Computing and Informatics, Vol. 31, 2012, 299–329

PARSING WITH CLAUSE AND INTRACLAUSAL COORDINATION DETECTION

Domen Marinčič, Tomaž Šef, Matjaž Gams

Jozef Stefan Institute
Jamova cesta 39
1000 Ljubljana, Slovenia
e-mail: {domen.marincic, tomaz.sef, matjaz.gams}@ijs.si

Communicated by Peter Vojtáš

Abstract. We present a new dependency parsing algorithm based on the decomposition of large sentences into smaller units such as clauses and intraclausal coordinations. For the identification of these units, new methods combining machine learning techniques and heuristic rules were developed. The algorithm was evaluated on the Slovene dependency treebank text corpus. Compared to the MSTP parser, currently the most accurate for Slovene, parsing accuracy was improved by 1.27 percentage points, which equals 6.4 % relative error reduction.

Keywords: Clause identification, intraclausal coordination detection, dependency parsing, artificial intelligence

1 INTRODUCTION

Syntactic parsing represents one of the possible intermediate steps of text analysis in the applications such as machine translation, information extraction from resources like World Wide Web, question answering, etc. The result of syntactic parsing are syntactic trees that demonstrate the structure of a sentence. They are the basis for the next step, the semantic analysis, which discovers the meaning of the text.

In the last decades, dependency formalisms [22] became popular for the description of syntactic structure. They use dependency trees to describe the relations among the constituents of the sentence in a human and computer readable form. An example of a dependency tree describing a sentence in Slovene is presented in Figure 1. In the dependency tree, each token (a word or a punctuation mark) is

represented by a node. Observing the nodes of the tree from left to right, they appear in the same order as the corresponding tokens in the sentence. An additional technical node is added as the root of the tree to ensure that all nodes are connected into a single tree. The edges connecting the nodes describe the relations between the tokens. The labels below the tokens indicate the functional role of the relation between the node and its parent such as *subject* ('Sb'), *object* ('Obj'), *predicate* ('Pred'), *auxiliary verb* ('AuxV'), etc.

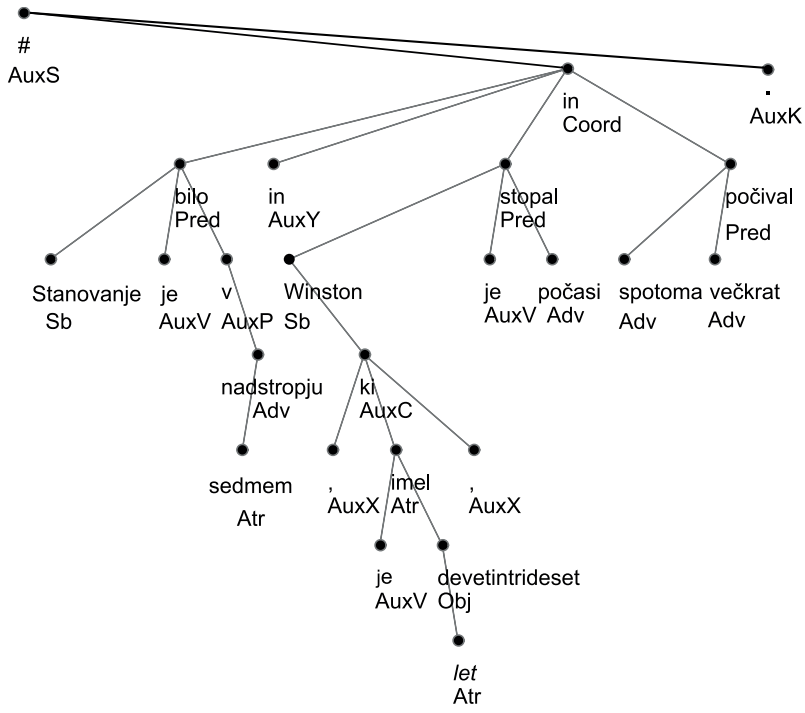


Fig. 1. An example dependency tree for a sentence in Slovene. The English word-by-word gloss is given below Slovene text.

In the following three subsections we present the motivation for the use of accurate syntax analysis and its possible benefits. An example of the positive effect of syntactic analysis in an application of machine translation is presented in Subsection 1.1. The advantages of the use of dependency-based formalisms over the constituency-based formalisms are discussed in Subsection 1.2. Finally, the role of the intraclausal coordination and clause detection at parsing is presented in Subsection 1.3.

1.1 A Motivating Example

Here, an example problem shows the benefits of the syntactic analysis for machine translation. The general ideas of two widely used approaches for solving the problem are presented:

1. the statistical/example-based approach and
2. the approach with the syntax analysis.

The example focuses on the use of passive and active voice. While in English passive voice is frequently used, in Slovene active voice is preferred[1]. The example demonstrates, how the consistent use of the active voice in Slovene translations can be achieved by the use of syntax analysis.

The approach with syntax analysis is depicted in Figure 2. First, the dependency tree of the English sentence is produced (Figure 2 a)). By matching it to the source tree template of the transfer rule (Figure 2 b)), the passive construction in the English sentence is discovered. The matched entities in Figure 2 are set in bold. The transfer rule converts the English passive construction to the Slovene active construction. Finally, the generic entities of the active construction are replaced with the translations of the English words to generate the Slovene active voice translation in Figure 2 c).

At the statistical/example based English-to-Slovene translation approach, we rely upon a parallel sentence-aligned English-Slovene corpus. This is a collection of the same text written in English and Slovene, where the aligned sentences are the translations of each other. If a certain phrase in Slovene and a certain phrase in English appear in aligned sentences very often, it is very likely that these two phrases are translations of each other. Here are some examples of aligned sentences, where the translation (italicized) of the phrase "the airplane wings" can be found:

- "*The airplane wings* provide aerodynamic lift." $\longleftrightarrow$ "*Letalska krila* zagotavljajo dinamični vzgon."
- "The engines are attached to *the airplane wings*." $\longleftrightarrow$ "Motorji so pritrjeni na *letalska krila*."
- "*The airplane wings* are delta-shaped" $\longleftrightarrow$ "*Letalska krila* imajo delta obliko."

The translations of the phrases "were broken" $\longleftrightarrow$ "so bila polomljena" and "by the heavy wind" $\longleftrightarrow$ "z močnim vetrom" can be found similarly. By combining the translations of the phrases, the sentence translation is formed:

"The airplane wings were broken by heavy winds." $\longrightarrow$ "Letalsko krilo je bilo polomljeno z močnim vetrom."

[1] An example of an English sentence in passive voice: "The airplane wings were broken by heavy winds." and in active voice: "Heavy winds broke the airplane wings."

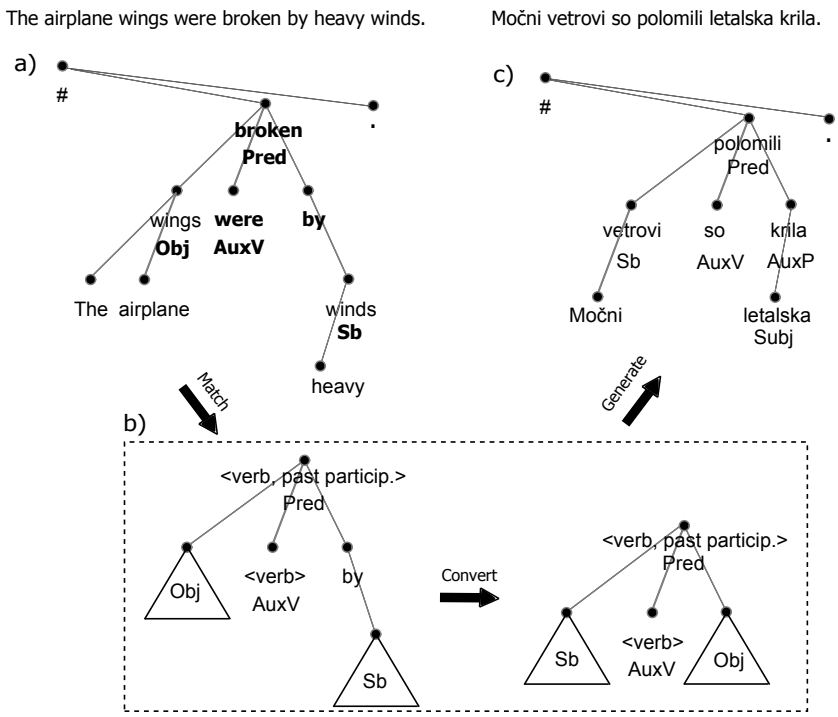


Fig. 2. Translation using syntax analysis

Contrary to the translation produced by the approach using syntax analysis, the sentence in Slovene remains in the passive voice – the statistical/example based approach offers no mechanisms to directly enforce the use of the active voice in the Slovene translations.

As demonstrated by this example, the use of syntax analysis raises the quality of machine translation by allowing for better control of the translation process. Obviously, this is only possible with accurate algorithms for automatic syntactic analysis, which are the object of the study of the work described in this paper.

1.2 Why Dependency Trees?

Two types of formalisms, the dependency-based and the constituency-based, are mainly used for the syntactic structure representation. We advocate for the use of the first one, and show its advantages over the constituency-based formalisms.

The tree in Figure 3 a) presents how the English sentence from Section 1.1 could be described with a constituency-based formalism. For the convenience, we show the dependency tree of the sentence again (Figure 3 b)). In the constituency tree, the words are represented by the leaves of the tree, while the internal nodes represent

the constituent phrases of the sentence. They are denoted by the labels 'NP' – noun phrase, 'VP' – verb phrase, 'DET' – determiner, 'S' – sentence, etc. Punctuation tokens are commonly not included in the constituency trees.

