Parsimonious Vole

A Systemic Functional Parser for English

Universität Bremen

Eugeniu Costetchi

Supervisor: Prof. John Bateman

Advisor: Dr. Eric Ras

Faculty 10: Linguistics and Literary Studies
University of Bremen

This dissertation is submitted for the degree of
Doctor of Philosophy

January 2019

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Eugeniu Costetchi
January 2019

Acknowledgements

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Table of contents

List of figures

List of tables

List of definitions

Chapter 1

On Graphs, Feature Structures and Systemic Networks

The parsing algorithm, whose pipeline architecture we have seen in Section **??**, operates mainly with operations on graphs, attribute-value matrices and ordered lists with logical operators. This chapter defines the main types of graphs, their structure and how they are used in the following chapters which detail on the parsing process. This chapter also covers the operations relevant to the parsing algorithm: *conditional traversal and querying* of nodes and edges, *graph matching*, *pattern-graph matching* and *pattern-based node selection, insertion and update*.

While developing the Parsimonious Vole parser a set of representational requirements arose that can be summarised as follows:

- graphic representation
- arbitrary relations (i.e. typed and untyped edges)
- description rich (i.e. features of nodes and edges)
- linear ordering and configurations (i.e. syntagmatic and compositional)
- hierarchical tree-like structure (with a root node) but also orthogonal relations among siblings and non-siblings
- statements of absence of a node or edge (i.e. negative statements in pattern graphs)
- disjunctive descriptions (handling uncertainty)

- conjunctive descriptions (handling multiple feature selections)
- (conditional) pattern specifications (i.e. define patterns of graphs)
- operational pattern specifications (i.e. a functional description to be executed in pattern graphs)

The general approach to construct an SFG parse structure revolves around the graph pattern matching and graph traversal. In the following sections I present the instruments used for building such structures, starting from a generic computer science definition of graphs and moving towards specific graph types covering also the feature structures and conditional sets.

1.1 On sets, feature structures and graphs

In the field of computational linguistics trees has been taken as the de facto data representation. In Section **??** I have mentioned already that I employ graph and not tree structures.

Firstly, the trees are a special kind of graphs. Anything expressed as a tree is as well a tree. Secondly, we gain a higher degree of expressiveness even if at the expense of computational complexity, a point to which we will come back latter in Section 1.4. This expressiveness is needed when dealing with interconnection of various linguistic theories which in practice is done by mapping the nodes of one tree structure onto the nodes of another one. In addition, the structures are not always trees. There are situations when a node has more than one parent or when a node is connected to its siblings which break the tree structure.

Definition 1.1.1 (Graph)**.** A *graph* $G = (V, E)$ is a data structure consisting of non-empty set V of nodes and a set $E \subseteq V \times V$ of edges connecting nodes.

Definition 1.1.2 (Digraph)**.** A *digraph* is a graph with directed edges. A directed edge $(u, v) \in E$ is an ordered pair that has a start node u and an end node v (with $u, v \in V$)

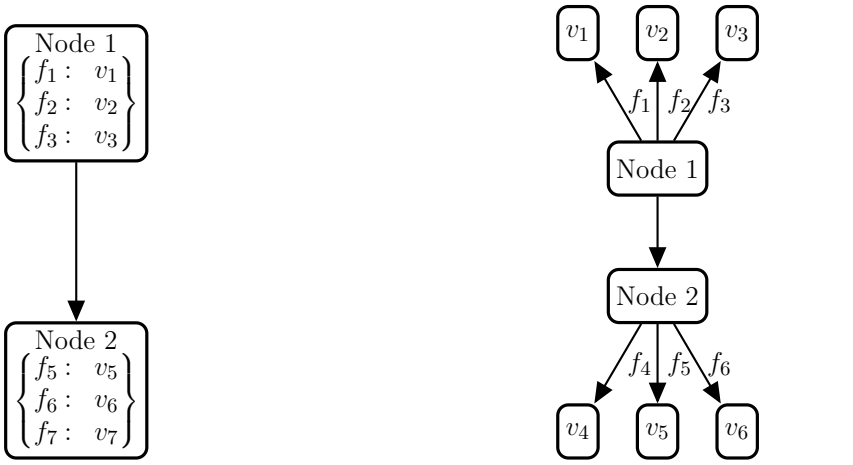
In this thesis the graph nodes are considered to be *feature structures* forming *Feature Rich Graphs* (see Definition 1.1.10). Before formally defining these graphs, I need to address first the notion of feature structure and a few kinds of sets.

In SFL the concept of *feature* takes up an important role. Also features are said to form systems of choices that are structured in relation to one another and are suitable for describing linguistic objects and phenomena.

Pollard & Sag (1987) have formally described useful concepts for grammatical representations in the context of Head-Driven Phrase Structure Grammar (HPSG). He adopts the *typed feature structure theory* and extends it in ingenious ways applicable in computational linguistics. Among others, he provides formal definitions for the concepts of *feature structure*, *hierarchy*, *logical evaluation*, *composition* and *unification*, the latter, being key operations in parsing using feature structured grammars.

In this thesis, feature structures are important but only in a simplified version serving as graph node descriptions. The main reason is the difference in approach as the main parsing operations, here, are based on graph pattern matching (introduced in the sections below).

In a broad computer science sense, including Pollard's definition, feature structures are equivalent to graph structures. So any feature structure can be expressed as a graph and any graph can be expressed as a feature structure. But in a narrow sense, as adopted in this thesis, it is useful to employ both concepts but each for a given purpose. The feature structure is reduced to an attribute-value matrix (see Definition 1.1.3) and the graphs to a network of feature structure nodes (see Definition 1.1.10) i.e. no atomic nodes.



(a) The graph with feature structure nodes

(b) The graph with atomic nodes

Fig. 1.1 Graphs with atomic nodes and feature structure nodes

The main reasons in this separation are efficiency and practicality. First, it is about handling the atomic values (strings or integers) and (ordered) arrays only as values of feature structures and never as graph nodes. Second, the graphs remain limited in

size, close to the conceptualised linguistic structures, i.e. dependency or constituency. Otherwise, the graphs would grow in complexity (a) by at least one more round of nodes for each dependency or constituency node and (b) by adoption of an additional node classification.

For example lets imagine a constituency graph fragment of two nodes *Node 1* and *Node 2* where each has three associated features as it can be seen in Figure 1.1a. If we would insist to dispose of the feature structure within the node and express the features as atomic graph nodes then the result would be a graph structure such as the one in Figure 1.1b.

Definition 1.1.3 (Feature Structure (FS))**.** A *feature structure* F is a finite set of attribute-value tuples $f_i \in F$. A *feature* $f = (a, v)$ is an association between an identifier a (a symbol) and a value v which is either an atomic value (symbol, number, string), a set or another feature structure.

The values of feature structures may be other feature structures allowing, if needed, to construct hierarchical descriptions. In the current implementation, however, the values of the feature structure are restricted to atomic values or sets of values.

For convenience I define two functions to access the identifier and value in a feature structure. The function $name : F \rightarrow symbol$ returns the feature symbol (identifier) $name(f) = a$ and the function $val : F \rightarrow \{atomic, Set, FS\}$ is a function returning the ascribed value of a feature $val(f) = v$.

Definition 1.1.3 stipulates that the value of a feature may be also a set (besides an atomic value). The sets used in this thesis need to carry additional properties required for their interpretation. Specifically, it is the order need to be addressed here and the capacity to specify that set elements stand in a certain logical relation one to another (e.g. conjunction, disjunction, negation, etc.). These two properties are covered in Definition 1.1.4 and 1.1.5. For convenience I will assume from now on that sets (see Definition 1.1.4) preserve order even when it is not really required.

Definition 1.1.4 (Set)**.** An (ordered) *set* $S = \{o_1, o_2, ..., o_n\}$ is a finite well defined collection of distinct objects o_i. A set is said to be ordered if the objects are arranged in a sequence such that $\forall o_{i-1}, o_i \in S : o_{i-1} < oi$.

Definition 1.1.5 (Conjunction Set)**.** A *conjunction set* $S_{conj} = (S, conj)$ is a set S whose interpretation is given by the logical operand $conj$ (also denoting the type of the set) such that $\forall o_i, o_j \in S : conj(o_i, o_j)$ holds.

The conjunction sets used in current work are *AND-set* (S_{AND}), *OR-set* (S_{OR}), *XOR-set* (S_{XOR}) and *NAND-set* (S_{NAND}). The assigned logical operands play a role in the functional interpretation of conjunction sets. Formally these sets are defined as follows.

Definition 1.1.6 (Conjunctive set)**.** Conjunctive set (also called *AND-set*) is a conjunction set $S_{AND} = \{a, b, c...\}$ that is interpreted as a logical conjunction of its elements $a \wedge b \wedge c \wedge ...$

Definition 1.1.7 (Negative conjunctive set)**.** Negative conjunctive set (also called NAND-set) is a conjunction set $S_{NAND} = \{a, b, c...\}$ that is interpreted as a negation of the logical conjunction of its elements $a \uparrow b \uparrow c \uparrow ...$ equivalent to $\neg(a \wedge b \wedge c \wedge ...)$

Definition 1.1.8 (Disjunctive set)**.** Disjunctive set (also called OR-set) is a conjunction set $S_{OR} = \{a, b, c...\}$ that is interpreted as a logical disjunction of its elements $a \vee b \vee c \vee ...$

Definition 1.1.9 (Exclusive disjunctive set)**.** Exclusive disjunctive set (also called XOR-set) is a conjunction set $S_{XOR} = \{a, b, c...\}$ that is interpreted as a logical exclusive disjunction of its elements $a \bigoplus b \bigoplus c \bigoplus ...$ equivalent to $(a \wedge \neg(b \wedge c \wedge ...)) \vee (b \wedge \neg(a \wedge c \wedge ...)) \vee (c \wedge \neg(a \wedge b \wedge ...))$

When conjunction sets are used as values in FSs then the logical operand dictates the interpretation of the FS. When the set type is S_{AND} then all the set elements hold simultaneously as feature values. If it is a S_{OR} then one or more of the set elements hold as values. If is S_{XOR} then one and only one of set elements holds and finally if it is a S_{NAND} set then none of elements hold as feature values.

The function $\tau(S)$, defined $\tau : S \to \{S_{AND}, S_{OR}, S_{XOR}, S_{NAND}\}$, returns the type of the conjunction set and the function $size(S)$, defined $size : S \to \mathbb{N}$, returns the number of elements in the set. The size function is also denoted as $|S|$.

Now that all the necessary basic notions nave been formally defined I now define the feature rich graph and provide a couple of examples afterwards.

Definition 1.1.10 (Feature Rich Graph (FRG))**.** A *feature rich graph* is a digraph whose nodes V are feature structures and whose edges $(u, v, f) \in E$ are three valued tuples with $u, v \in V$ and $f \in F$ an arbitrary feature structure.

Further on, for convenience, when I refer to a graph I will refer to a feature rich digraph unless otherwise stated. The parsing algorithm operates with such graph and they are further distinguished, based on purpose as: *Dependency Graphs* (DG)

(example figure 1.2), *Constituency Graphs* (CG) (Figure 1.3) and Pattern Graphs (PG) also referred as *Query Graphs* (QG).

Please note that the edges are defined to carry feature structures. This capacity will not be employed, for example, in the case of constituency graphs; only minimally employed in the case of dependency graphs, where the dependency relation is specified; and fully employed in pattern graphs. Nonetheless, treating all of them as feature rich graphs simplifies the implementation.

Definition 1.1.11 (Dependency Graph)**.** A *dependency graph* is a feature rich digraph whose nodes V correspond to words, morphemes or punctuation marks in the text and carry at least the following features: *word*, *lemma*, part of speech (*pos*) and, when appropriate, the named entity type (*net*); the edges E describe the dependency relation (*rel*).

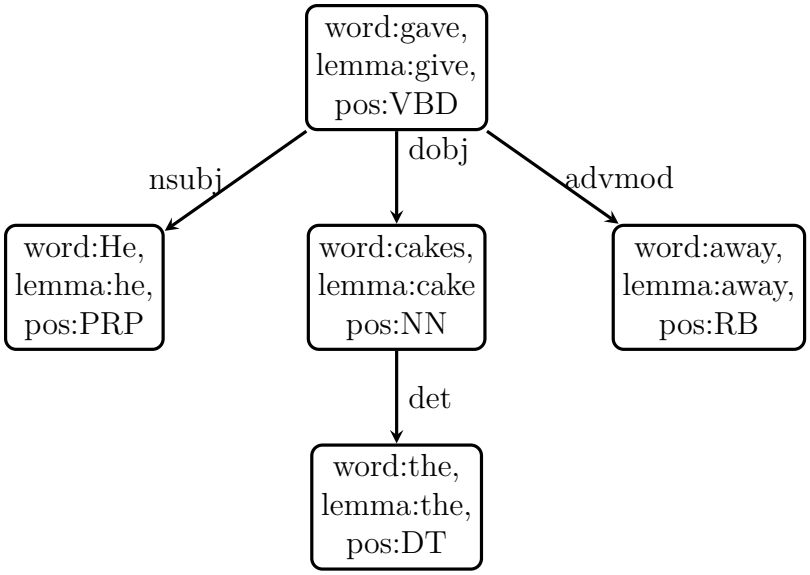


Fig. 1.2 Dependency graph example with FS nodes and edges

Definition 1.1.12 (Constituency Graph)**.** A *constituency graph* is a feature rich digraph whose nodes V correspond to SFL *units* and carry the *unit class* and the element function within the parent unit (except for the root node); while the edges E represent constituency relations between constituents.

The basic features of a constituent node are the *unit class* and the function(s) it takes, which is to say the *element(s)* it fills in the parent unit (as described in the discussion of theoretical aspects of SFL in Chapter **??**). The root node (usually a

clause) is an exception and it does not act as a functional element because it does not have a parent unit. The leaf nodes carry the same features as the DG nodes plus the word class feature which correspond to the traditional part of speech tags.

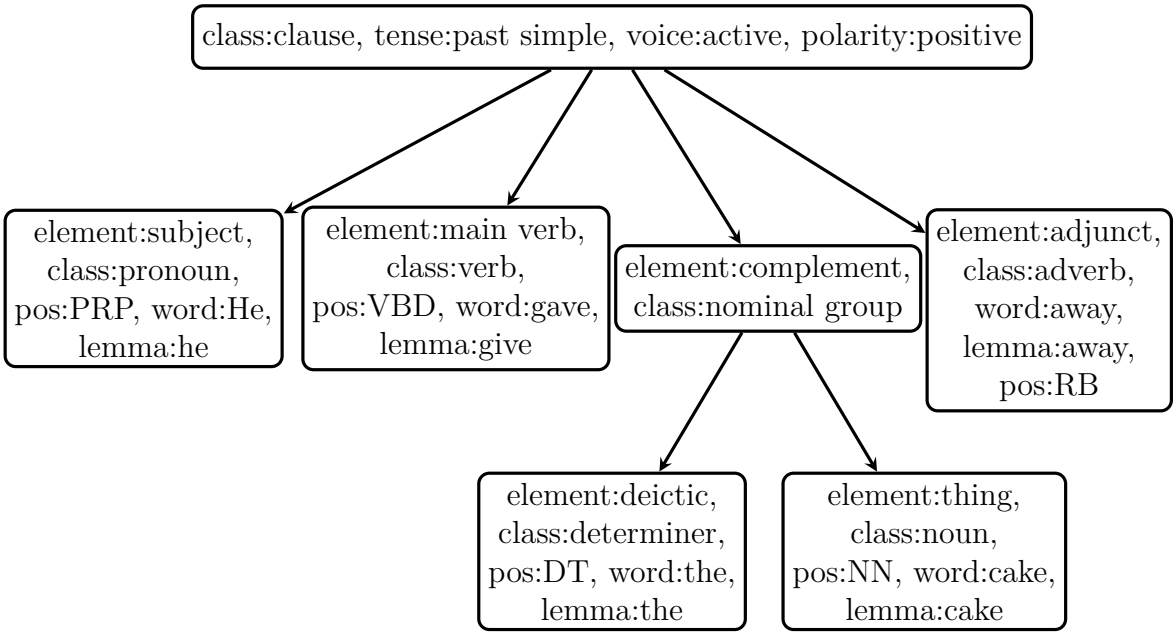


Fig. 1.3 Constituency graph example

Apart from the essential features of class and function, the CG nodes carry additional class specific features selected from the relevant system network. The features considered in this thesis are described in Chapter **??**. In addition, the leaf CG nodes contain the features of dependency graph nodes enumerated in Definition 1.1.11. The way CG is enriched with features is described in the next chapter. Below in Figure 1.3, is an example CG that carries tense, modality and polarity features on the clause node in addition to class and element function.

