost Mine of Phandelver. Wizards of the Coast LLC, 2014.

Petr Biskup. PŘedloŽka, 2017. URL https://www.czechency.org/slovnik/PÅŸEDLOÅ·KA.

Tomáš Machálek. Kontext – aplikace pro práci s jazykovými korpusy, 2014. URL http://kontext.korpus.cz.

Stéphane Sanchez, O. Roux, Véronique Gaildrat, and Hervé Luga. Constraint-based 3d-object layout using a genetic algorithm. *Intelligenza Artificiale - IA*, 01 2003. URL https://www.researchgate.net/publication/245776169_Constraint-based_3d-object_layout_using_a_genetic_algorithm.

Milan Straka and Jana Straková. UDPipe, 2016. URL http://hdl.handle.net/11234/1-1702. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

F.-A. Ursini. How space gets into language: a novel approach. *ASCS09: proceedings of the 9th Conference of the Australasian Society for Cognitive Science*, pages 348–355, 2010. URL https://www.researchgate.net/publication/269146797_How_space_gets_into_language_a_novel_approach.

Ken Xu, James Stewart, and Eugene Fiume. Constraint-based automatic placement for scene composition. *Proceedings - Graphics Interface*, 06 2002. URL https://www.researchgate.net/publication/2873054_Constraint-based_Automatic_Placement_for_Scene_Composition.

František Čermák. Systém, funkce, forma a sémantika českých předložek. *Slovo a slovesnost*, 57:30–46, 1996. URL http://sas.ujc.cas.cz/archiv.php?art=3658.

List of Figures

List of Tables

List of Abbreviations

D&D Dungeons & Dragons

GM gamemaster

RPG role-playing game

N nominative case

G genitive case

D dative case

A accusative case

V vocative casse

L locative case

I instrumental case

sg singular number

pl plural number

dl dual number

PoS part of speech

n. noun

v. verb

adj. adjective

adv. adverb

prep. preposition

postp. postposition

adp. adposition

num. numeral

pron. pronoun

ML machine learning

NLP natural language processing

MT machine translation

LLM large language models

CSP constraint satisfaction programing

GA genetic algorithm

UD Universal Dependencies

2D two-dimensional

3D three-dimensional

DOF degrees of freedom

v. version

ITIR item type index reference

II item instance

A. Attachments

A `zip` archive is attached to this thesis. It contains the Python code of the solution and an essay on Czech spatial prepositions.

A.1 The Code of the Solution

Besides the currently used scripts, it also contains some files with old implementation to show that multiple approaches were tried during the work. Some were abondened completely and others were reimplemented from scratch.

Also, testing files and scripts with experiments can be found there as well.

A.2 Czech Spatial Prepositions

If the reader is interested in linguistic details on describing Czech prepositions, they can read this attached essay. It briefly sums up approaches to Czech preposition description and focuses on the spatial prepositions.

It describes ethymology, syntax and semantics of the prepositions and covers interesting phenomena such as vocalization or preposition formation.