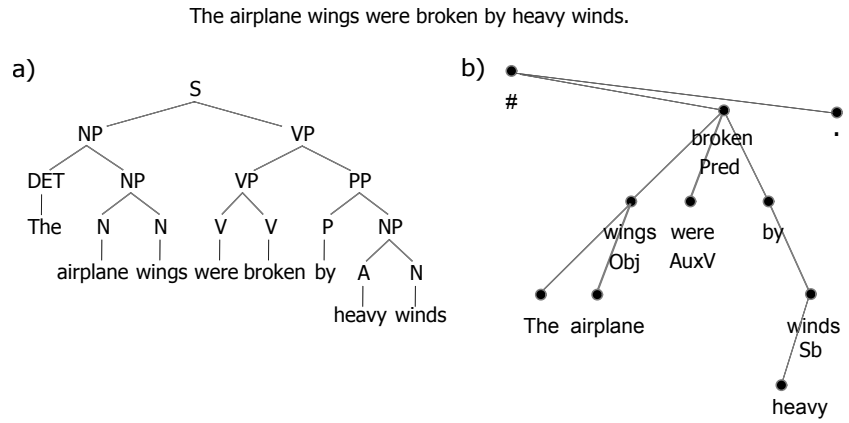


Fig. 3. The comparison of a) constituency- and b) dependency-based representations of the English sentence from Section 1.1

Some arguments are presented, why one might prefer a representation that is based on the notion of dependency [4, 16, 13, 3]:

- In a dependency tree, the head of a phrase can be directly found as it is the root of the phrase subtree, which is useful if semantic analysis is built on top of the syntactic representation. Take the subtree of the "The airplane wings" phrase as an example. The word "wings" is the phrase head, which is clearly indicated by the dependency tree. On the contrary, the constituency tree gives no clear information about the phrase head.

- Dependency trees contain no additional nodes but the ones representing the sentence tokens. Because the parser's job is only to connect existing nodes, the task may be regarded as less complicated.

- Dependency formalisms enable better treatment of languages with free word-order, e.g. Slovene. In such languages, phrases may be constituted of discontinuous sequences of words. With the constituency based formalisms only continuous blocks of words can be connected to a phrase. On the other hand, the dependency trees allow for connecting of arbitrary sequences of words into a subtree – the edges may cross each other – and are thus better suited for the representation of discontinuous phrases.

1.3 Parsing, Clause and Intraclausal Coordination Detection

Here a quick overview of parsing and detection of intraclausal coordinations and clauses is presented. The emphasis is on the role of intraclausal coordination and clause detection in terms of decomposing the complex problem of the parsing of a sentence into smaller, simpler tasks.

The parsing algorithms infer syntactic trees from text. There are two possible approaches, the grammar-driven and the data-driven approach. In the first approach, the text is described by a formal grammar, which guides the parsing process. In the second case, the parser is trained directly on a corpus containing sentences manually annotated with dependency trees. The primary goal of parsers is to achieve the best possible accuracy of parsing. Generally, the manually annotated corpus serves as a *gold standard* and is separated into two parts: the train set and the test set. To estimate the accuracy of parsing, the test set sentences are parsed and the output is then compared to the manually created trees. The most common accuracy evaluation metric is the number of correctly assigned dependencies, i.e. the edges between the nodes, divided by the total number of dependencies.

Tokens inside a sentence are structured into subunits such as clauses and intraclausal coordinations. These subunits are represented as subtrees of the dependency tree. The parsing complexity grows with the number of such subunits in the sentence. While the most successful dependency parsers [12, 16, 2, 15] operate on the level of a sentence, it makes sense to upgrade the parsing algorithms by identifying these closed subunits. The parsing problem is thus simplified, because each subunit can be parsed separately.

Clause identification is a well known problem, too. In [23, 20, 19] clause identification is modeled as clause border recognition. Several types of machine learning algorithms have been used to facilitate finding the clause borders. [7] presents an algorithm for retrieving nominal coordinations, which relies on semantic features of heads of noun conjuncts.

So far, clause identification and parsing have mostly been separately explored problems – with some exceptions. [1] describes an algorithm for parsing English, where a clause filter is used to recognize clauses prior to parsing. The algorithm is limited to finding simple non-nested clauses. In [18], a system for incremental parsing of Japanese spoken monologue is presented, where text is parsed on a clause-by-clause basis. A rule-based parser for Czech is presented in [8] with a short description how clause identification is included in the parsing process. A description of intraclausal coordination detection combined with parsing can be found in [11]. [14] deals with coordinations at parsing, including coordinations of clauses, by applying transformations of annotation schemes of coordinations as subtrees in dependency trees.

We propose a new algorithm for PArsing with Clause and Intraclausal coordination Detection – PACID. Our algorithm builds on the synergy of new methods for clause and intraclausal coordination detection joined with standard parsing algorithms. For training and evaluating the algorithm, the Slovene Dependency Tree-

bank (SDT) [5] was used, a corpus of Slovene text annotated with dependency trees containing 38 646 tokens.

2 LINGUISTIC BACKGROUND

The first issue to be resolved is the definition of the clause in Slovene language. In [24], one of the fundamental grammar books of modern Slovene, we can find the following theoretical definition: "The clause is a text unit, whose core is a complex verb form. It can contain optional additional constituents, such as subject, object, adjunct etc., which describe the verb form more exactly." Unfortunately, this definition is not exact enough for formal tasks. To ease the design of our algorithms and their evaluation, we adopted SDT as a standard for the definition of the clause, while trying to find such rules that match the grammar book definition as exactly as possible. As for intraclausal coordinations, to our knowledge there exist no widely accepted formal definition; SDT was used to define them as well.

The structure of dependency trees provides most of the information needed to construct the definitions, but some supplementary information contained in SDT is still needed. Each token is annotated with a MSD-tag (Morphosyntactic description tag), i.e. a sequence of characters, which describe the features such as category, number, person, case, gender etc. MSD-tags are formatted according to the Multext-East standard [6], the character at each position describes one feature. Further, the lemma is provided for each token. For example, the full MSD-tag of the word 'bilo' (Eng.: 'was') is 'Vcps-sna', while its lemma is 'biti' (Eng.: 'be'). The MSD-tags and the lemmas were determined by an automatic tagger and lemmatizer. The tags and lemmas were subsequently manually checked and corrected. However, still some errors in the annotations exist. In the following text, the tokens are denoted with small Latin letters, while the nodes in the trees are denoted with small Greek letters. Additionally, the SDT corpus provides each node with a label, which describes the functional role of the relation between the node and its parent.

In Table 1, we explain the annotations of the most important types of the words, which are used in the definitions of the algorihms and in the examples throughout the article.

Token type	MSD-tag
Coordinating conjunction	pos.1 = 'C' and pos.2 = 'c'
Subordinating conjunction	pos.1 = 'C' and pos.2 = 's'
Finite verb	pos.1 = 'V' and pos.3 $\neq$ 'n'
Relative pronoun	pos.1 = 'R'
	parent – node edge label
Auxiliary verb	label = 'AuxV'

Table 1. Token types, pos.x denotes the x$^{\text{th}}$ position in a MSD-tag

2.1 Clause Definitions

A clause is represented by a subtree of the sentence tree. According to the syntactic properties of the clause, they are determined a type, which provides additional information to the parsing process. Three types have been identified in the SDT corpus:

- coordinate clauses,
- type 1 subordinate clauses (starting with a subordinating conjunction),
- type 2 subordinate clauses (not starting with a subordinating conjunction).

The definition of a subordinate clause subtree consists of two parts: first, its root is defined and second, the pruning principle is described. For coordinate clauses, a third part is added where further nodes are selected beside the pruned subtree. Before we begin with the definition of the roots of clauses, the root of a coordination of clauses is defined.

Definition 1. The node ν with the parent edge label l is a *clause coordination root* if $l =$ 'Coord' and at least one of the children is a non-auxiliary finite verb or another clause coordination root.

Please note that the definition of the clause coordination root is recursive. We proceed with the definition of the roots of the subtrees representing the three types of clauses.

Definition 2. The node ν representing the token t is a *coordinate clause root* if the token t is a non-auxiliary finite verb and the node's parent is a clause coordination root or the technical root of the sentence.

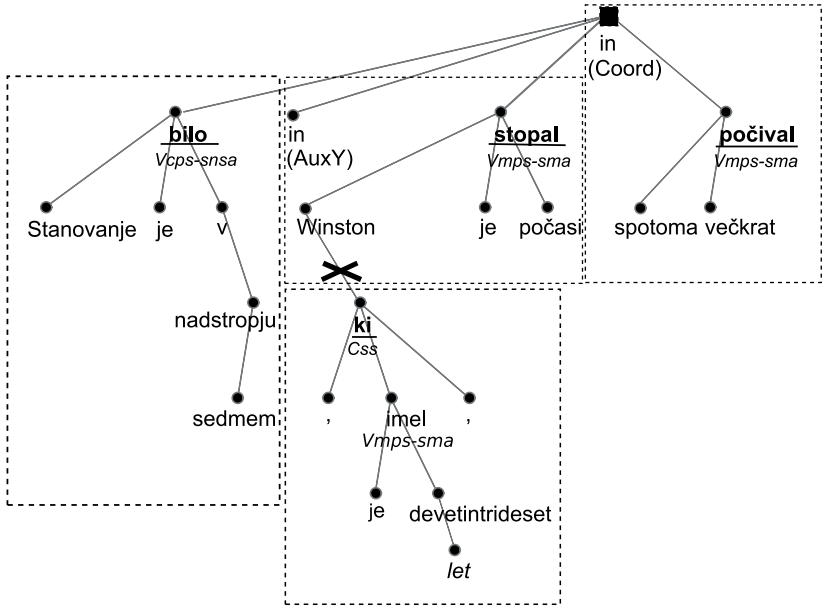
Please note that the clause coordination root and the coordinate clause root are different terms.