1.2 Graph traversal

From the general set of operations on graphs defined in graph theory (Bondy et al. 1976; West et al. 2001) graph traversal in particular is of importance for the current work. It is used for the constituency graph creation step presented in the parsing pipeline from Figure **??** (in Section **??**). The constituency graph is created throughout the traversal of a dependency graph. Traversal is used in this thesis for dependency graphs only. Next I address this operations in detail.

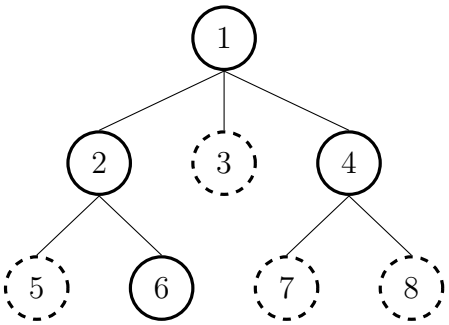


Fig. 1.4 Sample graph with numbered node of two types

The *graph traversal* defined in 1.2.1 and especially its conditional and generative extensions defined in Definition 1.2.2 and 1.2.3 are important operations in this work. They are employed in the first phase of the algorithm, for copying the dependency graph as constituency graph described in Chapter **??**.

Definition 1.2.1 (Traversal)**.** Traversal $t(V_S, G)$ of a graph G starting from node V_S is a recursive operation that returns a set of sequentially visited nodes neighbouring each other in either *breadth first* (t_{BF}) or *width first* (t_{WF}) orders.

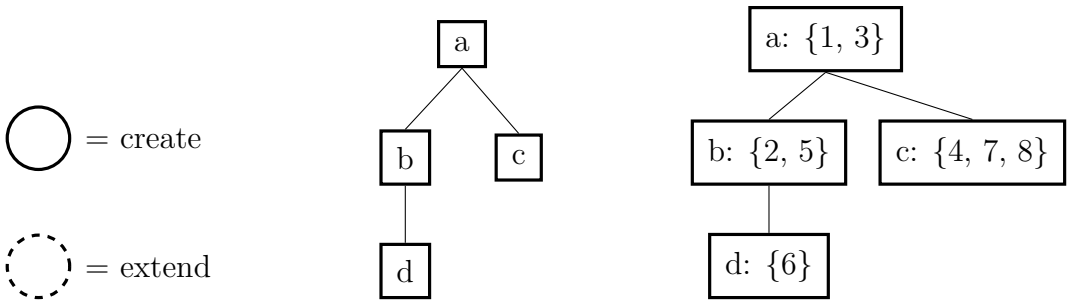
The graph traversal is employed either for searching for a node or an edge or finding a sub-graph that fulfils certain conditions on its nodes and edges if it is a conditional traversal. For example in the semantic enrichment phase (that will be described in Section **??**), to ensure that the semantic patterns are applied iteratively to each clause, from a multi-clause CG graphs are selected all clause sub-graphs without including the embedded (dependent) clauses.

Definition 1.2.2 (Conditional Traversal)**.** Conditional traversal $t(F_V, F_E, V_S, G)$ of the graph G starting from node V_S under node conditions F_V and edge conditions F_E is a traversal operation where a node is visited if and only if its feature structure conditionally fulfils the F_V and the edge that leads to this node conditionally fulfils the F_E.

One of the potential complete traversals for the graph from Figure 1.4 starting from the node 1 is {1, 2, 3, 4, 5, 6, 7, 8} using breadth first strategy or {1, 2, 5, 6, 3, 4, 7, 8} for depth first strategy. On the other hand, a conditional traversal of non-dashed nodes staring from the node 1 results in {1, 2, 4, 6}, {1, 4, 2, 6} or {1, 2, 6, 4}. The first two traversals corresponding to the width first strategy and the third one to the depth first strategy.

I also use the graph traversal to execute generative operations on a parallel graph, which is a special case of *graph rewriting*. For example DG traversal is employed for bootstrapping (i.e. creating in parallel) the CG as it was previously motivated in Section **??**.

Definition 1.2.3 (Generative Traversal)**.** Generative traversal $g(M,G)$ of a source graph G via a operation matrix M is an operation resulting in the creation of the target graph H by contextually applying generative operations to bring the latter into existence. The operation matrix M is a set of tuples (ctx,o,p) that link the visited source node context ctx (as features of the node, the edge and previously visited neighbour) to a certain operation o that shall is executed on the target graph H with parameters p.



(a) The rule set (b) The target (c) The explained target

Fig. 1.5 The generative traversal result for Figure 1.4 using create and extend operations

Next I provide an rough description of what happens when a generative traversal is executed and the exact algorithm will be described in detail in Section **??**. For example lets assume that only two types of operation are needed for our task at hand. First is to create a new node on the target graph once a non-dashed node is visited on the source graph. And, second, is to pass without doing anything the dashed nodes. This is schematically represented in Figure 1.5a. Lets now apply these operations on traversing the example graph using breadth first strategy following the order provided above {1, 2, 3, 4, 5, 6, 7, 8}. The traversal graph is considered source graph and the target graph is empty at the beginning of the process. Upon visiting the node 1 a first node is created on the target graph which is labelled *a*. When traversing nodes 2 and 4 then each of them signal creation of the nodes *b* and *c* as children of *a* in the target graph and correspondingly node 6 signals creation of node *d*. The final target graph is depicted in Figure 1.5b and in Figure 1.5c the source nodes are embedded into the

target node to make explicit that the non-dashed nodes (i.e. 3, 5, 7, 8) are simply passed over without any generative operation.

Now that generative traversal is defined, by analogy, *update*, *insert* and *delete* traversals can be defined on the source or target graph by using the same mechanism of *operation matrices* mapping contexts of visited nodes and edges to update, insert and delete operations. In this work, however, these operations are not used and therefore omitted here.

In Section 1.1 the last two definitions were for the constituency and dependency graphs. They are used in this thesis to represent grammatical analysis of of a sentence. Next we will look at a special type of graph which represents fragments of structure repeatable across multiple analysis. They represent generalisations or patterns that usually are associated with grammatical features or a set of features which I explain in Chapter **??**. These graphs are called *pattern graphs* and the next section is dedicated to them.

1.3 Pattern graphs

Regardless of the type, constituency or dependency, the parsing process (which will be described in Chapter **??**) relies on identifying patterns in graphs. The patterning is described as both graph structure and feature presence (or absence) in the nodes or edges. The *pattern graphs* (defined in 1.3.1) are special kinds of graphs meant to represent small (repeatable) parts of parse graphs that, in the context of the current work, are used to identify grammatical features.

The pattern graphs contrast with the *parse* (or *instance*) graphs which are either constituency or dependency graphs. The parse graphs express what is an actual analysis of a text, i.e. representing what is the case, whereas the pattern graph expresses a potential that could be the case in the instance graph. This way the pattern graphs have a prescriptive interpretation whereas the instance ones taken a factual interpretation.

Definition 1.3.1 (Pattern Graph)**.** A *pattern graph* (PG) is a feature rich graph for expressing regularities in the node and edge structure.

Next I discuss two example of pattern graphs. One example shows a pattern graph encoding the present perfect continuous tense, which traditional grammar defines as in Table 1.1. Afterwards, the second example will show how the notion of linear succession among nodes is accounted for in the pattern graphs for declarative and interrogative mood.

has/have	+	*been*	+	*Vb-ing*
to have, present simple		to be, past participle		verb, present participle

Table 1.1 Present perfect continuous tense

Examples 1–3 show variations of present perfect continuous tense in a simple clause according to declarative and interrogative mood and the "has" contraction. Of course there are more variations possible for example according to voice (active and passive) but they are omitted here because they adds combinatorially to the number of examples and the ones provided already serve their purpose here. The Figures 1.6-1.8 represent corresponding dependency parses for these examples (generated with Stanford dependency parser).

(1) He has been reading a text.

(2) He's been reading a text.

(3) Has he been reading a text?

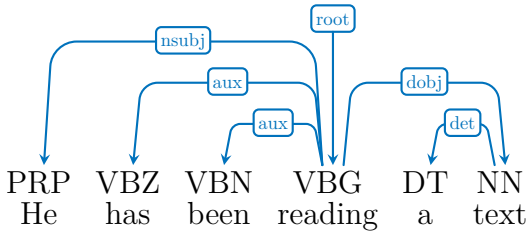


Fig. 1.6 Present perfect continuous: indicative mood, un-contracted "has"

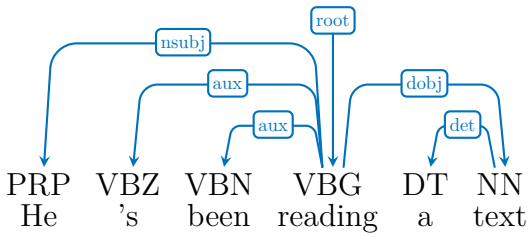


Fig. 1.7 Present perfect continuous: indicative mood, contracted "has"

The present perfect continuous tense can be formulated as a pattern graph (including voice) over the dependency structure as illustrated in Figure 1.9. In this pattern the main lexical verb is *present participle* indicated via the *VBG* part of speech. It is accompanied by two auxiliary verbs: *to be* in *past participle* (*VBN*) form and *to have* in *present simple* form specified by either *VBZ* for 3rd person or *VBP* for non-3rd

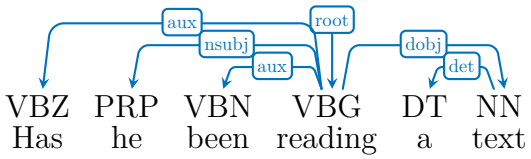


Fig. 1.8 Present perfect continuous: interrogative mood, un-contracted "has"

person. Also the *to be* can be either connected by the *aux* relation or in case of passive form by the *auxpass* relation. Note that the pattern in Figure 1.9 constraints the edge type (using an OR-set) to the verb *to be* which can be either *aux* or *auxpass* and the part of speech of the verb *to have* which can be *VBZ* or *VBP*.

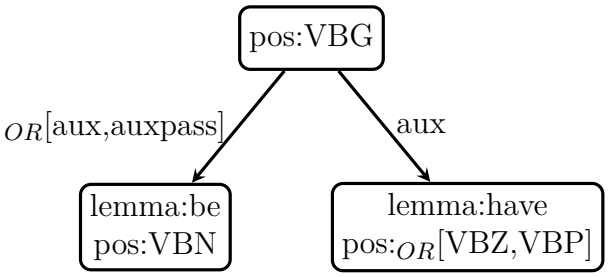


Fig. 1.9 The graph pattern capturing features of the present perfect continuous tense

One of the fundamental features of language is its sequentiality and directionality. This aspect is not inherent in graphs. In the simplest form, they just describe connections between nodes and are agnostic to any meaning or interpretation. Next, I introduce the way I deal with linear order in the pattern graphs.

Lets consider the clause mood and encode the distinction between *declarative* and *Yes/No interrogative* moods. In SFG this features is described in terms of the relative order of clause elements. If the finite is before the subject then the mood is Yes/No-interrogative, whereas when the finite succeeds subject then the mood is declarative. Example 3 contrasts in mood with 1 and 2.

Order can be specified in absolute or relative terms and partially or exhaustively. In order to cover these three kinds of constraints, I introduce three special features: the node *id*, *precede* and *position*. Node id takes a token to uniquely identify a node in the graph, the precede feature takes an ordered set to indicate the (partial) linear precedence to other node ids, and the position feature indicates the absolute position of a node.

One way to introduce order among nodes is then by marking them with an absolute position. This is a good method applicable to parse graphs. The DGs and CGs, are automatically assigned at creation time the absolute position of the node in the

sentence text via the feature *position*. This feature is present in the leaf nodes only and corresponds to the order number in which they occur in the sentence text while the non-leaf node's position is considered to be the lowest position of its constituent nodes. The absolute position description is rarely used in the PGs. The only cases are to state that the constituent is first or last position in a sentence.

Another way to specify node order is through relative precedence, for which the node id and the precedence features are introduced above. This is the preferred method to provide linear precedence dimension in pattern graphs. It is also relative so the specification can be partial. With this method a node specifies that it precedes a set of other nodes.

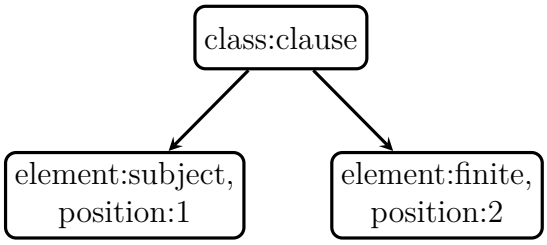


Fig. 1.10 Declarative mood pattern graph with absolute element order

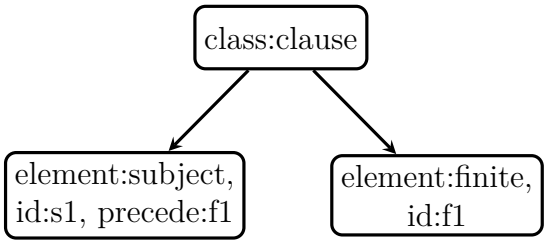


Fig. 1.11 Declarative mood pattern graph with relative element order

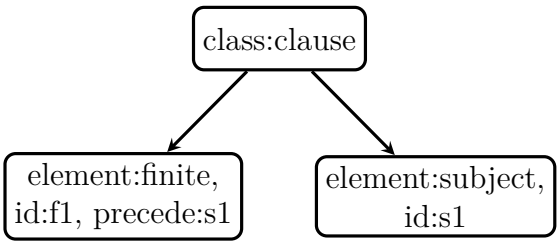


Fig. 1.12 Pattern graph for Yes/No interrogative mood

To continue the example of mood features, I illustrate in Figures 1.10 and 1.11 the use of relative and absolute node ordering constraints for declarative mood; and in

Figure 1.12, I depict the PG for the Yes/No interrogative mood. In both cases I use the relative node ordering.

Patterns like the ones explained above can be created for a wide range of grammatical features. Once the grammatical feature is encoded as a pattern graph it can be identified in parse graphs (DG or CG) via *graph pattern matching* operation described in Section 1.4. Moreover, once the pattern is identified it can act as a triggering condition for the node update operation. When the pattern is matched then additional features can be added to the nodes of the parse graph thus the mechanism of enriching the parse graph. Coming back to out tense example above, once the pattern 1.9 is identified then the clause can be marked with the tense feature. In the next section I address the graph matching operation and then show how it works using pattern graphs.