Definition 3. The node ν representing the token t is a *type 1 subordinate clause root* if the token t is a subordinating conjunction and one of its children is a non-auxiliary finite verb or a clause coordination root.

Definition 4. The node ν representing the token t is a *type 2 subordinate clause root* if the token t is a non-auxiliary finite verb, having neither a subordinating conjunction nor a clause coordination root nor a word with lemma 'biti' (Eng.: 'to be') nor the technical root of tree as its parent.

Figure 4 presents a dependency tree decomposed into clauses. The words of the clause roots are underlined. The clause coordination root is marked with a square node. The nodes of the words 'bilo', 'stopal' and 'počival' are coordinate clause roots. The node of the word 'ki' is a type 1 subordinate clause root.

We continue with the second part of the definition, i.e. pruning. To eliminate the embedded structures, a subtree is pruned at all subordinate clause roots and



Stanovanje je bilo v sedmem nadstropju in Winston , ki je imel
The-flat was on the-seventh floor and Winston, who had
devetintrideset let , je stopal počasi in spotoma večkrat počival.
thirty-nine years , was walking slowly and meanwhile many-times was-resting.

Fig. 4. A part of the tree from Figure 1 showing the clauses structure is presented. The clauses are enclosed in dashed rectangles. For clarity of the presentation, only the labels (in brackets) and the MSD-tags (italicized) of the tokens, which are related to the clause definition rules, are shown.

clause coordination roots. For the subtree having the node 'stopal' as its root, the pruning proceeds at the node 'ki', which is a subordinate clause root. With pruning, the subordinate clauses of both types are well defined (the only subordinate clause in Figure 4 actually does need to be pruned, because it contains no embedded structures).

In case of a coordinate clause subtree, further nodes not attached to the subtree may be selected after pruning to form a clause. The pruned subtree of the node 'stopal' does not constitute the whole clause: the node 'in' with the label 'AuxY' is missing. We continue the definition relatively to the clause coordination root which is the parent of the coordinate clause roots $\beta_1, \beta_2, \ldots, \beta_n$ (non-auxiliary finite verbs, 'bilo', 'stopal' and 'počival'). There are other children of the clause coordination root as well, such as the node of the word 'in' labeled with 'AuxY' in Figure 4. The rightmost clause of the coordination consists of the following subtrees:

- the subtree of the node β_n (in Figure 4, the subtree of the node 'počival'),
- the subtrees of the other children residing between the nodes β_{n-1} and β_n (not present in Figure 4),
- the subtrees of the other children residing right of the node β_n (not present in Figure 4) and
- the clause coordination root.

Other clauses of the coordination consist of the following subtrees:

- the subtree of the node β_i (for example, the subtree of the node 'stopal' in Figure 4) and
- the subtrees of the other children residing between the nodes β_{i-1} and β_i (in Figure 4, the subtree – actually a single node – of the node 'in' labeled 'AuxY').

2.2 Definition of Intraclausal Coordinations

Intraclausal coordinations are represented as subtrees of the sentence tree as well. We define the intraclausal coordination subtree in terms of its root.

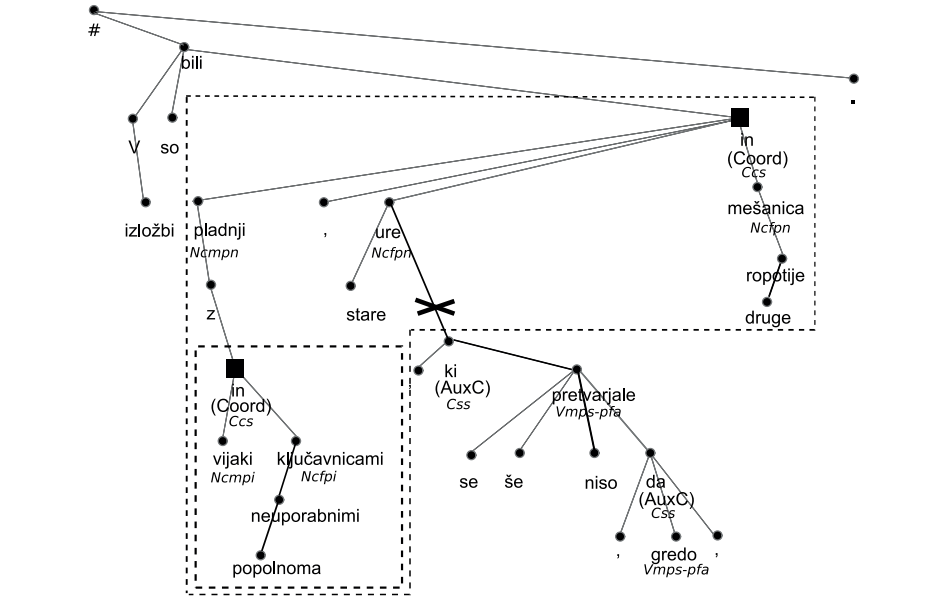
Definition 5. The node ν with the parent edge label l is an *intraclausal coordination* root if $l =$ 'Coord' and none of the children of the node is a finite verb.

Pruning the subtree determined by the root proceeds similarly as for clauses: the subtree is pruned at all clause coordination roots and subordinate clause roots. The tokens of the nodes that remain in the subtree after pruning constitute the intraclausal coordination.

The dependency tree in Figure 5 contains two intraclausal coordination subtrees. Their roots are represented with square nodes. The smaller subtree is embedded inside the larger one. The point where the larger subtree is pruned is marked with a cross. The groups of the nodes of the intraclausal coordinations are delimited by the dashed lines.

The tree corresponds to the sentence 'V izložbi so bili pladnji z vijaki in popolnoma neuporabnimi ključavnicami, stare ure, ki se še pretvarjale niso, da gredo, in mešanica druge ropotije.' The English translation of the sentence: 'In the window there were trays of nuts and barely useful bolts, tarnished watches that did not even pretend to be in going order, and other miscellaneous rubbish.' The word-by-word gloss in English is provided in the figure below the text in Slovene.

An intraclausal coordination consists of several conjuncts, governed by the head words. The head-word group of an intraclausal coordination can be regarded as a skeleton of the coordination. In the inner coordination in Figure 5, the conjuncts are represented by both subtrees of the word 'in'. The words 'vijaki' (Eng.: 'screws') and 'ključavnicami' (Eng.: 'locks') are the conjuncts' head words.



V	izložbi	so	bili	pladnji	z	vijaki	in	popolnoma	neuporabnimi	ključavnicami	,	stare
Spsl	*Ncfsl*	*Vcip3p--n*	*Vmps-pma*	*Ncmpn*	*Spsi*	*Ncmpi*	*Ccs*	*Rgp*	*Afpfpi*	*Ncfpi*	*Z*	*Afpnpn*
In	the-window	have	been	trays	with	bolts	and	completely	useless	nuts	,	old

ure	,	ki	se	še	pretvarjale	niso	,	da	gredo	,	in	mešanica	druge	ropotije	.
Ncnpn	*Z*	*Css*	*Px------y*	*Q*	*Vmps-pfa*	*Vcip3p--y*	*Z*	*Css*	*Vmip3p--n*	*Z*	*Ccs*	*Ncfsn*	*Pi-fsg----a*	*Ncfsg*	*Z*
watches	,	which	itself	even	pretend	did-not	,	that	they-work	,	and	a-mix	of-other	rubbish	.

Fig. 5. There are two intraclausal coordinations subtrees, delimited by dashed lines, one embedded inside another. Only the labels (in brackets) and the MSD-tags (italicized) needed for the definition of intraclausal coordinations are shown.

3 THE ALGORITHM FOR PARSING WITH CLAUSE AND INTRACLAUSAL COORDINATION DETECTION

In this section, the new algorithm *PA*rsing with *C*lause and *I*ntraclausal coordination *D*etection – PACID is described. The algorithm embodies a set of heuristic rules and five machine learning (ML) classifiers. As the base dependency parsers, a newly developed rule-based parser and the standard MSTP parser [12], version 0.2 in the non-projective mode with the parser's original feature pool are used. The algorithm is composed of two stages:

1. detection and reduction of clauses and intraclausal coordinations,
2. dependency tree construction.

3.1 The First Stage – Detection and Reduction Stage

This stage is an iteration, where the sentence is decomposed into clauses and intraclausal coordinations. The iteration consists of the following steps:

1. Split the sentence into the segments delimited by the punctuation tokens and conjunctions.
2. Detect intraclausal coordinations and reduce them to the meta nodes.
3. Split the sentence into the segments again.
4. Detect clauses and reduce them to the meta nodes.

The algorithm iterates until no more units can be detected.

3.1.1 Segmentation of the Sentence

This section describes the first and the third step of the first stage. The sentence is split into the segments as proposed by [9]. Let the sentence be the following sequence of tokens:

$$(s_{1,1}, \ldots, s_{1,k_1}, d_{1,1}, \ldots, d_{1,l_1}, \ldots, s_{i,j}, \ldots, d_{i,j}, \ldots, s_{n,1}, \ldots, s_{n,k_n}, d_{n,1}, \ldots, d_{n,l_n})$$

where the tokens $d_{i,j}$ are the punctuation marks or conjunctions and the tokens $s_{i,j}$ are other words. The sequences $(s_{i,1}, \ldots, s_{i,k_i})$ are the segments, while the sequences $(d_{i,1}, \ldots, d_{i,l_i})$ are the delimiters. The segments containing at least one finite verb are verb segments; the others are non-verb segments.

Definition 6. The segmentation of a sentence is the sequence of delimiters and segments $(\mathcal{S}_1, \mathcal{D}_1, \ldots, \mathcal{S}_n, \mathcal{D}_n)$, where $\mathcal{S}_i = (s_{i,1}, \ldots, s_{i,k_i})$ and $\mathcal{D}_i = (d_{i,1}, \ldots, d_{i,l_i})$.

In Figure 6, segmentation of the sentence from Figure 5 is shown.

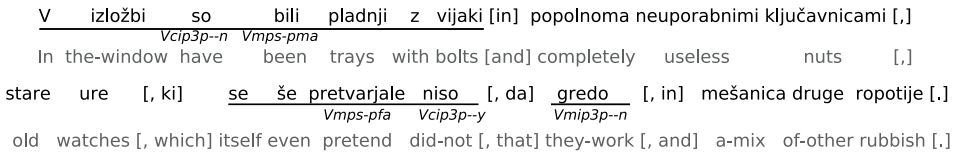


Fig. 6. Segmentation of a sentence. The verb segments are underlined.

3.1.2 The Algorithm for Intraclausal Coordination Detection and Reduction

This section describes the second step of the first stage. The algorithm is applicable to prepositional, nominal and adjectival intraclausal coordinations. For each of these three categories the following steps are performed:

1. Detect head-word group candidates.
2. Filter the candidates.
3. Reduce the detected intraclausal coordinations to the meta nodes.

The candidate head-word groups are detected as follows. Let $(c_1, t_{1,1}, \ldots, t_{1,n_1}, c_2, \ldots, c_i, \ldots, t_{i,j}, \ldots, c_n)$ be an arbitrary uninterrupted sequence of tokens inside the sentence. The tokens $(c_1, \ldots, c_n)$ represent a head-word group candidate if the following heuristic rule holds:

Definition 7. The heuristic rule A holds *iff* all the tokens c_i, $1 \leq i \leq n$ have the same case and the category.

For prepositions, the case of the dependent noun is taken. This covers the structure of the majority of the intraclausal coordinations. Unfortunately, the rule does not cover certain cases like the coordinations of the words of different categories, as for example in this case: "pod stolom ali tukaj" (Eng.: "under the chair or here"). The rule would not identify that the preposition "pod" (Eng. "under") and the adverb "tukaj" (Eng. "here") are coordinated.

Then, the candidates are filtered. First, they are converted to the pairs of neighboring head words (c_i, c_{i+1}), $1 \leq i < n$. To each pair, additional heuristic rules are applied in the same order as they are presented. If any of the heuristic rules does not hold, the candidate group the pair belongs to is discarded.

Definition 8. The heuristic rule B holds *iff* none of the tokens $t_{i,j}$ is a colon, semicolon, dash, bracket, finite verb, relative pronoun or subordinating conjunction.

This rule is motivated by the fact that by definition an intraclausal coordination cannot be split between two or more clauses. If one of the tokens specified in the rule appears between the words c_1 and c_n, it is very likely that the group of words $c_1, \ldots, c_n$ does not belong to an intraclausal coordination, because the token may indicate the beginning of a new clause.

Definition 9. The heuristic rule C holds *iff* for each i exactly one of the tokens $t_{i,j}$ is either a comma or a coordinating conjunction.

Among the tokens between two of the head words there always has to be exactly one comma or coordinating conjunction. The reason for introducing this rule is that the parts of an intraclausal coordination, i.e. the conjuncts, are in most of the cases delimited by a separator which is either a comma or coordinating conjunctions.

Let $t_{i,l}$ be the separator between the conjuncts containing the head words c_i and c_{i+1}. Since $t_{i,l}$ is a comma or a coordinating conjunction it is also a delimiter between two segments.

Definition 10. The heuristic rule D holds *iff* for each $t_{i,l}$ not both of the neighboring segments are verb segments.

The rule D takes into account that two neighboring verb segments very likely reside in two separate clauses, meaning that it is highly unlikely that an intraclausal coordination would spread over two such segments.

In Figure 7, there is one candidate group of two head words ('vijaki', 'ključavnicami'). There are, though, three other words ('pladnji', 'ure', 'mešanica') that meet the conditions of rule A by having the same category and case. However, they are not identified as a candidate group; among them, there are several unallowed conjunctions, commas and finite verbs and the three words fail to comply with the heuristic rules B, C and D.

V	izložbi	so	bili	pladnji	z	**vijaki**	[in]	popolnoma	neuporabnimi	**ključavnicami**	[,]
				Ncmpn		*Ncmpi*	*Ccs*	*Rgp*	*Afpfpi*	*Ncfpi*	
In	the-window	have	been	trays	with	**bolts**	[and]	completely	useless	**nuts**	[,]

stare	ure	[, ki]	se	še	pretvarjale	niso	[, da]	gredo	[, in]	mešanica	druge	ropotije.
	Ncfpn	*Css*	*Vmps-pfa*		*Vcip3p--y*		*Css*	*Vmps-pfa*	*Ccs*	*Ncfpn*		
old	watches	[, which]	itself	even	pretend	did-not	[, that]	they-work	[, and]	a-mix	of-other	rubbish.

Fig. 7. In the sentence, there is a candidate group of two head words (in boldface). The three words underlined with dotted lines match the conditions of rule A, but fail to comply with rules B, C and D. MSD-tags (italicized) are shown below the tokens.

The candidate groups are further filtered by ML classifiers. Each pair (c_i, c_{i+1}) is classified; if at least one pair is classified negatively, the group that the pair belongs to is discarded.

Three separate classifiers were built, one for each category of the head words PACID can handle. The software package WEKA [25] was used for the implementation of the classifiers. The AdaBoostM1 algorithm [21] was used with the J48 decision trees (a reimplementation of the C4.5 algorithm [17]) as the base classifiers. The examples for training the classifiers were extracted from the SDT corpus. To describe the examples with the attributes, the information is extracted from the tokens between the head words. Two sections of the tokens are formed: the section A consists of the tokens between the first head word and the delimiter, while the section B consists of the tokens between the delimiter and the second head word. Every attribute in the list below apart from the class attribute is present two times, once for each section:

- A preposition in the section; binary values. Motivation for the use of the attribute: in a nominal or adverbial coordination, a preposition would very rarely be encountered.
- An adverb in the section; binary values. Motivation: adverbs may be more frequently related to verbs than to prepositions, nouns or adjectives.
- An adjective matching/non-matching with the head word in case, number and gender in the section; two attributes, binary values, only the case is considered when matched to a preposition. Motivation: a matching adjective might appear in a nominal coordination, but not a non-matching one.

- A noun matching/non-matching with the head word in case, number and gender in the section; two attributes, binary values, only the case is considered when matched to a preposition. Motivation: a matching noun almost always appears in a prepositional coordination while a non-matching noun is less likely to be in such a coordination.
- The number of words in the section; values: 0, 1, 2, or >2. Motivation: very long sections A and B are not usual.
- class, binary values.

If all the pairs of the group are classified positively, the algorithm continues with the third step. The sequence of tokens starting with the leftmost head word and ending with the rightmost head word is created. A sequence is replaced by a meta node, which is assigned a MSD-tag containing the same category and case as the head words of the intraclausal coordination. In Figure 8, the reduction of an intraclausal coordination is depicted. In Table 2, the attribute-value description of the head word pair from Figure 8 is presented.

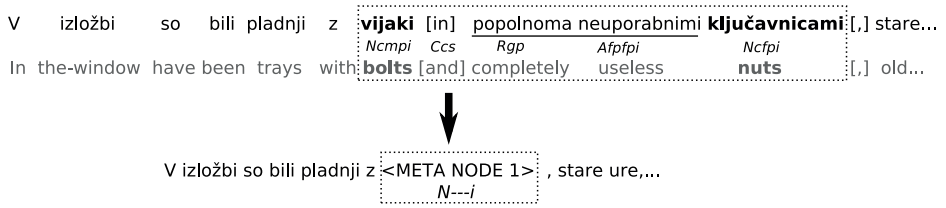


Fig. 8. Reduction of an intraclausal coordination is presented. The text in the example is a part of the sentence from Figure 5. The head words are set in boldface and the section B is underlined. The section A is empty in this example. The MSD-tags are shown below the words.

3.1.3 The Algorithm for Clause Detection and Reduction

This section describes the last step of the first stage. A clause is composed of one or more segments including the preceding delimiters. In Figure 9, the first, the second, the third and the last segment, together with the delimiters preceding them, constitute one clause. The fourth and the fifth segment with the delimiters that precede each of them represent two one-segment clauses. The algorithm for clause detection iteratively identifies and reduces one-segment clauses. After the reduction of a clause, the segments of another clause may join to form a single verb segment and in the next iteration other, originally multi-segment clauses are reduced.