1.4 Graph matching

So far we have discussed about constituency and dependency graphs and, in the last section, I introduced pattern graphs. The intuition behind pattern graphs is that they are meant to be matched against or found in other graphs. The patterns graphs can be viewed as small reusable modules and as generalizations consisting of structural patterns. I will address next what does it mean for two graphs to be the same, i.e. *isomorphic* and how this sameness checking is used in the current work.

In *mathematics* the structure-preserving mappings $f : X \to Y$ (from one object X to the other Y) of the same type is called *morphism*. The morphism $f : X \to Y$ is called *isomorphism* if there exists an inverse morphism $g : Y \to X$ such that $f \circ g = id_X$ and $g \circ f = id_X$.

In computer science, the *graph matching* operation is known (sub-)graph isomorphism. Two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are isomorphic if there exists a mapping from the nodes of graph G to the nodes of graph H, such that the edge neighbourhood is preserved. In such context the nodes are unique atomic symbols functioning as node identifiers.

Definition 1.4.1 (Graph matching)**.** For two graphs G and H, where $G \leq H$, *graph matching* is the operation of finding an isomorphism between G and H.

Definition 1.4.2 (Graph isomorphism)**.** An isomorphism of graph $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is a bijective function $f : V_G \to V_H$ such that if any two nodes $u, v \in V_G$ from G are adjacent $(u, v) \in E_G$ then $f(u), f(v)$ are adjacent in H as well $(f(u), f(v)) \in E_H$.

Graphs do not always have the same number of nodes (or edges). We say that a graph is smaller than another one, denoted $G \leq H$, when its number of nodes is less than that of the other graph. In such cases, for two graphs G and H where $G < H$, the (sub-)graph matching task is redefined to establishing isomorphism(s) from G to a sub-graph of H.

Definition 1.4.3 (Sub-graph isomorphism)**.** Given two feature rich graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, G is sub-graph isomorphic to G (denoted $G \subseteq H$) if there is an injective function $f : V_G \to V_H$ such that

- $\forall v \in V_G, f(v) \in V_H$ and
- any two nodes adjacent in G, $(u, v) \in E_G$, are also adjacent in H, $(f(u), f(v)) \in E_H$

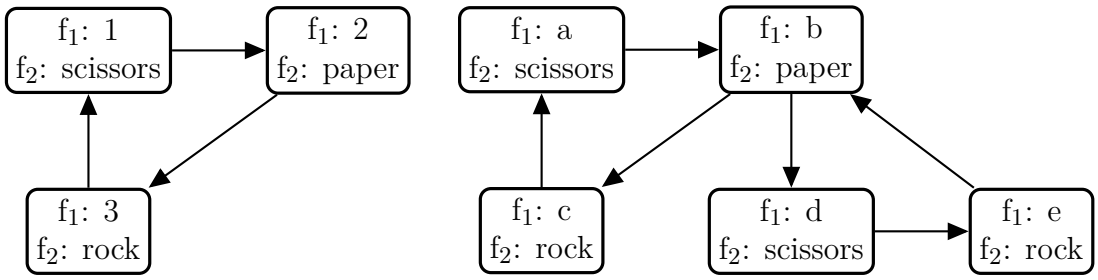
(a) The pattern graph (b) The target graph

Fig. 1.13 Sub-graph isomorphism {1=a, 2=b, 3=c}

In Figure 1.13a is depicted a labelled graph that is isomorphic to a sub-graph in Figure 1.13b. The example graphs presented in Figure 1.13 have atomic labels as nodes and the isomorphism is established as a mapping between labels. Section 1.1 above mentions that the graph considered in this thesis have feature structures as their nodes and not atomic nodes. But in case of feature rich graphs additional rules how to establish the isomorphism need to be provided because there are multiple ways to approaching it.

Lets look at Figure 1.14 where the graph nodes are feature structures using features: f_1 and f_2. One way to approach isomorphism in this scenario is by the value of one feature, for example f_1. Then we can identify two sub-graph isomorphisms: {1=a, 2=b, 3=c} and {1=b, 2=d, 3=e}. This approach, besides additional specification what values to compare, i.e. f_1s, is the same as providing a sub-graph isomorphism on the labelled graphs from Figure 1.13.

In addition to the rule above, lets add a constraint that the isomorphism is not only a mapping between the feature values (numbers to letters) but also that the mapped values are identical (strict value equality). If we consider the strict equality rule applied



(a) The pattern graph (b) The target graph

Fig. 1.14 An example of rich sub-graph isomorphism

on f_1 feature, there is no isomorphism between the two graphs because first one uses numbers {1, 2, 3} and the second uses letters{a, b, c, d}. Now if we turn to the values of f_2 and apply the same rule then there is one isomorphism possible {paper=paper, rock=rock, scissors=scissors}. The second one, even if it is a cycle, {paper=paper, rock=scissors, scissors=rock} is no longer acceptable because the "scissors" and "rock" switched places in the target graph and it would have been acceptable as a mapping, but not as strict value equality. Formally, the additional equality constraint can be expressed on the graph isomorphism f as $u = f(u)$.

This brings us to the idea that, in the feature rich (sub-)graph isomorphism, we need to introduce a *matching* operator (denoted $\succcurlyeq$) for nodes. This means that we no longer can use atomic symbol mapping but have to employ the matching operator. The node matching operation is defined on feature structures. We say that a feature structure may match once, several times or not at all another feature structure, $FS_1 \succcurlyeq FS_2$. This intuition is expressed in Definition 1.4.6. The sub-graph isomorphism over the feature rich graphs is captured in Definition 1.4.4 below.

Definition 1.4.4 (Rich sub-graph isomorphism)**.** Given two feature rich graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ and a matching relation $\succcurlyeq$, G is sub-graph isomorphic to H if there is an injective mapping $f : V_G \to V_H$ such that

- each node in V is mapped to exactly one node in H, $\forall v \in V_G, f(v) \in V_H$ and
- each node in G matches with its correspondent in H, $\forall v \in V_G, v \succcurlyeq f(v)$ and
- any two nodes which are adjacent in G, are also adjacent in H, $\forall (u,v) \in E_G, \exists\Big(f(u), f(v)\Big) \in E_H$

We already mentioned in the introduction of this section that pattern graphs are considered as generalizations over parse graphs (constituency or dependency). The

pattern graphs are smaller and their nodes have fewer features specified. The parse graphs have more nodes and features. We say that a parse graph *instantiates* a pattern graph if there exists a rich sub-graph isomorphism. This operation is called *pattern graph matching* and is defined in Definition 1.4.6 below.

Definition 1.4.5 (Pattern graph matching)**.** Given a pattern graph G and an instance (parse) graph H (either dependency or constituency), *pattern graph matching* is the operation of finding a rich sub-graph isomorphism from G to H such that each pattern node matches its corresponded instance node each H, $\forall v \in V_G, v \succcurlyeq f(v)$.

As mentioned before, nodes of the parse and the pattern graphs are feature structures. I approach the matching between them in two steps: first, matching the feature names in Definition 1.4.6, and second, matching the feature values in Table 1.2. In order to simplify and make explanations clear, I will further refer to the feature structures that constitute nodes in the pattern graphs as *pattern feature structures* and the feature structures that constitute nodes in the instance graphs as *instance feature structure*.

Definition 1.4.6 (Feature structure matching)**.** A pattern feature structure V matches an instance feature structure U if and only if every feature in V has a corresponding feature U and their values match; that is $\forall f_V \in V, \exists f_U$ such that $name(f_V) = name(f_U)$ and $val(f_V) \succcurlyeq val(f_U)$.

According to the Definition 1.1.3, the values of feature structures can be either atomic (numbers, strings, symbols, etc.), denoted *atomic*, or one of the conjunction sets: S_{AND}, S_{OR}, S_{XOR} and S_{NAND}. For simplicity, the option of nested feature structures is excluded in the current work even though it is perfectly viable configuration. Consequently, the matching relation takes into consideration the *type* of the compared elements in addition to how they relate to each other, including comparisons between set and atomic values. Please note that this relation is *not symmetric* i.e. *not commutative* because the subsequent relations used in the definition i.e. set inclusion and set subsumption are not symmetric. This means that $A \succcurlyeq B \neq B \succcurlyeq A$.

I will remind here that, the function $\tau(S)$, defined in Section 1.1, returns the *type* of the feature value. I extend its definition here to handle also atomic data types as follows $\tau : x \rightarrow \{atomic, S_{AND}, S_{OR}, S_{XOR}, S_{NAND}\}$. The matching rules have to be defined for each possible pair of types returned by the function τ yielding 25 possibilities. Table 1.2 provides matching relations for each pair of types where the rows represent value types of the pattern features, denoted $\tau(v)$, and the columns represent value types of the instance features, denoted $\tau(u)$. Each table cell contains a comparison expression

returning a truth value. Cells with a bottom sign $\perp$ mean that there can be no match between these types no matter the values.

$\tau(v) \backslash \tau(u)$	*atomic*	S_{AND}	S_{OR}	S_{XOR}	S_{NAND}
atomic	$v = u$	$v \in u$	$\perp$	$\perp$	$v \notin u$
S_{AND}	$\perp$	$v \subseteq u$	$\perp$	$\perp$	$\perp$
S_{OR}	$v \ni u$	$v \cap u \neq \varnothing$	$v \supseteq u$	$v \supseteq u$	$\perp$
S_{XOR}	$v \ni u$	$\perp$	$\perp$	$v \supseteq u$	$\perp$
S_{NAND}	$v \not\ni u$	$v \cap u = \varnothing$	$v \cap u = \varnothing$	$v \cap u = \varnothing$	$v \subseteq u$

Table 1.2 Strict matching between pattern and instance feature values organised by value type

For example, if both values are of atomic type then in order to match they have to equal. If the $\tau(v)$ is atomic and the $\tau(u)$ is an AND set then v needs to be among the set of values constituting u; whereas if the $\tau(u)$ is an OR or XOR set then these values do not match, designated by the bottom sign $\perp$. The same reading applies to the rest of the table for each pair of value types.

A more permissive matching is defined in Table 1.3. Here, on the instance feature values, the two types of disjunction (OR and XOR) and the negated conjunction (NAND) are interpreted as possibly matching and provided with the corresponding relation whereas in the previous definition these cases were completely excluded. The permissive match is rarely used in this work but it is nevertheless useful cases of instance graphs where the feature values could not be are assigned with a certainty but as a disjunction of either one or the other.

$\tau(v) \backslash \tau(u)$	*atomic*	S_{AND}	S_{OR}	S_{XOR}	S_{NAND}
atomic	$v = u$	$v \in u$	$v \in u$	$v \in u$	$v \notin u$
S_{AND}	$\perp$	$v \subseteq u$	$v \subseteq u$	$v \subseteq u$	$v \cap u = \varnothing$
S_{OR}	$v \ni u$	$v \cap u \neq \varnothing$	$v \supseteq u$	$v \supseteq u$	$v \backslash u \neq \varnothing$
S_{XOR}	$v \ni u$	$\perp$	$\perp$	$v \supseteq u$	$\lvert v \backslash u \rvert = 1$
S_{NAND}	$v \not\ni u$	$v \cap u = \varnothing$	$v \cap u = \varnothing$	$v \cap u = \varnothing$	$v \subseteq u$

Table 1.3 Permissive matching between pattern and instance feature values organised by value type

The permissive FS matching has been developed together with the strict FS matching as it was not clear in the beginning which one is suitable for the current work. After conducting several tests it became clear that strict matching is the one yielding the better results because it yields fewer matches. This is especially visible in the cases of Transitivity enrichment when too many incorrect patterns are qualified by

the permissive operator. Now that pattern graph matching is explained, lets take a look next at how it is used to perform operation on instance graphs.

1.5 Pattern based operations

The patterns are searched for in a graph always for a purpose. Graph isomorphism is only a precondition for another operation, be it a simple selection (i.e. non-affecting operation) or an affecting operation such as feature structure enrichment (on either nodes or edges), inserting or deleting a node or drawing a new connection between nodes. It seems only natural that the end goal is embedded into the pattern, so that when it is identified, also the desired operation(s) is(are) triggered. I call such graph patterns *operational pattern graphs* (Definition 1.5.1). Next I explain how to embed the operations into the graph pattern and how they are used in the algorithm.

Definition 1.5.1 (Operational graph pattern)**.** An *operative graph pattern* is a pattern graph that has least on one node *operation* and *arg* features.

The operational aspect of the pattern graph is specified in the node FS via three special features: *id*, *operation* and *arg*. The *id* feature (the same as for relative node ordering) is used to mark the node for further referencing as an argument of an operation, the *operation* feature names the function to be executed once the pattern is identified and the *arg* feature specifies the function arguments if any required and they are tightly coupled with function implementation. If a node has the feature operation then it is called *operational node*. Also, in the current implementation, the special features such as operation, arg, id, precede etc. are excluded from the pattern matching operation because they have functional role linked to the implementation and do not describe the linguistic properties of a graph node.

So far the implemented operations are *insert*, which is used for creation of empty nodes, *delete*, which is used for corrections of predictable errors in dependency graphs and *update* operation, which is the main mechanism behind graph enrichment. These operations are depicted in Figure **??** of the parser pipeline architecture from Section **??**.

Operative patterns are enacted once they are matched to an instance graph. An operative pattern graph G is *enacted* on an instance graph H, in two steps. First, the pattern graph is strictly matched to instance graph. If an isomorphism f is found then, second, for every operational node $v \in G, \exists att(v) = operation$, the specified operation $op = val(v.operation)$ and the corresponding node of the instance graph $u \in H$, the

operation is executed on the node of the instance graph $op(u)$. If the arg feature is provided then the operation is executed with that additional argument.

1.5.1 Pattern based node update

As mentioned above the pattern based node update is used for adding onto the constituency graph new features. Lets consider Example 4 whose constituency graph is provided in Figure 1.15 and the task to assign *agent* feature to the subject node and *affected-possessed* feature to the complement. This can be done using the pattern graph matching with a feature update operation indicated on subject and complement nodes. The PG depicted in Figure 1.16 fulfils this purpose because it matches constituency graph from Figure 1.15 and has the corresponding update operations indicated.

(4) He gave the cake away.

(5) He gave her the cake.