The clause detection step is performed only if there are more than one verb segments in the sentence. Let $(\mathcal{S}_1, \mathcal{D}_1, \ldots, \mathcal{S}_n, \mathcal{D}_n)$ be the segmentation of the sentence. First, the following heuristic rule is applied to each of the verb segments $\mathcal{S}_i$:

[empty]		‘popolnoma neuporabnimi’	
Attribute	Value	Attribute	Value
Adverb, sec. A	0	Adverb, sec. B	1
Preposition, sec. A	0	Preposition, sec. B	0
Matching noun, sec. A	0	Matching noun, sec. B	0
Non-matching noun, sec. A	0	Non-matching noun, sec. B	0
Matching adject., sec. A	0	Matching adject., sec. B	1
Non-matching adject., sec. A	0	Non-matching adject., sec. B	0
Section size, sec. A	0	Section size, sec. B	2

Attribute	Value
Class	1

Table 2. Attribute-value description of a head word pair constituting the intraclausal coordination from Figure 8

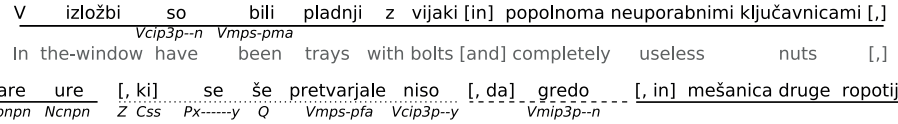


Fig. 9. There are three clauses in the sentence. Two clauses (underlined with dashed and dotted lines) are embedded in the third one (underlined with solid lines).

Definition 11. The heuristic rule E holds *iff* each of the segments $\mathcal{S}_{i-2}, \mathcal{S}_{i-1}, \mathcal{S}_{i+1}$, $\mathcal{S}_{i+2}$ is either a verb segment or a non-existent segment.

If the two segments preceding S_i and the two segments succeeding S_i are verb segments or some of these segments are missing, the rule E holds and the segment $\mathcal{S}_i$ is identified as a one-segment clause. The motivation for this rule is that a clause rarely consists of segments being very far away. If the segment $\mathcal{S}_i$ were not a whole clause, it would most probably imply that other segments of the clause are at least three positions away, which does not happen very frequently.

If the rule E does not hold, the verb segment is classified by a ML classifier. Two classifiers (both the AdaBoostM1 algorithm with J48 decision trees as base classifiers) are used: the first one when both neighboring segments are verb segments and the second one when at least one neighbor is a non-verb segment. To describe the segment in the attribute model the delimiter preceding it, two segments left and two right with the preceding delimiters are included. For each delimiter/segment pair the following attribute set is used:

- The presence of a coordinating conjunction; binary values. Motivation for the use of the attribute: a coordinating conjunction in the delimiter before a verb segment may indicate the beginning of a new clause.

- The presence of a subordinating conjunction; binary values. Motivation: a subordinating conjunction in the delimiter before a verb segment practically always indicates the beginning of a new type 1 subordinate clause.
- The presence of a punctuation token; values: 'none', 'comma', 'colon_or_semicolon', 'other'. Motivation: a punctuation token in the delimiter before a verb segment more (colon, semicolon) or less (comma) strongly indicates the beginning of a new clause.
- The presence of a relative pronoun; binary values. Motivation: a relative pronoun in the delimiter before a verb segment sometimes indicates the beginning of a new clause.
- The auxiliary verb appears before the participle; values: 'yes', 'no', 'not_def'. Motivation: in Slovene, this word order sometimes indicates if a clause is embedded in another clause.
- The possible existence of a crossing intraclausal coordination, e.g. one head word lies in the described segment and the others lie in the neighboring segments. To locate such head word groups, the relaxed versions of the heuristic rules A, B, C and D are used. In the rule A, numerals, adverbs and infinite verbs are also allowed. For the latter two word categories, only the category is checked, the case does not exist. In the rule B, beside commas and coordinating conjunctions the semicolons are accepted as well. The attribute has binary values. Motivation: if there is a possibility of a crossing intraclausal coordination, the segment may not represent a whole clause – other neighboring segments may have to be included to form the whole clause. Such a segment should not be reduced to avoid making errors of the type false positive.

To complete the attribute model, the class attribute with binary values is added.

Accordingly, in the attribute-value vector, each of the attributes listed above appears five times, except for the class attribute. In Table 3, the attribute-value description of the segment 'se še pretvarjale niso' from Figure 9 is presented. The complete attribute-value description comprises the following delimiter/segment pairs:

- [in]/'popolnoma neuporabnimi ključavnicami',
- [,]/'stare ure',
- [, ki]/'se še pretvarjale niso',
- [, da]/'gredo',
- [, in]/'mešanica druge ropotije'.

If the segment is classified positively or the heuristic rule E holds for the segment, a one-segment clause is identified and the segment is reduced. Although the whole clause consists of the segment and the delimiter in front of it, the delimiter is not reduced; the delimiter tokens play a key role in the dependency tree construction stage, especially for coordination structures. Special attention is paid to the delimiter to the right of the reduced segment. If the delimiter starts with a comma

[, ki]/'se še pretvarjale niso'

Attribute	Value
Coordinating conj.	0
Subordinating conj.	1
Punctuation token	'comma'
Relative pronoun	0
Aux. verb before particip.	'no'
Segment type	'verb'
Crossing intracl. coord.	0

[,]/'stare ure'

Attribute	Value
Coordinating conj.	0
Subordinating conj.	0
Punctuation token	'comma'
Relative pronoun	0
Aux. verb before particip.	'not_def'
Segment type	'non_verb'
Crossing intracl. coord.	0

Attribute	Value
Class	1

Table 3. The tables present the attribute-value description of the segment 'se še pretvarjale niso' from Figure 9. For clarity of the presentation the table shows only the attributes pertaining to the delimiter/segment pair [, ki]/'se še pretvarjale niso' and the neighboring pair [,]/'stare ure'.

followed by a coordinating conjunction, the comma always marks the end of the clause. In this case, the comma is reduced together with the segment.

In Figure 10, processing of the sentence from Figure 5 by the first stage of PACID is presented. For the example sentence, two iterations are performed. In step a), reduction of the intraclausal coordination from Figure 8 is depicted. The reduced sequences are pushed onto the stack for further processing in the second stage of PACID. The sequences reduced in the same iteration are pushed onto the same stack level. The algorithm continues with step b), where two one-segment clauses are identified. The segments are reduced and replaced with meta nodes. Together with the second segment, the comma from the following delimiter is reduced, because it is followed by a coordinating conjunction. The meta node is assigned the MSD-tag of the main verb of the segment.

The name of the meta node is assigned according to the type of the reduced clause that is determined by the heuristic rules defined as follows. Let $\mathcal{S}_i = (s_1, \ldots, s_n)$ be the segment replaced by the meta node we want to assign the name to, where $\mathcal{D}_{i-1} = (a_1, \ldots, a_k)$ and $\mathcal{D}_i = (b_1, \ldots, b_l)$ are the delimiters preceding and succeeding the segment, respectively.

Definition 12. The heuristic rule F holds *iff* one of the tokens a_i is a subordinating conjunction and none of the tokens b_i is a coordinating conjunction.

If the rule F holds, a type 1 subordinate clause is discovered and the name 'SUB_CLS_TYPE1' is assigned to the meta node. Type 1 subordinate clauses contain a subordinating conjunction. If the next delimiter ($\mathcal{D}_i$) contains a coordinating conjunction, it is very likely that the clause corresponding to the segment $\mathcal{S}_i$ and the clause containing the segment $\mathcal{S}_{i+1}$ are in a coordination. In such case, the clause corresponding to the segment $\mathcal{S}_i$ can not be treated as a subordinate clause.

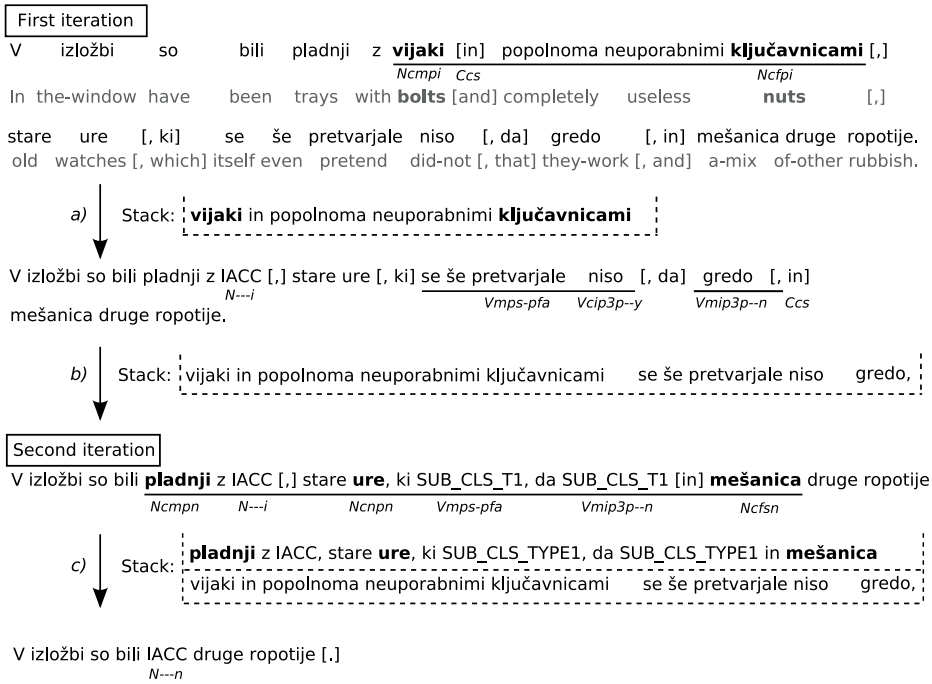


Fig. 10. The execution of the first stage of PACID is presented. There are two iterations. The reduced sequences are underlined, the head words of the intraclausal coordinations are set in boldface, the delimiters between the segments are enclosed in square brackets, MSD-tags are italicized, the intraclausal coordination meta nodes are named 'IACC', the clause meta nodes are named 'SUB_CLS_TYPE1', 'SUB_CLS_TYPE2' or 'COORD_CLS'.