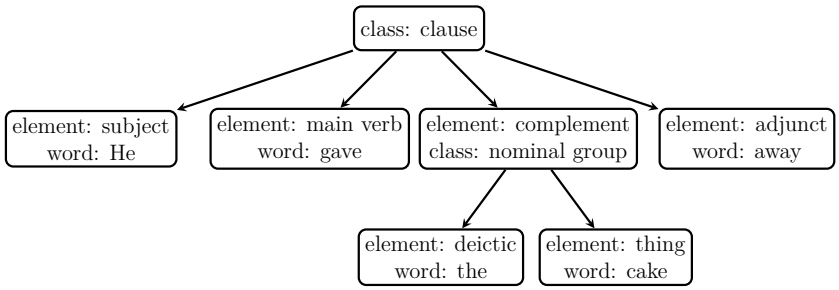


Fig. 1.15 Constituency graph corresponding to Example 4

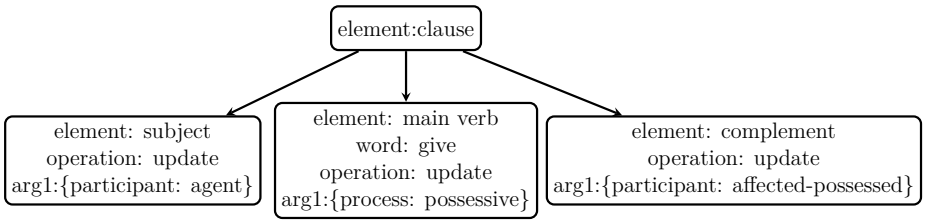


Fig. 1.16 A graph pattern for inserting agent and affected-possessed participant roles

Consider the same pattern, but applied to a sentence in the Table 1.4. The clause has two complements and they are by no means distinguished in the pattern graph. When such cases are encountered the PG yields two matches, (each with another complement) and the update operation is executed to both of the complements. To

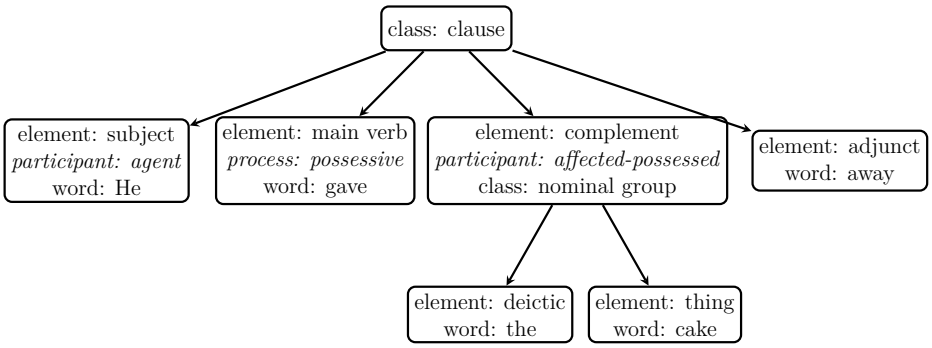


Fig. 1.17 The resulting constituency graph enriched with participant roles

overcome such cases from happening PG allow defining *negative nodes*, meaning that those are nodes that shall be missing in the target graph.

For example to solve previous case I define the PG depicted in figure 1.18 whose second complement is a negative node and it is marked with dashed line. This pattern is matched only against clauses with exactly one complement leaving aside the di-transitive ones because of the second complement.

class:clause				
element:subject	element: main verb	element:complement	element:complement	
He	gave	her	the	cake.

Table 1.4 CG with a di-transitive verb

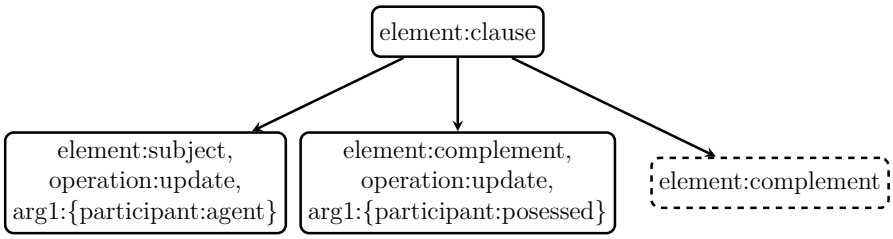


Fig. 1.18 PG for inserting agent and possessed participant roles to subject and complement nodes only if there is no second complement.

The current implementation of matching the patterns that contain negative nodes is performed in two steps. First the matching is performed with the PG without the negative nodes and in case of success another matching is attempted with the negative nodes included. If the second time the matching yields success then the whole matching process is unsuccessful but if the second phase fails then the whole matching process is successful because no configuration with negative nodes is detected.

For the sake of explanation I call the pattern graph with all the nodes (turned positive) *big* and the pattern graph without the nodes marked negative *small*. So then, matching a pattern with negative nodes means that matching the *big* pattern (with negative nodes turned into positive) shall fail while matching the *small* one (without the negative nodes) shall yield success.

1.5.2 Pattern based node insertion

In English language there are cases when an constituent is missing because it is implied by the (grammatical) context. These are the cases of Null Elements treated in the Chapter **??**.

(6) Albert asked [∅ to go alone].

Consider the Example 6. There are two clauses: first in which Albert asks something and the second where he goes alone. So it is Albert that goes alone, however it is not made explicit through a subject constituent in the second clause. Such implied elements are called *null or empty constituents* discussed in detail in the Section **??**. The table 1.5 provides a constituency analysis for the example and the null elements (in italic) are appended for the explicit grammatical account. In the Section **??** I offer the grammatical account of the graph patterns that insert these null elements into the parse graphs (so in fact extensively using the pattern based node insertion treated here).

<table>
<tr><td colspan="5">class:clause</td></tr>
<tr><td rowspan="2">element: subject</td><td rowspan="2">element: main verb</td><td colspan="3">element: complement, class:clause</td></tr>
<tr><td>element: subject</td><td colspan="2">element: main verb</td><td>element: adjunct</td></tr>
<tr><td>Albert</td><td>asked</td><td>Albert</td><td>to</td><td>go</td><td>alone.</td></tr>
</table>

Table 1.5 The constituency analysis that takes null elements into consideration

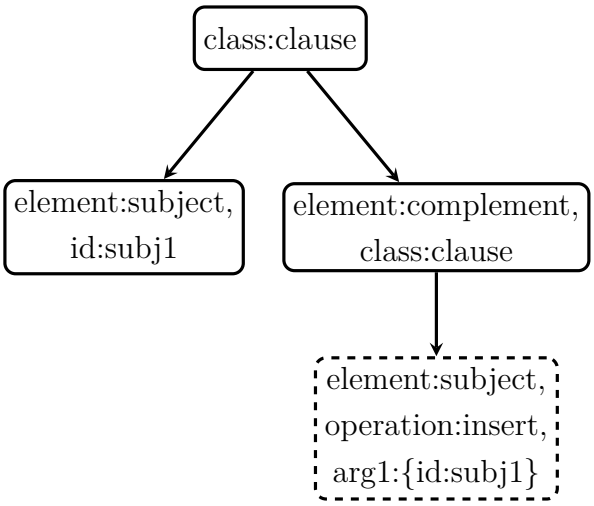


Fig. 1.19 A graph pattern to insert a reference node

To insert a new node the, PG needs to specify that (1) the inserted node does not already exist, so it is marked as negative node, (2) specify *operation:insert* in the FS of the same and (3) provide id of the referenced node as FS argument (arg1) if one shall be taken.

In operational terms, the insertion operation means that the whole pattern will first go through a matching process. If there is a match then the new node is created. A peculiar thing about the created node is that it may keep a reference to another node or not. In our example it does keep a reference to the subject of dominant clause. If so, then all the features of the referee node are inherited by the new node. And if any are additionally provided then the new node overrides the inherited ones.

This section concludes our journey in the world of graph patterns, isomorphisms and graph based operations. Leaving only one more important data structure to cover: the system networks.

1.6 Systems and Systemic Networks

In the Section **??** I presented the basic definition of *system* and *system network* and the notations as formulated in the SF theory of grammar. In this work the system networks are simplified to hierarchies of disjoint features maintaining the entry conditions. This corresponds to the type logic component described in (O'Donnell 1993) where the syntagmatic organisation is restricted to a single functional layer. The reason behind this simplification is because the hierarchy of disjoint feature structures are perfectly suitable to correctly derive the complete set of parent features from a one or several

manual selections. Moreover the systemic networks are not interconnected into an uniform grammar but separate modules describing selected aspects of language. Once the complete set of features is derived then it is associated with a graph pattern. This pattern graph is then used in the parse graph enrichment process described in Section **??**. Next I define the necessary concepts serving the current work.

First I would like to introduce abstract concept of *hierarchy* defined in a computer science fashion by Pollard & Sag (1987: 30) which is a formal rephrasing of Definition **??** stating that the hierarchy is a sort of logical precedence among the terms of a system.

Definition 1.6.1 (Hierarchy)**.** A hierarchy is finite bounded complete partial order $(\Delta, \prec)$.

The next concept that requires closer formalization is that of a system first established in Definition **??**. For precision purposes, this has a narrower scope without considering the system networks or precondition constraints; these are introduced shortly afterwards building upon current one.

Definition 1.6.2 (System)**.** A *system* $\Sigma = (l, P, C)$ labelled l is defined by a finite disjoint set of distinct and mutually defining terms called a *choice set* C (of type S_{XOR}) and an *entry condition set* P (of type S_{OR}, S_{XOR} or S_{AND}) establishing the delicacy relations within a system network.

There is a set of functions defined that apply to systems: $label(\Sigma) = l$ is a function returning the system name, $choices(\Sigma) = C$ is a function returning the choice set, $precondition(\Sigma) = P$ is a function returning the set of entry condition features, and the $size(\Sigma)$ return the number of elements in the system choice set.

Definition 1.6.3 (Systemic delicacy)**.** We say that a system S_1 is more delicate than S_2 denoted as $S_1 \prec S_2$ if

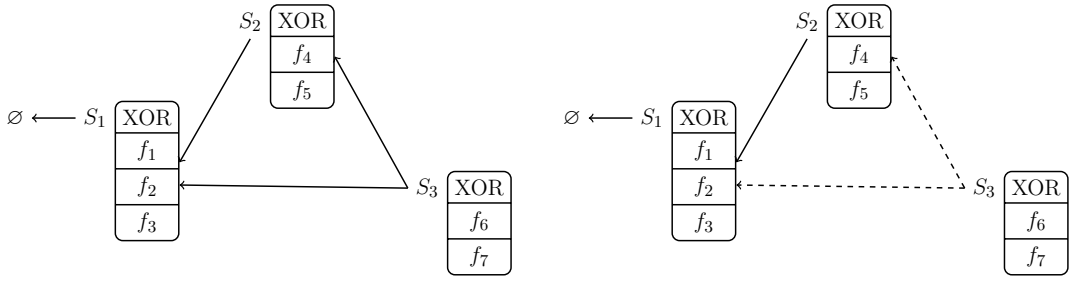
1. both system belong to the same system network: $S_1, S_2 \in SN$
2. there is at least a feature but not all of S_1 which belong to the entry condition of S_2 i.e. $choices(S_1) \in precondition(S_2)$ or
3. there is another system S_3 that has a among the entry conditions a feature of S_1 and whose features are among the entry conditions of S_2 i.e. $\exists S_1' \in SN$ such that $choices(S_1) \in precondition(S_3) \wedge choices(S_1') \in precondition(S_2)$

Systems are never used in isolation. SF grammars often are vast networks of interconnected systems defined as follows.

Definition 1.6.4 (System Network)**.** A *system network* $SN = (r, \Gamma)$ is defined as a hierarchy over a set of systems Γ where the order is that of systemic delicacy starting from a root feature r such that for any system in the network either there is another less delicate system or its entry condition is the root feature i.e. $\forall S_i \in \Gamma, \exists S_j \in \Gamma | S_j \prec S_i \vee precondition(S_i) = r$.

In a systemic network SN where a system S_l depends on the choices in another system S_e (i.e. the preconditions of S_l are features of S_e) we call the S_e an *early(older) system* and the S_l a *late(younger) system*. This is just another way to refer to order systems according to their delicacy but applying this ordering to execution of systemic selection.

The graphical notation of system networks has been introduced in Section **??**. Considering the above definitions, the system network can be represented as a graph where each node is a system features and edges represent precondition dependencies. Figure 1.20 depicts examples of a system network with three systems. In Figure 1.20a the entry conditions are S_AND sets only and in Figure 1.20b the entry condition for S_3 is $OR(f_2, f_4)$ depicted with dashed lines. In such a graph, all system features must be unique i.e. $\forall S_1, S_2 \in SN : choices(S_1) \cap choices(S_2) = \varnothing$ and there must be no dependency loops.



(a) System Network with conjunctive entry conditions

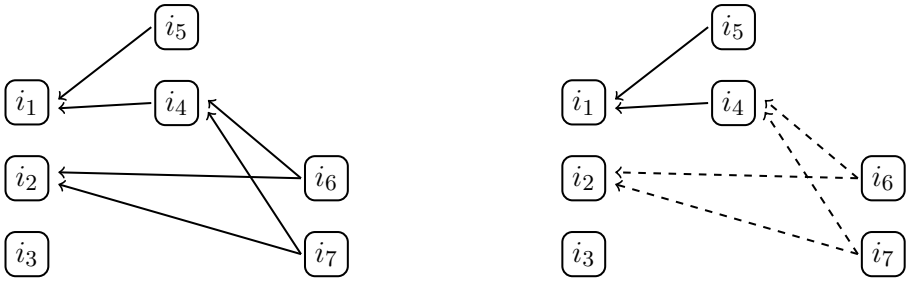
(b) System Network with disjunctive entry condition for S_3

Fig. 1.20 Example System Network presented as graphs

For a chosen feature in the system network it is possible to trace a path to the root feature by traversing systems through their preconditions. Generating such a path is equivalent to preselecting the features in all earlier systems. This is needed for assigning a complete set of feature selections to a pattern graph before it is used in the parse graph enrichment. Conversely, in the verification process it is necessary to check

whether a set of arbitrary features belong to a *consistent* and *complete selection path*. Next are addressed the concepts needed for addressing this task and how it is executed.

The system networks from Figure 1.20 can be unpacked into a *feature network* (Definition 1.6.5 which is referred to as *maximal selection graph*) interconnected by system entry conditions. In Figure 1.21 are depicted two feature networks corresponding to the system networks above. The dashed lines mean disjunction and continuous ones mean conjunction of entry features.



(a) Feature Network graph with conjunctive entry conditions only

(b) Feature Network graph with some disjunctive entry condition

Fig. 1.21 Example Feature Network graphs

The feature network can be defined in relation to the system network as follows.

Definition 1.6.5 (Feature Network)**.** For a given system network $SN = (r, \Gamma)$, a *Feature Network* $FN(N, E)$ is a directed graph whose nodes N are the union of choice sets of the systems Γ and the edges E connect choice features with the entry condition (precondition) features of the systems Γ.

Given a feature network $FN(N, E)$ and a feature the network $f \in N$ a *selection path* $SP(N, E)$ is a connected sub-graph of traversal paths between the root of the feature network and the feature f. A *complete selection path* is a selection path from one of the leaf nodes up to the network root.

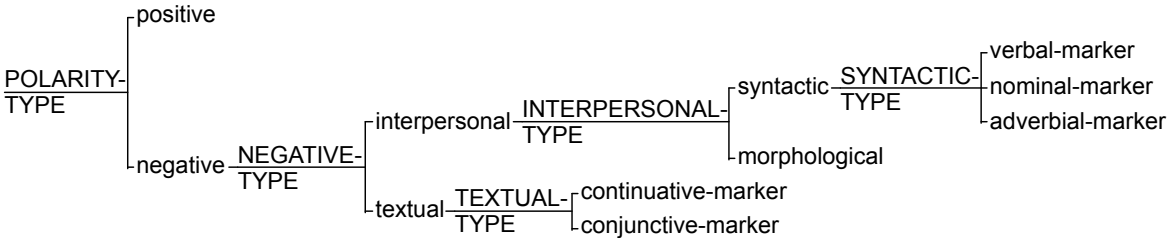


Fig. 1.22 Polarity System

The selection path is generated by traversal of the feature network from a given feature node towards the rood node. If the node has no incoming edge then the result

traversal it is a leaf node and the resulting selection path is a complete one. For example if *verbal-marker* feature is selected in the system network depicted in Figure 1.22 the traversal of the corresponding feature network towards root feature yields the selection path *negative – interpersonal – syntactic – verbal-marker*. This path is also complete with respect to the network because there are no systems younger than the SYNTACTIC-TYPE system.

When the system networks define more than one feature in the precondition it leads to split paths it means that there is more than one feature needs to or can be preselected. If the edge is a continuous line (depicted in Figure 1.21a) the both variants are part of the same selection path. In case the edge is a dashed line (depicted in Figure 1.21b) the paths are considered alternative and further decision making mechanism shall be employed to reduce the disjunction to a single variant. In the current work no such mechanism is employed and the parse result is presented with both alternatives. In most cases the entry condition is constituted of a single feature, when there are multiple ones, usually they are conjuncted and only in a small minority of cases entry condition is a disjunction of features.

Algorithm 1: Naive backwards induction of a selection path

```
   input  : feature, system network
1  begin
2      add feature to empty selection path
3      for system in traversal path to the root of system network:
4          get entry condition features of system
5          add entry condition features to the selection path
6      return selection path
7  end
```

The pseudo code for creating a selection path as described above is outlined in Algorithm 1. Given a system network a random non root feature belonging to the network, it traverses the networks, one by one, towards the root and collects the entry conditions of each system into a selection path.

1.7 On realisation rules

The previous section explains that in the current work systemic networks are simplified to a taxonomy of features and no realisation rules are considered. This section explains how the pattern graphs are a substitute to the realisation rules and how they relate to each other.

The *realisation rule* of a systemic feature specifies how that feature is realised as a syntagmatic structure down to a text form using operations such as insert or expand constituent, order two constituent, preselect another feature, lexicalise etc. They are the essential ingredient binding the paradigmatic description into a syntagmatic structure. Robin Fawcett recurrently emphasises the role of realization rules in the composition of system networks. He often stresses "no system networks without realization rules". It is the *instantiation* process that in Halliday's words "is the relation between a semiotic system and the *observable* events or 'acts' of meaning" (Halliday 2003: emphasis added). The realisation rules for a systemic feature are the statement of operations through which that feature contributes to the structural configuration (that is being either generated or recognised) (Fawcett 2000: p.86).

It is not easy however for linguists and grammarians to provide such statements for the systemic features. Doing so means an explicit formalisation of grammar on top of charting the systemic composition and dependencies which is already a challenging task in its own. Not to mention that writing a good grammar is already a very difficult task. The realisation rules most of the time remain in the minds of the interpreters who can recognise a feature when it occurs. Adding the formal specification of the realisation rule requires tools for consistency checking with respect to the rest of the grammar and large corpus query tool to test various rule hypotheses which are not really available yet.

In this work the graph patterns can be considered as bundles of realisation rules and feature selections and therefore an approach to replace the realisation rules. This idea is implicitly covered in Section 1.3 and here I explicitly describe it through an example. Consider the fragment of the Mood system network from Halliday & Matthiessen (2013: 162) depicted in Figure 1.23. This example aims at three feature selections: major, indicative and declarative. The root feature in the system network is realised through a constituent *clause*, any of the selected fratures is ascribed directly to it, and the selection of any subsequent features impacts the elements below through the associated realisation rules. The pattern graph that captures selection of major feature is depicted in Figure 1.24.

In Figure 1.25 is depicted the pattern graph corresponding to the selection of indicative feature. Here the clause constituent receives the whole selection path up to the root feature *clause – major – free – indicative* and in addition two more constituents are required: Subject and Finite. Almost same pattern graph is valid in case of selecting bound feature instead of indicative.

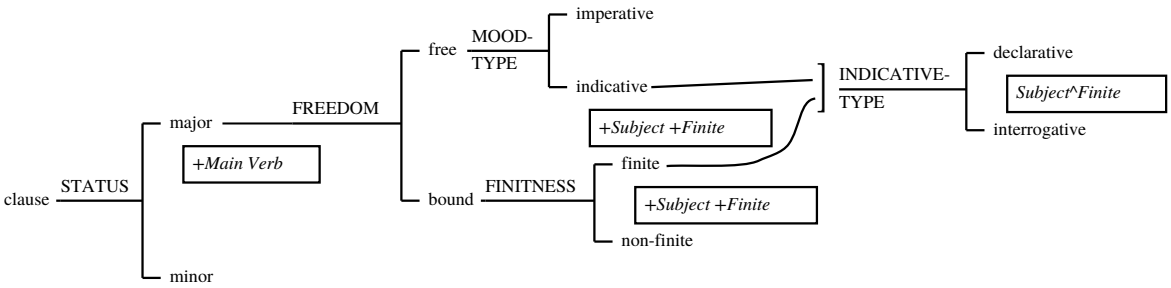


Fig. 1.23 An adapted fragment of a Mood system from (Halliday & Matthiessen 2013: 162)

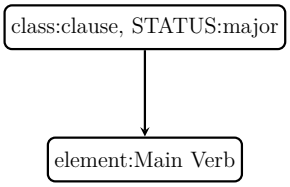


Fig. 1.24 A graph pattern for *major* feature selection in Figure 1.23

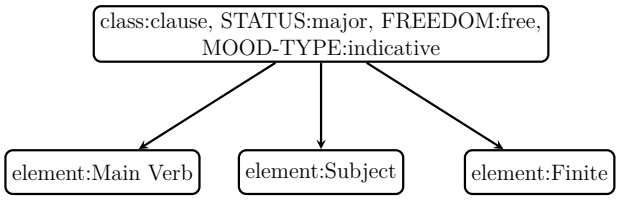


Fig. 1.25 A graph pattern for *indicative* feature selection in Figure 1.23

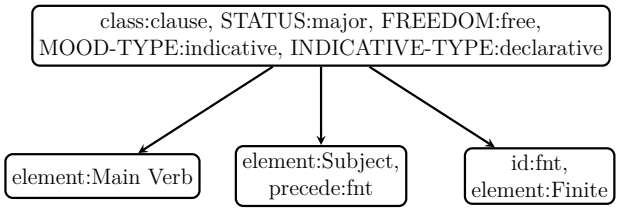


Fig. 1.26 A graph pattern for *declarative* feature selection in Figure 1.23

In Figure 1.26 the selection is taken one step further to the declarative feature. The associated realisation rule is ordering the Subject and Finite elements. This is captured via the "precede:fnt" where "fnt" is the id of the Finite constituent.

The pattern does not need to refer to the complete selection path but can be limited to the context of a few related features. For example to represent selection of indicative and declarative features in isolation is depicted in Figure 1.27. In this case the class of the parent constituent (that in the above cases is the clause) is no longer specified because the restricted selection path and thus the root of the network is not reached. Also none of the preselected features major and free are specified either.

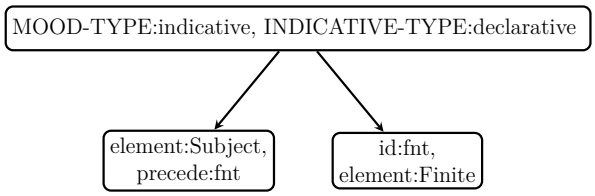


Fig. 1.27 A graph pattern for the selection if *indicative* and *declarative* features in Figure 1.23

One limitation, however, of the graph patterns is dealing with conflation phenomena. For example, the cases of the Main Verb conflated with Finite elements have to be accounted by with pattern graphs where one constituent has two functions instead of pattern graphs with two separate constituent each with the corresponding function. This limitation can be addressed in the future by manipulating the definition of the feature structure matcher described in Section 1.4.

1.8 Discussion

This chapter described from a computer science perspective the SFL concepts introduced in Chapter **??** and prepares the ground for the following chapters that address the implementation details of the Parsimonious Vole parser.

A central theme covered here are the graphs and graph patterns. They play the key role in identifying grammatical features in dependency and constituency structures. They are also an excellent candidate for expressing *systemic realizations* which have not been considered in the current work as being associated to the systemic features.

In Section 1.7 I mentioned that authoring realisation rules is a difficult task and requires a proper tool support. The same is the case for graph patterns and even more so when they need to be related to systemic networks or network parts. The system network authoring tool, such as the one available in UAM Corpus Tool (O'Donnell 2008b), should provide also a graph pattern editor allowing association of graph patterns to systemic features. Unfortunately building such an editor is out of the scope of the current work and is among the priorities in the future developments. Also employing a mature specialised technology for manipulating large amounts of graph data available in the Semantic Web suite of tools is another direction for the future described in the Section 3.2.

In the next chapter I describe the parsing pipeline and how each step is implemented starting from Stanford dependency graph all the way down to a rich constituency systemic functional parse structure.

Chapter 2

The Empirical Evaluation

The present parser is evaluated by comparing the text analysis it outputs with manually analysed text. The evaluation seeks to measure the parser *precision* i.e. how many segments have been produced by the parser that are also found in manual analysis giving us; and the parser *recall* i.e. how many correct segments have been produced by the parser relative to the total number of produced segments.

In order to count correct and incorrect number of segments they need to be matched first. This task is not a trivial because of the annotation mistakes, some differences in grammar and methodology of treating certain phenomena described below. In this work slight divergences are tolerated provided that the segment label is the same and there is other segment that constitutes a better match. This is known, in computer science as the Stable Marriage problem defined in Section 2.4. The next two sections define segments and provide details on how and why they diverge. Section 2.3 describe how the segment labels differ and how they are mapped to extend the scope of the evaluation.

To establish the matches between manual and automatic segments I have implemented a variation of the Gale-Shapley algorithm (Gale & Shapley 1962) which is described in Section 2.4 below. Then, the section that follows, are finally described the empirical findings of the current evaluation.

In the current evaluation I use two sets of annotated texts. Table 2.1 summarises the corpora used for evaluation in this work.. The first dataset (OCD) was created by Ela Oren and I and is focused on syntactic constituency structure and clause Mood features. The texts represent blog articles of people diagnosed with Obsessive Compulsive Disorder (OCD) who self-report on the challenge of overcoming OCD. The second dataset (OE1) is the PhD work of Anke Shultz covering Cardiff Transitivity analysis. It comprises 31 files spanning over 1503 clauses and 20864 words. In addition

She provided another similar smaller corpus of 157 clauses that si also included into the evaluation.

Corpus	Elements	System Network	# characters (thousands)	# clauses	Annotator(s)
OCD	Syntax	Mood	16.2	147	Ela Oren & Eugeniu Costetchi
OE1	Semantics	Transitivity	51.8	1503	Anke Schultz
BTC	Semantics	Transitivity	5.6	157	Anke Schultz

Table 2.1 Evaluation corpus summary

The OE1 and BTC datasets have not been developed for the purpose of evaluating the current parser however they are compatible and can be adapted to evaluate (a) the boundaries of a constituent segment, (b) the constituent semantic function and (c) some Transitivity features. To enable each of these evaluations the annotation data and parser output need to be represented in such a way that they are comparable to each other. Specifically the feature names had to be harmonised with the ones from PTDB. The adjustments were the same as described in Section **??**.

2.1 Segment definition

To compare the segment boundaries we need to understand how they are represented in each output and how they can be brought to a common for comparison.

Listing 2.1 Segment example in UAM corpus tool

```
<segment numbersid="4" numbersstart="20" numbersend="27"
numbersfeatures="configuration;relational;attributive"
numbersstate="active"/>
```

Both datasets were created with UAM Corpus Tool (O'Donnell 2008a,b) version 2.4. The annotations, in this software, are recorded as segments spanning from a start to an end positions in the text file together with the set of features (selected from a systemic network) attributed to that segment. There are no constituency or dependency relations between segments. In Listing 2.1 is the XML representation for an example annotation segment where the *id* attribute indicates the unique identification number within the annotation dataset, the *start* and *end* attributes define the segment between two character offsets relative to the beginning of the text file.

Listing 2.2 Raw text example in annotation data

```
0 0Red riding hood excerpt24
1 25"What have you in that basket,  Little Red Riding Hood?"82
```

```
2 83
3 84"Eggs and butter and cake, Mr. Wolf."111
```

Listing 2.2 presents an example raw text from the annotation dataset containing an initial line and two sentences separated by an empty line. The greyed numbers in the beginning and end of each line indicate character offset. In some files, the first line plays the role of a header containing a title the file name. In this example it is a title. Either way, this, first line, is neither considered for annotation nor for parsing. Some files contain one sentence per line, others separate sentences by an extra blank line and in other files, the text is one continuous block. The text may contain sporadically tabs and blank spaces such as in the line 1 between the comma and the Little Red Riding Hood.

Parsimonious Vole, parses sentences one by one and not whole text document at once. Before parsing, the document is chunked into sentences guided by the punctuation delimiters. In addition, the Stanford dependency parser normalises the input text by trimming spaces and tabs, adjusting character encoding and replaces some special characters such as quotes, brackets, punctuation, etc.

Listing 2.3 presents the preprocessed text before it is parsed. The title line is cropped from the file. Each line contains a separate sentence. The text is tokenised such that words and punctuation marks are separated by a single space.

Listing 2.3 Parser preprocessed text example

```
0 0" What have you in that basket , Little Red Riding Hood ? "60
1 61" Eggs and butter and cake , Mr. Wolf . "102
```

After the parser preprocessing process, the input and output texts are no longer identical. The sentence and word offsets are no longer reliably findable with respect to the original text and the segments need to be matched. There are two fundamental kinds of segment misalignments: the segment offset shifting and the segment resizing (either shrinking or enlarging). Sometimes it can also be a combination of the two. Allen interval algebra (Allen 1983) systematizes the basic thirteen relations to position segments relative to each other. Nonetheless in the current work we are not interested to investigate how the segment offset shifts but rather what would be an efficient way to match them independent of their position.

To make manual and automatic annotations comparable, the constituency graphs need to be reduced to a set of segments corresponding to each constituent. The first task of the evaluation is to find matches or near-matches between two segment bundles that is addressed in the next section.

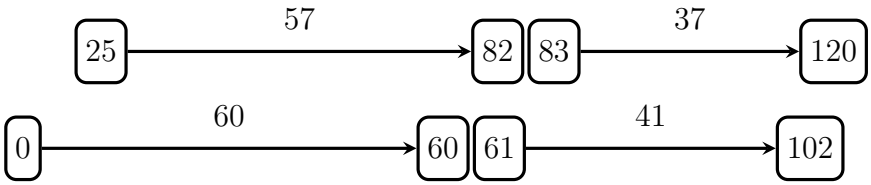


Fig. 2.1 Graphic representation of the sentence segment misalignment between Listing 2.2 and Listing 2.3

2.2 Reducing the CG to a set of segments

If we ignore the set of associated labels (systemic features), which we will consider in the next section, then two segments are we identical when (a) their start and end indexes and (b) the text in between is identical.

To address this issue in the current evaluation, the text processed by the parser is re-indexed back into the original raw text at the level of words (tokens), constituents and sentences. The high level perspective for generating segments for the constituency graphs indexed on the original text file is provided in Algorithm 2.

Algorithm 2: Algorithm to generate segments for the CG bundle indexed on the raw text

```
   input  : CG bundle, text
1  begin
2      offset ← 0
3      for cg in CG bundle:
4          generate segments for cg indexed on text given the offset
5          offset ← the end of cg
6  end
```

The result of the Algorithm 2 is a set of segments that are indexed according to the raw text by iterating the resulting constituency graphs one by one and indexing each with respect to the offset given by the previous one.