If the rule F does not hold for the reduced segment, the next rule is applied:

Definition 13. The heuristic rule G holds *iff* none of the tokens a_i is a subordinating conjunction and one of them is relative pronoun.

If the rule G holds, a type 2 subordinate clause is discovered and the name 'SUB_CLS_TYPE2' is assigned to the meta node. If neither of the heuristic rules F or G hold, a coordinate clause is discovered and the meta node is named 'COORD_CLS'.

In the next iterations the delimiter tokens immediately preceding the meta nodes are temporarily removed during sentence segmentation and intraclausal coordination detection. Neither they play the role of segment delimiters nor do they influence the application of the heuristic rules B, C and D. In Figure 10 in step b), there are two such delimiters: [, ki] and [, da]. In step c), where the sequence ', stare ure, ki SUB_CLS_TYPE1, da SUB_CLS_TYPE1' represents only one segment, another intraclausal coordination is reduced. The reduction stage terminates after step c),

since only one verb segment exists and no more intraclausal coordinations can be retrieved.

3.2 The Second Stage – Dependency Tree Construction Stage

In this stage, the sequences of text reduced in the first stage are parsed, see Figure 11 and Algorithm 1. The resulting trees are merged to form the final dependency tree of the sentence. A rule-based parser is used plus three different MSTP parsing models for each of the following types of the token sequences, which have distinct syntactic structure:

1. the initial sequence which remains unreduced after the end of the first stage,
2. clauses and
3. intraclausal coordinations.

Choosing three different MSTP models enables the parser to focus on specific structure of a certain type of token sequence.

The stage begins by parsing the initial sequence of tokens, producing the initial sentence tree, Figure 11 a). Note that the technical root '#' is added in front of the sequence before parsing. The sequence is parsed using the MSTP parsing model for the initial sequences. Certain errors in the tree can be detected, as described further in Section 3.3. In such case, the tree produced by the MSTP parser is discarded and the rule-based parser is applied to construct a new initial sentence tree.

The stage continues with an iteration, which processes the sequences on the stack. In the first step of the iteration, the sequences are popped from the upper level of the stack, Figure 11 b). Then, the sequences are joined with the tokens of the corresponding meta-node subtrees inside the sentence tree, Figure 11 c). The meta nodes themselves are not added to the sequences. The extended sequences are parsed using two separate MSTP models, one for verb segments and the other for intraclausal coordinations, Figure 11 d). In the sentence tree, the meta node subtrees are replaced with the newly created subtrees, Figure 11 e).

Algorithm 1 Dependency tree construction stage

```
1: parse inital-sequence
2: repeat
3:     sequences = pop stack
4:     for all S ∈ sequences do
5:         S := S ∪ meta-node-subtree
6:         parse S
7:         replace meta-node-subtree with new-subtree
8:     end for
9:     rearrange stack
10: until stack is empty
```

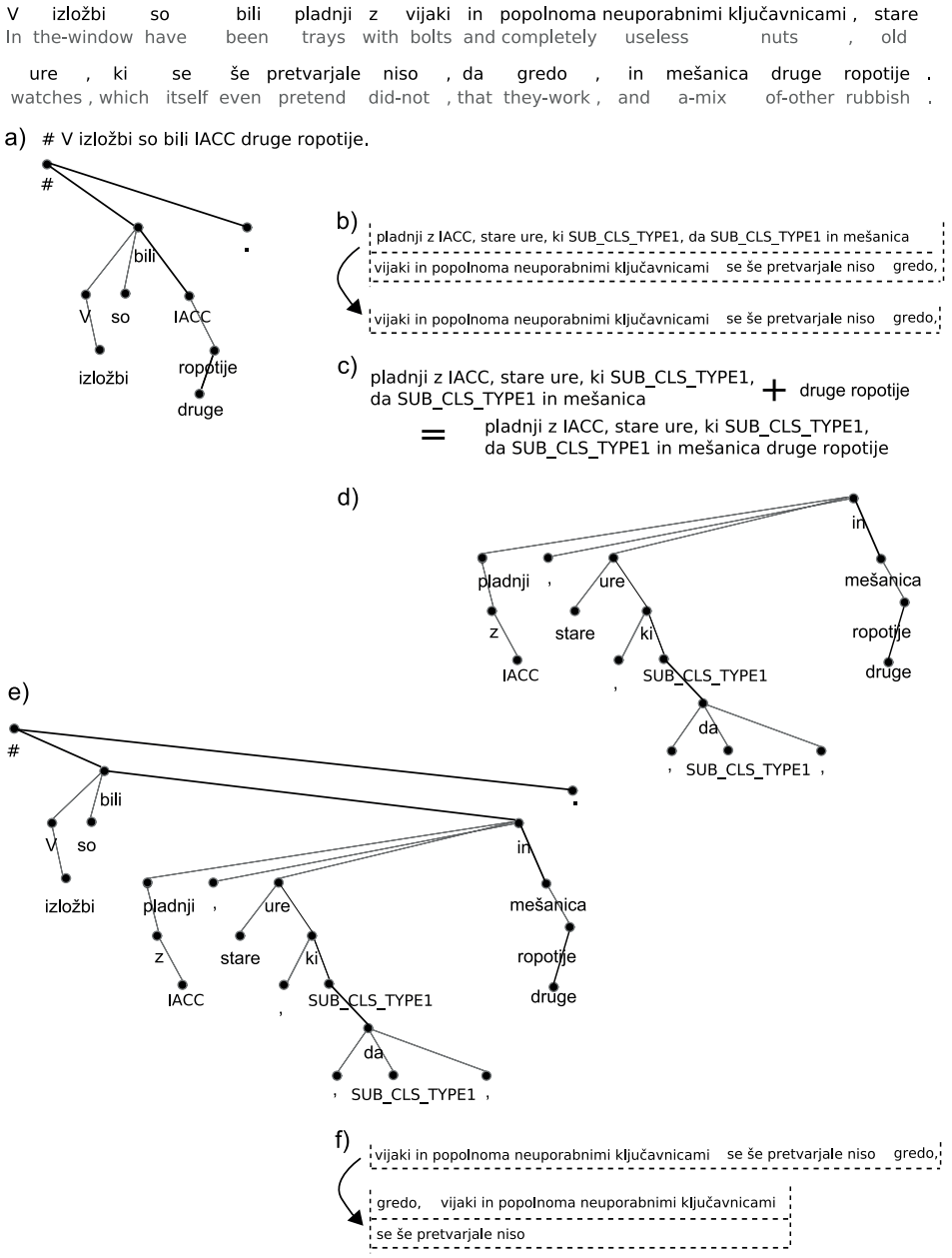


Fig. 11. The first iteration of the dependency tree construction stage is presented. On the left side, the growth of the sentence tree is presented. The right side shows how the sequences of tokens are popped from the stack and parsed into the subtrees.

Then, the sequences remaining on the stack are examined. The meta nodes of some sequences on the same stack level might now appear in the ancestor – descendant relation in the sentence tree. Such an example are both 'SUB_CLS_TYPE1' nodes in Figure 11 e). In this case, the stack is rearranged so that the sequences of the meta nodes closer to the leaves are processed first, Figure 11 f).

The iteration terminates when the stack is empty. In Figure 11, only the first iteration is presented. To get the complete sentence tree of this example, two more iterations are needed.

3.3 Rule-Based Parser

The rule-based parser is used to correct certain errors in the initial dependency tree created by the MSTP parser. The error correction is tried for trees containing only punctuation tokens, conjunctions and meta nodes. All the paths starting with the root and ending with the leaves are inspected, whereby only meta nodes are considered. If there exists a path having a subordinate clause meta node at the beginning, this is treated as an error, because a subordinate clause cannot be the main clause of the sentence, neither can it be coordinated with another coordinate clause. The initial tree is constructed from scratch by the rule-based parser in three passes over the sentence.

In the first pass, see Algorithm 2, the skeleton of the initial tree is created containing the coordinate clause meta nodes. A simplification is used: it is presumed that there is at most one coordination of clauses in the sentence; if there is one, the coordination subtree resides on the highest level in the tree, right under the technical root of the tree.

The first pass proceeds as follows: if there is only one coordinate clause meta node, it is placed directly below the technical root; otherwise, a subtree is created, appended directly to the technical root. The root of the subtree is the punctuation mark or the coordinating conjunction directly preceding the last coordinate clause meta node. The meta nodes themselves and other delimiter tokens directly preceding them become children of the root node.

Algorithm 2 Rule-based parser, pass 1

```
1: if ∃!coord-clause-meta-node then
2:     append coord-clause-meta-node to technical-root
3: else
4:     append coordination-root to technical-root
5:     for all ν ∈ coord-clause-meta-nodes do
6:         append ν to coordination-root
7:         append delimiter-preceding-ν to coordination-root
8:     end for
9: end if
```

In the second pass, see Algorithm 3, the positions of the subordinate clauses in the tree are determined. It is presumed that the subordinate clauses depend on the nearest clause to the left – if there exists one; otherwise they depend on the nearest clause to the right. Note that a subordinating conjunction is the root of a type 1 subordinate clause and is always appended to the tree before the corresponding meta node.

The second pass goes through all the nodes of the sentence, either already appended to the tree or not, from the left to the right. When a subordinating conjunction is encountered, it is appended to the closest meta node to the left. If no meta node to the left exists, the subordinating conjunction becomes the child of the next coordinate clause meta node to the right. The type 1 subordinate clause meta nodes are appended to the subordinating conjunctions preceding them. The type 2 subordinate clause meta nodes are appended to the closest meta node to the left; if there are no meta nodes to the left they are appended to the closest meta node to the right.