The way each CG is indexed is described by the Algorithm 3 which returns a set of segments from the constituency graph knowing given an offset. First a token level indexing is performed and corresponding segments generated. Then for each CG constituent, segments are generated based on its word span. The indexes are set from beginning of the first word to the end of the last word. The labels of the segment are the CG class, functions and systemic features discussed in the next section.

$$d = \sqrt{(start_S - start_T)^2 + (end_S - end_T)^2} \tag{2.1}$$

Algorithm 3: Algorithm to generate segments for CG constituents indexed on the raw text

```
   input  : cg, text, sentence offset
 1 begin
 2     words ← get cg the list of words
 3     for word in list of sentence word segments:
 4         find the word in the text after a given sentence offset
 5         if word found:
 6             start ← get first word start index
 7             end ← get the last word end index
 8             create a new segment (start, end, word)
 9         else:
10             generate a warning (manual adjustment needed)
11     for node in cg in BFS postorder:
12         find the word span of the constituent
13         start ← get first word start index
14         end ← get the last word end index
15         labels ← get node class, function and features
16         create new segment (start, end, labels)
17     return set of segments
18 end
```

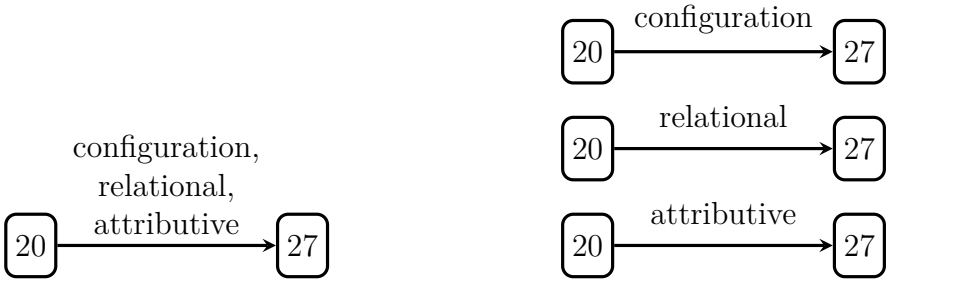
$$d = \sqrt{\Delta^2_{start} + \Delta^2_{end}} \tag{2.2}$$

Later in this chapter I address how the manually created segments and parser generated segments are aligned. Here it is noteworthy to mention that the manually created segments contain errors. Some segments are either shifted and include the adjacent spaces or, the converse, leave out one or two characters of a marginal word. For this reason the segment alignment process needs to tolerate a certain distance between the stand/end indexes of aligned segments. I adopted the geometric distance to measure the difference between the segment start and end indexes. For two segments $S(start_S, end_S)$ and $T(start_T, end_T)$ the geometric distance is defined in Equation 2.1. We can replace the difference between start and end indexes with Δ_{start} and Δ_{end} notation and obtain a reduced form provided in Equation 2.2.

2.3 Considering the segment labels

Provided that the task of indexing the segment indexes and content such that they "roughly" correspond to each other we need to start comparing the segment labels. However the labels of manually created segments differ from the segments provided by the parser. Besides shift in the offset mentioned above, another difference is in number of ascribed segments. In many instances, the manual annotation is less delicate and thus contain less features than the one provided by the parser. Still, some of the manual annotations contain features that are not provided by the parser (i.e. not yet implemented), such as the Thematic structure of the clause or the Modal Assessment features.

To address this issue, in the current evaluation, the segments are reduced to carry only one label. The consequence is that the segments with multiple features (Figure 2.2a) are broken down into multiple segments with the same span (Figure 2.2b) for each feature. When each segment contains exactly one feature the evaluation can be focused on one or a set of features of interest by selecting only the segments that contain exactly those.



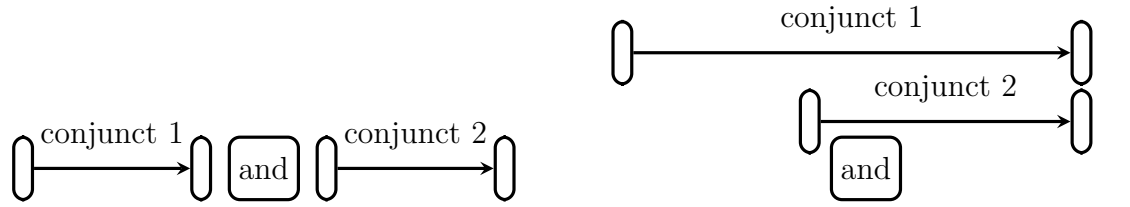
(a) A segment with a set of features (b) A set of segments with single features

Fig. 2.2 Example of breaking down a segment with multiple features into set of segments with a single feature

There are more differences in the grammar and annotation style between the manual and automatic processes. This means that both structure and feature names provided by the system networks differs (already mentioned above that the Theme system network is not implemented in the current version of the parser but provided in some manual annotations).

In the corpus, punctuation marks such as commas, semicolons, three dots and full stops are not included into the constituent segments while the parser includes them at the end of each adjacent segment. None of the conjunctions (such as "and", "but", "so", etc.) are included into the conjunct segments, they are considered markers in

the clause/group complexes rather than part of the constituent. The treatment of conjunction is discussed in Section **??**. The parser, on the other hand, includes the conjunctions into the following adjacent segment. Moreover the conjuncts spans differ as well. Instead of being annotated in parallel, as depicted in Figure 2.3a, the segments are subsumed in a cascade from the former to the latter as depicted in Figure 2.3b.



(a) Conjuncts annotated as parallel segments (b) Conjuncts annotated as subsumed segments

Fig. 2.3 Treatment of conjunctions in the corpus compared to the parser

The differences in the grammar are due to the slight variations in the feature names and in the system network structure. For example the element names are spelled differently "predeictic" and "pre-deictic", "root" and "clause" or "explative-marker" and "expletive". Similar spelling variations are present in the feature names for example "two role" and "two-role-action" and other variations alike. To remedy this issue the current evaluation establishes mappings between the manual and automatically generated features.

Now that the main divergences have already been outlined, I proceed to explain how the stable matches between the segments are established.

2.4 The matching algorithm

The task of aligning two sets of labelled segments is almost the same as the well know problem in computer science called *stable marriage problem* (Gusfield & Irving 1989). The standard enunciation of the problem is provided below and is solved in an efficient algorithm named Gale-Shapley (Gale & Shapley 1962) after its authors.

> Given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. When there are no such pairs of people, the set of marriages is deemed stable Iwama & Miyazaki (2008).

In the context of this evaluation the group of men is associated with the segments generated automatically by the parser and the group of women with the segments available from the manual analysis.

The standard stable marriage problem is formulated such that there is a group of men and a group of women and each individual from each group expresses their preferences for every individual from the opposite group as an ordered list. The assumption is that the preferences of every individual are known and expressed as a complete ordered list of individuals from the opposite group ranging from the most to the least preferred one. Thus the preference list must be *complete* and *fully ordered*.

To fulfil these requirements I construct a distance matrix from each automatically created segment to every and manually created one. The distance measure considered here is the Euclidean one provided in Equation 2.2 above. The matrix represents the complete and fully ordered set of preferences stipulated in the original problem formulation. In addition to distance the segments need to carry the same labels in order to be considered a match. This condition is not found in the original problem but is considerer in Algorithm 4 below.

Algorithm 4: The algorithm for matching automatic and manual segments

```
   input  : aut segments, man segments
 1 begin
 2     mark all aut segments and man segments free
 3     compute distances from each man segments to every aut segments
 4     while ∃ free aut segments:
 5         aut ← first free from aut segments
 6         if ∃ man segments not yet tested to match aut:
 7             man ← the nearest among man segments to aut with identical label
 8             if man is free:
 9                 match aut and man
10                 mark aut and man as non-free
11             else:
12                 aut′ ← the current match of man
13                 if aut is closer to man than aut′:
14                     match aut and man
15                     mark aut and man as non-free
16                     mark aut′ as free
17         else:
18             mark aut as non-free and non-matching
19     fintq
20 end
```

The input to the algorithm is the set of automatically generated segment list aut segments and the manually created segment list man segments. It begins with initialising all the segments as *free* meaning that none of them are matched yet (originally married). The segments can be marked either be as *non-free matching* or *non-free non-matching*. A Part of the initialisation is also creation of the distance matrix mentioned above representing a complete and ordered set of preferences for each segment.

The algorithm iterates over the aut segments for as long as there exist free ones. If there exists a man (manually created segment) which have not been attempted to match aut then attempt matching aut to the nearest man that has the same label. If man is free then it is a match. Otherwise compare aut to aut′ (currently assigned match of man) and then consider a match with the nearest one (i.e. shortest distance). When there are no man segments to test mark aut as non-free and non-matching meaning that it shall no longer be considered by the algorithm.

In the end the algorithm provides a list of successful matches out of which we can also deduce which aut segments man segments have not been matched. The next section provides the empirical measurements resulting from applying the current method.

2.5 The measurements

In this section I present a series of measurements addressing two quality criteria of the parser. Firstly, the degree of divergence in segment spans between manual and parse segments in all corpora (OCD, BTC and OE1). And secondly, the accuracy of the parser with respect to manual annotations measured through precision, recall and F_1 measures. The syntactic constituency and some Mood features are derived from OCD corpus only, followed by the semantic constituency and Transitivity features evaluation based on OE1 and BTC corpora.

2.5.1 Segment divergence: general findings

The segments don't perfectly align, as described in previous sections, due to minor differences in annotation approach, text normalisation and trimming before parsing (that was not performed before the manual annotation), errors in the annotations (missing or including characters). The misalignment is measured using the geometric distance on segments matched with Algorithm 4. The presented statistics are calculated on a number of over 12500 segment pairs whose mean distance is 4.84 with a standard deviation of 14.51. The distance is distributed between a minimum 0 and maximum

219. A mean close to the minimum point and a large standard deviation indicates that most of the data points are situated between 0 and slightly over the mean continued by a very long tail of rare and distant data points. This intuition can be seen in the histogram depicted in Figure 2.4 below. This dataset resembles the Pareto distribution where 74% are almost not shifted (by up to 1 character) or 83% of the segments are slightly shifted up to 5 characters.

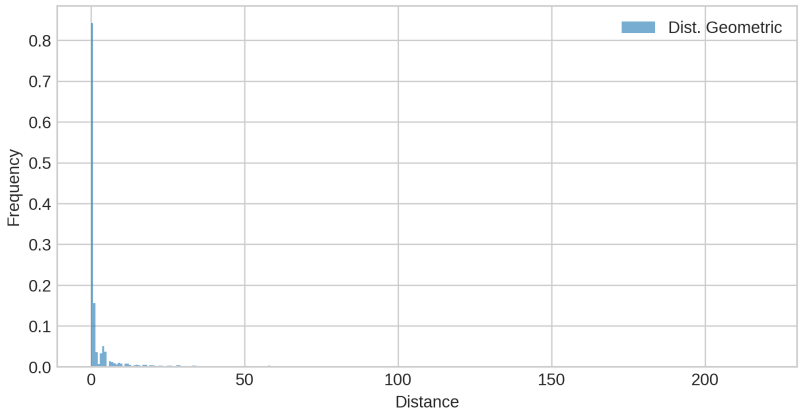


Fig. 2.4 Full histogram of the distance distribution of the matched segments (binning=300)

Because of a very long tail, I reduce the further analysis to the distance span between 0 and 40. Figure 2.6 depicts the histogram of distances between matched segments (in 300 bins). It shows that most segments (89%) are cumulated in first two bins with distance up to 2. The consequent three bins also contain a significant portion of segments with distance up to 5. The rest of the bins, represent a very long tail, spanning on distances up to maximum 219 and containing a small amount of segments.

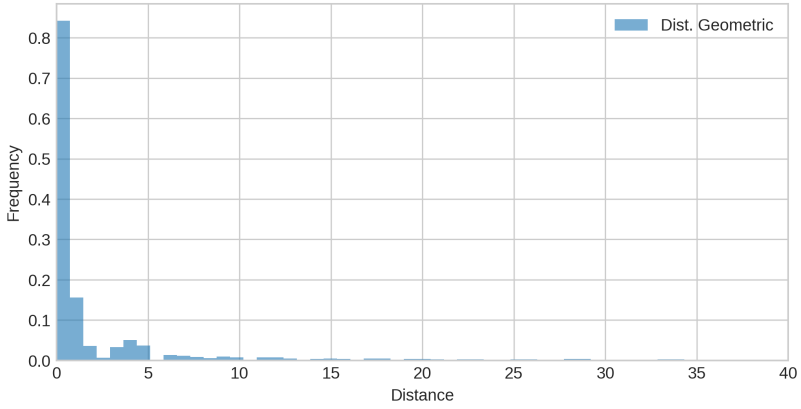


Fig. 2.5 Reduced histogram of the distance distribution of the matched segments (binning=300). View reduced to the a distance of 40 characters

This distribution can be viewed in it's cumulative form depicted in Figure 2.6. Here we see that over 51% of segments are perfectly aligned. 80% of the segments are slightly shifted up to a distance of maximum 5 characters. This can be explained by differences in (a) punctuation, (b) conjunction and (c)verbal group treatment described above. The next 5% of the segments are shifted between 5 and 10 characters. The last 10% cover the heavy shift of distances 20 to 219. This may be due differences in verbal group treatment, subsumed conjuncts (clause and group conjunctions) and erroneous prepositional phrase attachment (treated as Qualifier instead of Complement/Adjunct or vice versa).

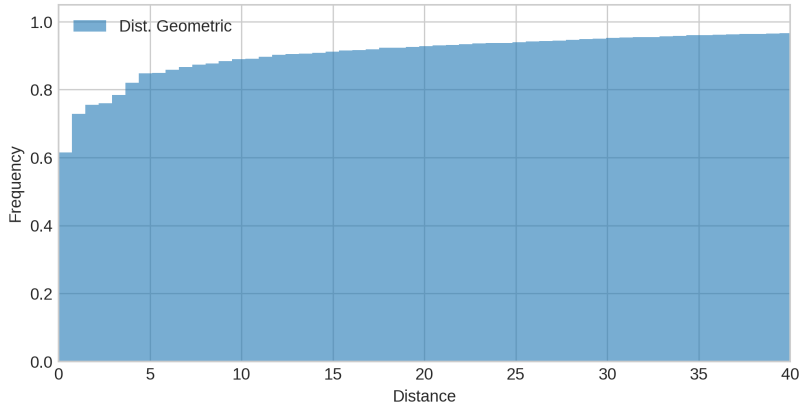


Fig. 2.6 Cumulative histogram of the distance distribution of the matched segments (binning=300). View reduced to the a distance of 40 characters

2.5.2 Segment divergence breakdown by element type

If we take a closer look at the Figure 2.5 we will notice some spikes and a tendency towards some local values. What would the histogram look like if we group values in a different manner considering these spikes and the long tail of the distribution. In Figure 2.5 we can observe several dips at around distances 3, 6, 11, 13, 17, 19, 22, 24 and so on. I decided to consider these dips for deciding bin borders in the new bin design. And, since the tail is very long with very few values, as the bins advance to the left, each is designed over longer span this way compressing the tail. In Table 2.2 is presented such binning design. Also each interval is assigned a category that means the degree of deviation.

Besides custom binning described above, looking at each segment label in part brings more clarity to the evaluation. The most relevant segment labels are the unit elements. In the current evaluation, the annotators provide clause level syntactic

degree	insignificant	tiny	little	moderate	significant	high
distance intervals	0-3	3-5	5-10	10-20	20-50	50-250

Table 2.2 The progressive binning scale considering the dataset properties

(subject, complement, adjunct, predicator, finite) and semantic (configuration, main verb, participant) elements. These elements are used a dimension in the breakdown of the segment divergence that follows.

Using the bins defined in Table 2.2 the distribution of segment deviations grouped by the main syntactic elements is provided in Table 2.3 that is also depicted in Figure 2.7.

element	**% of segments per degree of deviation**					
	insignificant (0-3)	**tiny (3-5)**	**little (5-10)**	**moderate (10-20)**	**significant (20-50)**	**high (50-250)**
predicator	29.38	0.95	0.30	0.10	0.00	0.05
subject	22.70	0.10	0.25	0.30	0.05	0.00
adjunct	13.86	0.45	0.60	0.15	0.85	0.20
complement	12.10	0.75	1.46	2.26	1.96	1.86
finite	9.19	0.00	0.00	0.00	0.05	0.05

Table 2.3 Percentage of segments deviated to a given degree for major syntactic elements

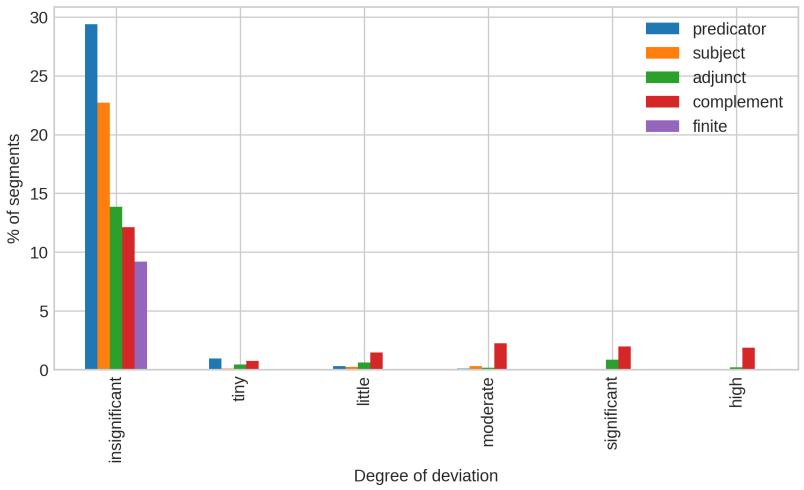


Fig. 2.7 Bar chart of the segments deviated to a given degree for major syntactic elements

In Figure 2.7 we can see that most deviations are insignificant. The number of segments in the rest of the bins, from tiny to high, is below 1% in every bin which perhaps reflects errors in annotation and/or matching algorithm requiring further

investigation. In case of complements, however, the proportion of segments is slightly higher (0.7–2.2%). This may be explained by the problem of prepositional phrase attachment and further analysis is needed to test this hypothesis.

When we switch to a grouping by the main Transitivity elements maintaining the same bins defined in Table 2.2, then the distribution of segment deviations looks as outlined in Table 2.4. The same data are depicted in Figure 2.8.

feature	**% of segments per degree of deviation**					
	insignificant (0-3)	**tiny (3-5)**	**little (5-10)**	**moderate (10-20)**	**significant (20-50)**	**high (50-250)**
participant-role	36.78	2.21	3.48	2.80	3.04	1.67
main	20.75	3.48	1.23	0.39	0.00	0.00
configuration	7.75	3.73	2.80	3.04	4.56	2.31

Table 2.4 Percentage of segments deviated to a given degree for major semantic elements

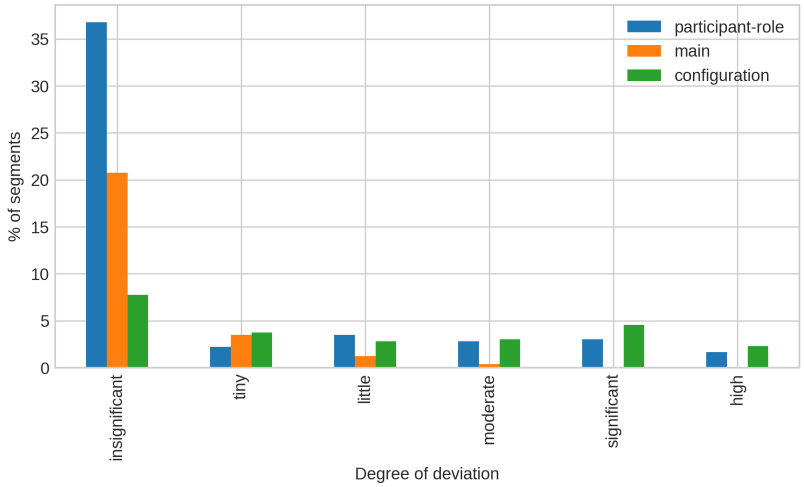


Fig. 2.8 Bar chart of the feature segments deviated to a given degree for major semantic elements

Similarly to Figure 2.7, in Figure 2.8 we can see that most of the deviations are insignificant (0-3 characters). In the rest of the bins representing higher degrees of deviation the amount of segments is up to 4.7% which reflects higher number of shifted segments. One exception is the main verb whose occurrences decreases towards higher degrees of deviation, i.e. it is shifted by an insignificant, tiny or little degree that is up to about 10 characters. This is explainable by the fact that the main verb is most of the time (if not always) a single word, which is always shorter than the configuration or participant elements which usually span clauses of groups comprising multiple words.

The large variations in participant seems to correlate with complement variation and might be explained by the attachment errors. In cases of high configuration deviation however further investigation are needed because these may possibly be errors in the segment matching algorithm or other unknown anomalies.

2.5.3 Syntactic evaluation: Constituency elements

The evaluation data in this and the following sections will be presented in tables with the same structure. Using Table 2.5 as example I explain next what the columns mean. The first column will contain the name of the unit type, element or feature. The next three columns *Match*, *Manual mn* and *Parse mn* represent the number of segments that are considered identical between the corpus and parser, the number of unmatched the corpus (manually created) segments and the number of unmatched parser (automatically generated) segments. The next three columns *Precision*, *Recall* and F_1 represent standard accuracy metrics indicating the fraction of relevant instances among the retrieved instances, fraction of relevant instances that have been retrieved over the total amount of relevant instances and the harmonic mean of the previous two. In addition, the column *%Total matched* represents the percentage the current item (row) in the table while the *% Manual nm* and *%Parse nm* represent the number of remaining unmatched segments of the current item (row) that represents a translation of the Manual and Parse mn columns into relative terms.

Unit type	Matched	Manual nm	Parse nm	Precision	Recall	F1	%Total matched	%Manual nm	%Parse nm
clause	612.00	64.00	78.00	0.89	0.91	0.90	37.00	9.47	11.30
nominal-group	717.00	108.00	67.00	0.91	0.87	0.89	43.35	13.09	8.55
prepositional-group	119.00	39.00	39.00	0.75	0.75	0.75	7.19	24.68	24.68
adverbial-group	161.00	79.00	103.00	0.61	0.67	0.64	9.73	32.92	39.02
adjectival-group	45.00	36.00	38.00	0.54	0.56	0.55	2.72	44.44	45.78

Table 2.5 The evaluation statistics for the main constituency unit types

The syntactic accuracy aims to measure how well the main unit types and the clause main elements have been detected by the parsed compared to the corpus. The evaluation is performed on the OCD corpus. This evaluation is restricted to the clause and four group types: nominal, prepositional, adverbial and adjectival. No clause complexes, group complexes or word types are included. The evaluation results are presented in Table 2.5 where we can see that clauses and nominal groups have an F_1 score of about 90%. Prepositional, adverbial and adjectival group scores decrease to 55% and requires investigation of the errors in the parsing and matching algorithms. There is also a contrast in the number of segments, visible in the bar chart from Figure

2.9, between the first two element types and the last three with a ratio of one to four or more.

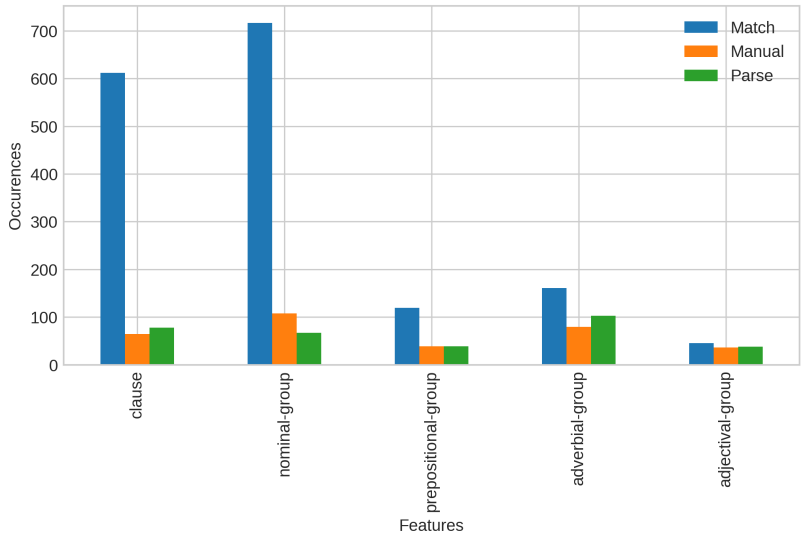


Fig. 2.9 Bar chart of matched and non-matched (manual and parse) segments of the main constituency unit types

Table 2.6 presents the evaluation result for the main clause elements. Some of them such as, auxiliary verbs, main verb extension, negation particle, and others have been omitted in the corpus and thus missing in the present evaluation. For the predicator (i.e main verb) and subject elements the F_1 measure raises to 90%. For the complements and finite the F_1 score is 67% and 63%. Surprisingly the complements have a small number of corpus unmatched segments and a high number of parser unmatched segments. This is explained by a flaw in the annotation methodology because the clausal complements were often annotated directly as new clause and omitting to draw the same segment with complement element. This required the corpus revision and correction. Adjuncts however have a higher number of unmatched segments on both sides and this may be dues to bugs in the parser.

Clause element	Matched	Manual nm	Parse nm	Precision	Recall	F1	%Total matched	%Manual nm	%Parse nm
predicator	613	60	79	0.89	0.91	0.90	30.79	8.92	11.42
subject	466	22	86	0.84	0.95	0.90	23.41	4.51	15.58
complement	406	43	350	0.54	0.90	0.67	20.39	9.58	46.30
adjunct	321	159	224	0.59	0.67	0.63	16.12	33.12	41.10
finite	185	3	392	0.32	0.98	0.48	9.29	1.60	67.94

Table 2.6 The evaluation statistics for the clause main elements

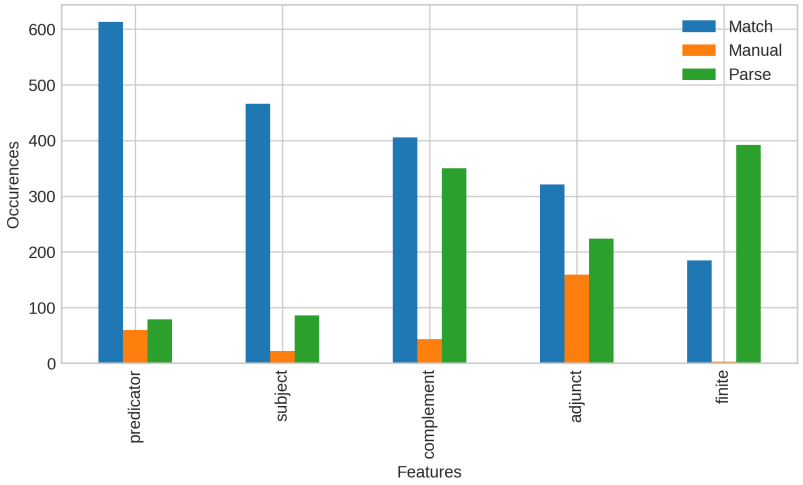


Fig. 2.10 Bar chart of matched and non-matched (manual and parse) segments of the clause main elements

2.5.4 Syntactic evaluation: Mood feature selections

In this section I present the evaluation of Mood system network selections. The corpus contains selections from a pat of the system network that is depicted in Figure 2.11. The full system network is provided in Figure **??**. Employing the entire system network in the annotations was difficult because as the delicacy increases the time spent for the annotation process increases.

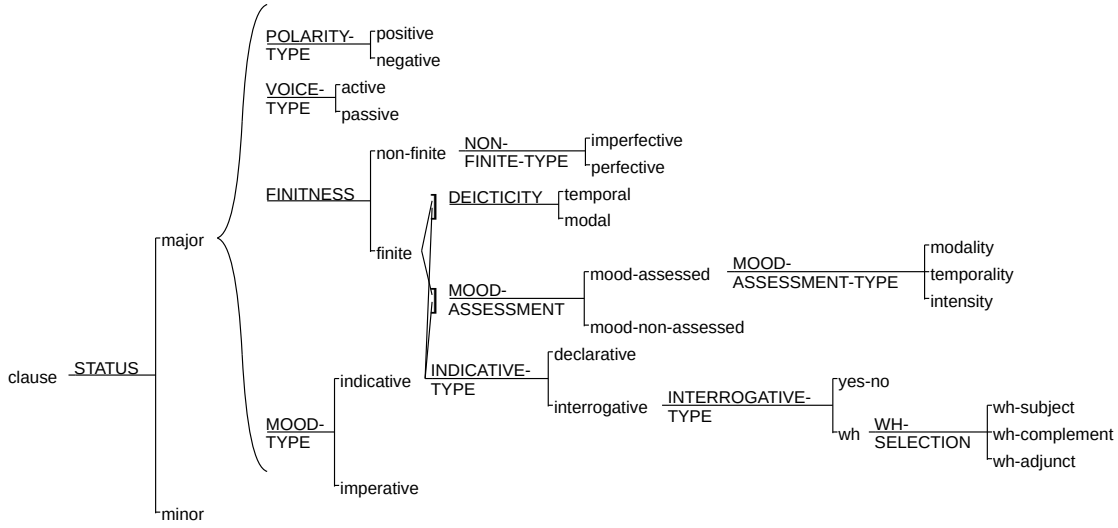


Fig. 2.11 The part of the Mood system network that has been used in OCD corpus annotation

Table 2.7 summarises the evaluation results for MOOD system network features grouped by system names (in capital). The precision and recall values vary quite a lot

Feature	Matched	Manual nm	Parse nm	Precision	Recall	F1	%Total matched	%Manual nm	%Parse nm
POLARITY-TYPE									
positive	485.00	125.00	55.00	0.90	0.80	0.84	89.48	20.49	10.19
negative	57.00	10.00	70.00	0.45	0.85	0.59	10.52	14.93	55.12
VOICE-TYPE									
active	553.00	102.00	68.00	0.89	0.84	0.87	98.05	15.57	10.95
passive	11.00	11.00	28.00	0.28	0.50	0.36	1.95	50.00	71.79
FINITNESS									
non-finite	99.00	19.00	38.00	0.72	0.84	0.78	15.84	16.10	27.74
finite	526.00	33.00	554.00	0.49	0.94	0.64	84.16	5.90	51.30
NON-FINITE-TYPE									
perfective	71.00	12.00	16.00	0.82	0.86	0.84	73.20	14.46	18.39
imperfective	26.00	9.00	24.00	0.52	0.74	0.61	26.80	25.71	48.00
DEICTICITY									
temporal	446.00	74.00	55.00	0.89	0.86	0.87	97.38	14.23	10.98
modal	12.00	33.00	6.00	0.67	0.27	0.38	2.62	73.33	33.33
MOOD-ASSESSMENT-TYPE									
temporality	35.00	17.00	27.00	0.56	0.67	0.61	56.45	32.69	43.55
modality	15.00	32.00	8.00	0.65	0.32	0.43	24.19	68.09	34.78
intensity	12.00	14.00	43.00	0.22	0.46	0.30	19.35	53.85	78.18
MOOD-TYPE									
indicative	455.00	216.00	37.00	0.92	0.68	0.78	99.13	32.19	7.52
imperative	4.00	1.00	31.00	0.11	0.80	0.20	0.87	20.00	88.57
INDICATIVE-TYPE									
declarative	355.00	260.00	27.00	0.93	0.58	0.71	88.31	42.28	7.07
interrogative	47.00	7.00	63.00	0.43	0.87	0.57	11.69	12.96	57.27
INTERROGATIVE-TYPE									
wh	40.00	6.00	57.00	0.41	0.87	0.56	88.89	13.04	58.76
yes-no	5.00	3.00	8.00	0.38	0.62	0.48	11.11	37.50	61.54
WH-SELECTION									
wh-subject	9.00	3.00	7.00	0.56	0.75	0.64	32.14	25.00	43.75
wh-adjunct	11.00	15.00	3.00	0.79	0.42	0.55	39.29	57.69	21.43
wh-complement	8.00	0.00	62.00	0.11	1.00	0.21	28.57	0.00	88.57

Table 2.7 The evaluation statistics for POLARITY-TYPE systemic choices

from a minimum of 11% up to a maximum of 93% and the harmonic mean, the F_1 score, between 30% up to 87% averaging to almost 60%. The details can be read in Table 2.8. The distribution of these values can be seen in Figures 2.12 and 2.13. A noticeable feature is the presence of two peaks in the precision and recall distributions: one around 50% and the other one around 90%. They translate into a similar F_1 distribution with peaks at 60% and 85%.