Algorithm 3 Rule-based parser, pass 2

```
1: ζ := leftmost-coord-clause-meta-node
2: for all ν ∈ all-nodes-of-sentence do
3:     if ν is subordinate-conjunction or sub-clause-meta-node then
4:         append ν to ζ
5:     end if
6:     if ν is subordinate-conjunction or clause-meta-node then
7:         ζ := ν
8:     end if
9: end for
```

Finally, in the third pass, the commas directly preceding the subordinate clause meta nodes are appended to them. All the remaining nodes are appended to the closest meta node to the left, except for the final punctuation mark, which is placed directly under the technical root-node.

4 EVALUATION

PACID was tested on the SDT corpus. All the experiments were done with 10-fold cross-validation (except for the experiment on the CoNLL-X shared task data [2]), by dividing the corpus into 10 disjunctive parts. In each fold, one of the parts was used for testing, the other nine for training. The input data for the train phase as well as for the test phase were obtained from the SDT corpus, meaning that no automatic MSD-tagging and lemmatization was used in our experiments. The first experiment was designed with the aim of evaluating retrieval of clauses and intraclausal coordinations. In the second experiment, the overall parsing accuracy of PACID was estimated.

4.1 Clause and Intraclausal Coordination Retrieval

In this experiment only the first stage of PACID was examined. First, precision and recall at retrieval of intraclausal coordinations was measured. In the test set the distribution of different groups of intraclausal coordinations according to the category of head words as follows: prepositions 6 %, nouns 42 %, adjectives 31 %, other categories 21 %. Each type was analyzed separately. The results are presented in Table 4. Since PACID detects prepositional, nominal and adjectival coordinations only, the fourth group is not considered in the measurements.

Coordination type	Prepositional	Nominal	Adjectival	All
Recall	60 %	72 %	81 %	79 %
Precision	69 %	69 %	95 %	78 %

Table 4. Recall and precision of intraclausal coordination retrieval

As expected, the highest recall and precision were achieved on the least complex adjectival coordinations. Among the false positives at nominal coordination retrieval, 32 % were actually appositions. These cases should rather not be viewed as errors, but as a positive contribution: appositions are represented as subtrees as well and the same mechanism can be applied to them as to intraclausal coordinations. However, the problem of distinguishing nominal coordinations and appositions remains out of the scope of this paper.

For prepositional coordinations, recall and precision were the lowest due to their high complexity compared to the other types of intraclausal coordinations. Furthermore, the prepositional coordinations where the associated nouns do not have the same case are not considered by the PACID algorithm, since the heuristic rules do not recognize them.

The performance of clause retrieval was evaluated with the second set of tests. The results are presented in Table 5. Only the sentences containing two segments or more were considered. The clause type is determined according to the segment of the clause's main verb. The distribution of the types in the test set as follows: 52 % coordinate clauses, 36 % type 1 subordinate clauses, 12 % type 2 subordinate clauses.

Clause type	Coord.	Type 1 Subord.	Type 2 Subord	All
Recall	68 %	70 %	75 %	70 %
Precision	91 %	95 %	95 %	93 %

Table 5. Recall and precision of clause retrieval

The experiment shows better results for subordinate clauses. In coordinate clauses one can find embedded clauses more often than in subordinate clauses which makes retrieval of coordinate clauses a difficult problem. Regarding the type 1 subordinate clauses, the presence of the subordinating conjunction is very important.

It unambiguously marks the beginning of the clause and thus contributes to better precision and recall for this type of clauses.

Intraclausal coordination and clause detection can raise the accuracy of parsing if the positive influence of constraining the parsing process is larger than the impact of causing additional errors by reducing wrong sequences. Therefore, high precision at intraclausal coordination and clause detection is preferred, while high recall might not be crucial, since missing out some valid clauses and intraclausal coordinations does not introduce new errors.

4.2 Evaluation of Dependency Parsing

In this section the evaluation of the complete algorithm PACID is presented. The performance measure was UAS (unlabeled attachment score). This is the quotient between

1. the number of scoring tokens with the correctly assigned parent and
2. the number of all scoring tokens in the test set.

We define the scoring tokens as follows: all words are scoring tokens; a punctuation mark is a scoring token only if it is the root of a coordination subtree in the gold tree. Thus, most of the punctuation marks are excluded from the measurements.

The first set of tests was conceived to analyze the differences between various versions of PACID. The results are presented in Table 6. First, we measured the accuracy of the plain MSTP parser without clause and intraclausal coordination detection. The parser scored the best result on the CoNLL-X dependency parsing shared task for Slovene [2]. This result serves as the baseline. Then, PACID including the complete algorithm for intraclausal coordination detection but without clause identification was used. In the third test, both clauses and intraclausal coordinations were retrieved; since ML filtering was not used, all candidates admitted by the heuristic rules were reduced. This version did not include the rule-based parser. In the next version, ML classifiers were turned on while the rule-based parser was still switched off. Finally, the accuracy of the full version was measured.

PACID version	Accuracy
Plain MSTP	80.24 %
PACID, intraclausal coord. detection, no clause detection	80.57 %
PACID, no ML in reduction stage, no rule-based parser	*81.05 %
PACID, ML in reduction stage, no rule-based parser	*81.34 %
PACID, full version	*81.51 %

Table 6. The table shows the parsing accuracy of various versions of PACID, compared to the baseline result achieved by the plain MSTP parser. The results marked with * are statistically significantly better than the baseline at the 95 % confidence level.

As expected, the full version achieved the highest accuracy. Compared to the baseline result, this presents a 6.4 % relative decrease of error. Compared to

the fourth test, the use of the rule-based parser increased the accuracy. In [10], the accuracy of the MSTP parser on sentences of various complexity was examined. The lowest accuracy was measured for the sentences without verbs, which are normaly the shortest ones. Since the target of the rule-based parser are short sequences of meta nodes and delimiter tokens, the rule-based parser might compensate for the inability of the MSTP parser to deal with very simple sentences effectively. In the version without ML classifiers, the errors of retrieving false positives among clauses and intraclausal coordinations contributed to worse results. The version without clause detection shows that even by using intraclausal coordination detection only, the algorithm still achieves better results.

Sometimes, only a part of an intraclausal coordination or a clause is retrieved in the first stage. An example of this phenomenon can be found in Figure 10 in step c), where the tokens 'druge ropotije' are left out although they are part of the coordination reduced in this step (see Figure 5). If the parser appends these tokens to the correct meta node as in Figure 11 a), they can still be placed into the correct subtree by this self-correcting mechanism, as shown in Figure 11 d).

Another test was performed using the CoNLL-X SDT data set. Here, the SDT corpus was divided into a train set (5/6 of data) and test set (1/6) of the data. The UAS of the plain MSTP was 82.96 % while the accuracy of the full version of the PACID algorithm was 83.07 %. The difference is small compared to the difference measured with 10-fold cross validation. This could be accounted for by large variation of results due to the small datasets.

We further inspected PACID at processing sentences of various complexity. The output of the full version was analyzed separately for test sentences containing one, two, three, four and more than four clauses. In Table 7, the increase of accuracy in percentage points (pp) compared to the baseline result achieved on the same set of test sentences is shown.

Number of clauses	1	2	3	4	>4
Accuracy increase (pp)	*2.77	*1.22	1.05	−0.01	0.01

Table 7. The results of parsing sets of sentences containing one, two, three, four and more than four clauses are presented. The table shows the difference of the accuracy between the full version of PACID and the plain MSTP on the same data set. The measurement units are percentage points. The differences marked with * are statistically significant at the 95 % confidence level.

At the first glance it seems surprising that improvement is the largest for one clause sentences. Two additional tests were performed on one clause sentences, both compared to the plain MSTP parser:

- PACID retrieving intraclausal coordinations only, no clause retrieval. This version improved the accuracy by 3.03 pp.
- PACID retrieving clauses only, no intraclausal coordination retrieval. In this test, the accuracy increase of 0.24 pp was achieved.

The first result is statistically significantly different than the baseline while the second one is not. This confirms the obvious expectation that clause detection does not help for one clause sentences. The improvement of accuracy can thus be attributed to intraclausal coordination detection.

4.3 Analysis of Errors

In general, as the complexity of a sentence increases, PACID helps less and less. A detailed manual analysis has shown that the algorithm is confronted with decisions exhibiting high probability of error. In such cases PACID does not make a decision since it was designed to minimize the number of false positives in the reduction stage. This effectively reduces the algorithm to behave like the plain MSTP parser.

We sum up the following main situations, where the PACID algorithm usually performs many errors:

- Sentences containing a lot of non-verb segments: errors at determining the clausal structure by the detection part of the algorithm.
- Ellipsis of the main verb in a clause: the heuristic rules for the detection of clauses do not detect such a clause.
- Sentences with deeply embedded clauses and intraclausal coordinations: the most deeply embedded entities are usually successfully retrieved, while the multi-segment outer entities pose a problem, because they are constituted of a discontinuous sequence of segments.
- Wrong values in the MSD-tags: the heuristic rules do not tolerate some of the errors in the morphosyntactic annotation. The errors of the type false negative occur, i.e. valid coordinations are not recognized. The precision of retrieval remains the same, meaning that such detection errors do not cause additional parsing errors.

4.4 Comparison with Commercial Products

Since we developed an algorithm that could be used in products interesting for general public, it would merit a comparison with commercial products used for the same purpose. However, there are some problematic issues concerning such a comparison. There are hardly some Slovene language resources annotated on the syntactic or higher level available for the commercial use. The SDT, which is the main language resource used in our experiments is unfortunately only available for non-profit scientific research, meaning that a comparison on the same data set is not possible. Furthermore, detailed publications about similar technologies and results of the industrial state-of-the-art are generally not available. All this makes the direct comparison of our scientific achievements with commercial products almost impossible in terms of providing concrete relevant numbers.

5 CONCLUSIONS AND FUTURE WORK

Our experiments have shown that decomposing large parsing problems to smaller ones is beneficial in terms of improving the overall parsing accuracy. This was achieved by upgrading the approach as used by the MSTP parser with ML and rule-based methods that rely on knowledge about the language. Certain language phenomena, such as the structure of intraclausal coordinations, seem to be hard to discover even from the text annotated with dependency trees.

Considering the statistically significant improvement of 1.27 % percentage points one should keep in mind that PACID focuses only on multi-clause sentences and/or sentences containing intraclausal coordinations, which represent 70 % of all sentences in the test set. Since the time complexity of the reduction mechanism equals $O(n)$, n being the number of tokens in the sentence, compared to the complexity $O(n^2)$ of the MSTP algorithm, additional time consumption is acceptable.

In summary, we have shown that additional information provided by the richly inflected languages can improve parsing results. Although the PACID algorithm was tested for Slovene, it could be ported to other languages with similar patterns of inflection. Not only other Slavic languages are among them, the Baltic languages seem suitable, some Finno-Ugric as well. For example, in Finnish, prepositional phrases can be expressed by inflecting the nouns. Some Germanic languages, e.g. German, show properties that could be exploited by the PACID algorithm as well. On the other hand, languages like Chinese or English have much poorer inflection than Slovene. They would not be among the most appropriate target languages, because some other features would have to be employed for the PACID algorithm to achieve good results.

First position: *word category*

V	verb
N	noun
A	adjective
R	adverb
S	preposition
C	conjunction

Preposition, third pos., noun, fifth pos., adjective, sixth pos.: *case*

n	nominative
g	genitive
d	dative
a	accusative
l	locative
i	instrumental

Noun, third pos., adjective, fourth pos.: *gender*

m	masculine
f	feminine
n	neuter

Noun, fourth pos., adjective, fifth pos.: *number*

s	singular
p	plural
d	dual

Conjunct., second pos.: *type*

c	coordinating
s	subordinating

Verb, third pos.: *verb form*

i	indicative
p	participle

Fig. 12. Explanation of MSD-tag positions

There are further ways how to improve PACID. One of the current problems is rigid treatment of reduced units. PACID either declares a sequence of tokens to be reduced or not. It would probably be better to raise the weights of the appropriate edges in the sentence graph and let the maximum spanning tree algorithm in the MSTP parser find the best solution. Moreover, the set of attributes used in ML

classifiers could be extended. Also, the rule-based parser could be more elaborated. The most important further improvement seems to be better treatment of very complex sentences.

6 APPENDIX

In Figure 12, the most important positions of MSD-tags in SDT are described. Note that the same feature may be described with different positions for different word categories.

REFERENCES

[1] Abney, S.: Rapid Incremental Parsing with Repair. In: Proceedings of the 6th New OED Conference, University of Waterloo, Waterloo, Ontario, 1990, pp. 1–9.

[2] Buchholz, S.—Marsi, E.: CoNLL-X Shared Task on Multilingual Dependency Parsing. In: Proceedings of the Tenth Conference on Computational Natural Language Learning, CoNLL-X, New York City, USA, 2006, pp. 149–164.

[3] Covington, M. A.: Parsing Discontinuous Constituents in Dependency Grammar. Computational linguistics, 1990(16), pp. 234–236.

[4] Covington, M. A.: A Fundamental Algorithm for Dependency Parsing. In: Proceedings of the 39th Annual ACM Southeast Conference, Athens, Georgia, USA, 2001, pp. 95–102.

[5] Džeroski, S.—Erjavec, T.—Ledinek, N.—Pajas, P.—Žabokrtský, Z.—Žele, A.: Towards a Slovene Dependency Treebank. In: Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC 2006, Genova, Italy, 2006, pp. 1388–1391.

[6] Erjavec, T.: MULTEXT-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In: Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC 2006, Genova, Italy, 2006, pp. 1388–1391.

[7] Hogan, D.: Empirical Measurements of Lexical Similarity in Noun Phrase Conjuncts. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Prague, Czechia, 2007, pp. 149–152.

[8] Holan, T.—Žabokrtský, Z.: Combining Czech Dependency Parsers. In: Proceedings of the Ninth International Conference on Text, Speech and Dialogue, TSD 2006, Brno, Czech Republic, 2006, pp. 95–102.

[9] Kuboň, V.—Lopatková, M.—Plátek, M.—Pognan, P.: Segmentation of Complex Sentences. In: Proceedings of the Ninth International Conference on Text, Speech and Dialogue, TSD 2006, Brno, Czechia, 2006, pp. 151–158.

[10] Marinčič, D.—Gams, M.—Šef, T.: How Much Can Clause Identification Help to Improve Dependency Parsing? In: Proceedings of the Tenth International Multiconference Information Society, IS 2007, Ljubljana, Slovenia, 2007, pp. 92–94.

[11] Marinčič, D.—Gams, M.—Šef, T.—Žabokrtský, Z.: Parsing Aided by Intra-clausal Coordination Detection. In: Proceedings of The Sixth International Workshop on Treebanks and Linguistic Theories, TLT 2007, Bergen, Norway, 2007, pp. 79–84.

[12] McDonald, R.—Pereira, F.—Ribarov, K.—Hajič, J.: Non-projective Dependency Parsing Using Spanning Tree Algorithms. In: Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing, HLT-EMNLP 2005, Vancouver, Canada, 2005, pp. 523–530.

[13] Melčuk, I.: Dependency Syntax: Theory and Practice. State University of New York Press, USA, 1988.

[14] Nilsson, J.—Nivre, J.—Hall, J.: Generalizing Tree Transformations for Inductive Dependency Parsing. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Prague, Czechia, 2007, pp. 968–975.

[15] Nivre, J.—Hall, J.—Kuebler, S.—McDonald, R.—Nilsson, J.—Riedel, S.—Yuret, D.: The CoNLL 2007 Shared Task on Dependency Parsing. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing and the Conference on Computational Natural Language Learning, EMNLP-CoNLL 2007, Prague, Czechia, 2007, pp. 915–932.

[16] Nivre, J.: Inductive Dependency Parsing. Springer, Dordrecht, Netherlands, 2006.

[17] Quinlan, J. R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1993.

[18] Ohno, T.—Matsubara, S.—Kashioka, H.—Maruyama, T.—Inagaki, Y.: Incremental Dependency Parsing of Japanese Spoken Monologue Based on Clause Boundaries. In: Proceedings of the joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics, COLING/ACL-2006, Sydney, Australia, 2006, pp. 169–176.

[19] Orasăn, C.: A Hybrid Method for Clause Splitting in Unrestricted English Texts. In: Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications, ACIDCA 2000, Monastir, Tunisia, 2000, pp. 129–134.

[20] Puscasu, G.: A Multilingual Method for Clause Splitting. In: Proceedings of the Seventh Annual CLUK Research Colloquium, Birmingham, UK, 2004, pp. 199–206.

[21] Freund, Y.—Schapire, R. E.: Experiments with a New Boosting Algorithm. In: Proceedings of the 13th International Conference on Machine Learning, ICML '96, Bari, Italy, 1996, pp. 148–156.

[22] Tesnière, L.: Éléments de Syntaxe Structurale. Editons Klincksieck, Paris, France, 1959.

[23] Tjong Kim Sang, E. F.—Déjean, H.: Introduction to the CoNLL-2001 Shared Task: Clause Identification. In: Proceedings of the Conference on Computational Natural Language Learning, CoNLL-2001, Toulouse, France, 2001, pp. 53–57.

[24] Toporišič, J.: Slovene Grammar (Slovenska Slovnica). Založba Obzorja, Maribor, Slovenia, 2000.

[25] Witten, I. H.—Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. 2nd edition. Morgan Kaufmann, San Francisco, CA, USA, 2005.

Domen Marinčič is a researcher at the Department of Intelligent Systems, Jozef Stefan Institute, Ljubljana, Slovenia. He received his Ph. D. in computer science at the Jozef Stefan international postgraduate school in 2008. His major research interests include language technologies, dependency parsing of unrestricted Slovene text, artificial intelligence, ambient intelligence and computer game playing.

Tomaž Šef is a Senior Researcher at the Department of Intelligent Systems at the Jozef Stefan Institute, Ljubljana, Slovenia. He received his Ph. D. degree in computer science (language and speech technologies) in 2001 at the University of Ljubljana. His research interests include artificial intelligence, intelligent systems, natural language processing, speech processing, Slovenian text-to-speech synthesis and forensic speaker identification. He developed the Slovenian text-to-speech system "Govorec" (Speaker) donated to several thousand users.

Matjaž Gams is an Associate Professor of computer and information science at the University of Ljubljana and a Senior Researcher at the Jozef Stefan Institute, Ljubljana, Slovenia. He teaches several courses in computer science at graduate and postgraduate levels at Faculties of Computer and Information Science, Economics, etc. His research interests include artificial intelligence, intelligent systems, intelligent agents, machine learning, cognitive sciences, and information society. He has headed several major artificial intelligence applications in Slovenia, including the major national employment agent on the Internet, and the Slovenian text-to-speech system "Govorec" (Speaker).