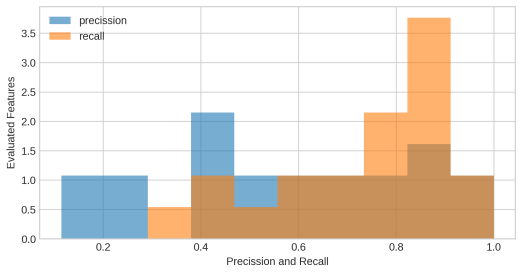


Fig. 2.12 The distribution of precision and recall for selected features from the Mood system network

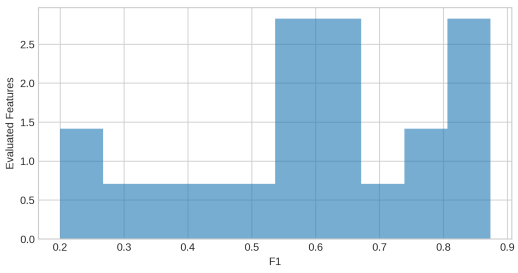


Fig. 2.13 The distribution of F1 score for selected features from the Mood system network

	precision	**recall**	**F1**
feature count	21.00	21.00	21.00
mean	0.57	0.73	0.59
standard deviation	0.27	0.18	0.21
min value	0.11	0.32	0.20
25% quantile	0.41	0.62	0.48
50% quantile	0.56	0.80	0.61
75% quantile	0.82	0.86	0.78
max value	0.93	1.00	0.87

Table 2.8 Descriptive statistics of the precision, recall and F1 scores

Within most systems, the F_1 scores exhibit high contrast from one feature to the other. What does it means is discussed in the next two cases. For example in POLARITY-TYPE system the positive polarity feature scores 84% F_1 measure and negative one almost 60%. As per Definition **??**, the system features are mutually exclusive. The polarity of an English clause is positive by default unless a negation marker is found and this represents only 10% of the clauses in the corpus. This means that it should be sufficient for the parser to detect one feature with a reasonably high F_1 score then the converse feature should be detectable with the same degree of accuracy. Yet the current data invalidate this hypothesis.

In case of POLARITY-TYPE system, the phenomena may be explained, as a consequence of low delicacy in the parsing algorithm. Current implementation determines polarity by checking for the presence of the negation verbal marker only. Nonetheless a more delicate polarity testing will have to take into consideration polarity indicators from the subject, complement and adjuncts of various type that may have been taken into consideration during the annotation process. Providing an incomplete, less delicate implementation for systemic choices may be a source of errors. This hypothesis requires further investigation.

Nonetheless, the same phenomena can be seen if we look at the systems that do not span other ones with higher degrees of delicacy. For example, detection mechanism for VOICE-TYPE is implemented similarly to POLARITY-TYPE. The parser checks whether there is a passive order of elements in the clause otherwise the active voice is selected. Detection of the positive voice scores a much higher F_1 measure of 87% than the negative one of 36%. There is no more the problem of low delicacy and still there is a discrepancy between the F_1 scores of the two features.

This high discrepancy between the feature F_1 scores can be seen as follows. Most instances of active voice are easy to detect. But there is a portion of cases, regardless of

the voice, that are difficult for the parser to distinguish. The passive voice selections are executed mostly for these ambiguous cases. More investigation is required to pinpoint exactly how such phenomena materialises.

Nonetheless, in the future implementation, we can take advantage of the mutual exclusivity of system features and capitalise on the above finding, at least for leaf systems as follows. Because the score of one feature is higher, then clauses should be provided only with selections of only that one leaving the complement feature unselected. In a way assign only features that can be provided with a high confidence (using F_1 score as a measure of that). Then an additional process can select features of systems that are known to be selected but without a score.

2.5.5 Semantic evaluation: Constituency elements

This section describes the empirical findings regarding the semantic parsing. This evaluation is based on the OE1 and BTC corpora created by Anke Schultz for her PhD project. She annotated the text with Cardiff Transitivity system. The employed elements are Configuration, Participant role and Main verb.

	Match	**Manual nm**	**Parse nm**	**Precision**	**Recall**	**F1**	**%Total matched**	**%Manual nm**	**%Parse nm**
Participant role	2780	233	1002	0.74	0.92	0.82	49.54	7.73	26.49
Configuration	1376	127	493	0.74	0.92	0.82	24.52	8.45	26.38
Main Verb	1456	160	605	0.71	0.90	0.79	25.94	9.90	29.35

Table 2.9 The evaluation statistics for the main semantic elements

2.5.6 Semantic evaluation: Transitivity feature selections

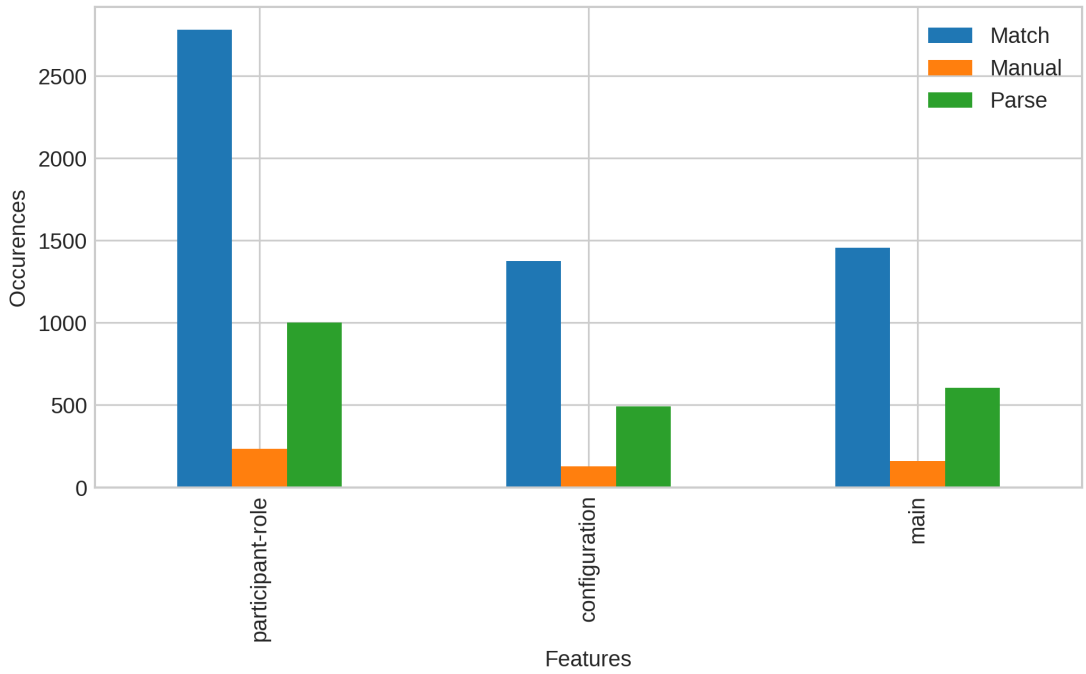


Fig. 2.14 Bar chart of matched and non-matched (manual and parse) segments of the main semantic elements

For the semantic evaluation has been used a corpus created by Anke Schultz for her PhD project annotated with Transitivity elements i.e Configuration, Process and Participant which from constituency point of view correspond to the Clause, Predicator and Participants(predicate arguments) to Subjects and Complements cumulated.

Name	*M/N*	*M/%*	*A/N*	*A/%*	*Prec*	*Rec*	F_1
Configuration	1466	-	1833	-	0.736	0.915	0.814
Process	1423	-	1814	-	0.707	0.902	0.791
Participant	2652	-	3398	-	0.732	0.925	0.816

Table 2.10 Transitivity System evaluation statistics

Transitivity analysis is semantic in nature and poses challenges in meaning selection beyond constituent class or function. Current parser does not select one meaning but rather proposes a set of possible configurations for each clause. If top level Transitivity features correspond fairly to the number of syntactic constituents, the more delicate features are parsed with an average of 2.7 proposed features for each manually annotated one.

Name	*M/N*	*M/%*	*A/N*	*A/%*	*Prec*	*Rec*	F_1
CONFIGURATION-TYPE	1466	23.12	1308	7.57	0.542	0.483	0.508
action	359	24.82	453	33.99	0.315	0.409	0.333
relational	546	37.79	445	34.77	0.645	0.530	0.574
mental	452	29.86	318	24.27	0.744	0.530	0.605
influential	81	6.69	91	7.39	0.556	0.519	0.484
event-relating	28	3.48	1	1.85	1.0	0.5	0.667
environmental	0	0	0	0	0	0	0
other	0	0	0	0	0	0	0

Table 2.11 Configuration type evaluation statistics

The semantic evaluation yields surprisingly positive results for relational and metal processes. At the same time, actions would have been expected to score a high accuracy but as seen in the results it is not the case. Despite a high number of them both in the corpus (24%) and in the parses (33%) they seem to be misaligned and perhaps not matched in the process. Further investigation should be conducted to spot the source of such a low score. Next I present a short critical discussion of the overall evaluation.

2.6 Discussion

Further investigation shall be conducted to determine the error types, shortcomings in the corpus and the parser. But beyond the simple improvement to the corpus such as adding the missing Finite elements it would benefit tremendously from a second annotator in order to evaluate reliability of the annotation itself and how much of a gold standard it can be considered for the current work.

Also the corpus size is very small and many features are missing or severely underrepresented and bear no statistical significance. For example event relating, environmental and other processes are missing from the corpus. Also the number of other features that the parser provides are missing from the manual analysis and it

would be interesting to add some of them to study how varies the accuracy distribution as the delicacy of features increases.

Since for both syntactic and semantic annotations there is only a single author annotation available, the results shall be considered indicative and by no means representative for the parser performance. Nevertheless they can already be considered as a good feedback for looking into certain areas of the grammar with considerably low performance in order identify the potential problems.

Chapter 3

Conclusions

The aim of this work is to design a reliable method for English text parsing with Systemic Functional Grammars. To achieve this goal I have designed a pipeline which, starting from a dependency parse of a sentence, generates the SFL-like constituency structure serving as a syntactic backbone and then enriches it with various grammatical features.

In this process a primary milestone the first steps is the creation of constituency structure. Chapter **??** describes the essential theoretical foundations of two SFL schools, namely Sydney and Cardiff schools, and provides a critical analysis of the two to reconcile on the diverging points on rank scale, unit classes, the constituency structure, treatment of coordination, grammatical unit structure, clause boundaries, etc. and state the position adopted in this work.

In order to create the constituency structure from the dependency structure there needs to be a mechanism in place providing a theoretical and a practical mapping between the two. The theoretical account on the dependency grammar and how it is related to SFL is described in Chapter **??**. The practical aspects and concrete algorithms are described in Chapter **??** together with the mapping rules used in the process.

To make clear what are the basic ingredients and how the algorithms are cooked, Chapter 1 introduces all the data structures and operations on them. These structures are defined from a computer scientific point of view emulating the needed SFL concepts. These range from a few graph types, simple attribute-value dictionaries and ordered lists with logical operators. In addition to that, a set of specific graph operations have been defined to perform pattern matching and system network traversals.

Once the constituency structure is created, the second milestone is to enrich it with systemic features. Many features can be associated to or derived from the dependency

and constituency graph fragments. Therefore graph pattern matching is a cornerstone operation used for inserting new or missing units and adding features to existing ones. I describe these operations in detail in the second part of 1. Then in Chapters **??** and **??** I show how these operations are being used for enrichment of the syntactic backbone with systemic features.

The more precisely graph pattern is defined the less instances it will be matched to and thus decreasing the number of errors and increasing the accuracy. The semantic enrichment is performed via spotting instances of semantic graph patterns. It is often the case that the patterns, in their canonical form, list all the participants of a semantic configuration but in practice, instances of such configurations may miss a participant or two. If applied in their canonical form the patterns will not identify with such the instance. One solution would be to reduce the specificity of the patterns, which leads to a high increase in erroneous applications or populate where possible the covert participants to yield successful matches. It is the second approach that was implemented in this work. To identify and create the covert participants I turned to Government and Binding theory. Two more contributions I bring in this thesis is the theoretical mapping from GBT into dependency structures covered in Chapter **??** and then a concrete implementation described in Chapter **??**.

In the last part of the thesis I describe the empirical evaluation I conducted in order to test the parser accuracy on various features. To conduct this evaluation I created together with Ela Oren a corpus using blog articles of OCD patients covering the Mood system and another corpus was provided to my by Anke Schultz covering the Transitivity system. The results show very good performance (0.6 – 0.9 F1) on Mood features slightly decreasing as the delicacy of the features increases. On Transitivity features, the results are expectedly less precise (0.4 – 0.8 F1) and constitute a good baseline for future improvements.

As discussed in the Section 2.6 further investigation shall be conducted to determine the error types, shortcomings in the corpus and the parser. Since for both syntactic and semantic annotations there is only a single author annotation available, the results shall be considered indicative and by no means representative for the parser performance. Nevertheless they can already be considered as a good feedback for looking into certain areas of the grammar with considerably low performance in order identify the potential problems.

3.1 Practical applications

A wide variety of tasks in natural language processing such as document classification, topic detection, sentiment analysis, word sense disambiguation don't need parsing. These are tasks can achieve high performance and accuracy with no linguistic feature or with shallow ones such as as lemmas or part of speech by using powerful statical or machine learning techniques. What these tasks have in common is that they generally train on a large corpus and then operate again on large input text to finally yield a prediction for a single feature that they have been trained for. Consider for example the existing methods for sentiment analysis: they will provide a value between -1 to 1 estimating the sentiment polarity for a text that can be anything from one word to a whole page.

Conversely, there are tasks where extracting from text (usually short) as much knowledge as possible is crucial for the task success. Consider a dialogue system: where deep understanding is essential for a meaningful, engaging and close to natural interaction with a human subject. It is no longer enough to assign a few shallow features to the input text, but a deep understanding for planning a proper response. Or consider the case of information extraction or relationship mining tasks when knowledge is extracted at the sub-sentential level thus the deeper linguistic understanding is possible the better.

Current parser is useful to achieve the latter set of tasks. The rich constituency parses can be an essential ingredient for further goals such as anaphora resolution, clausal taxis analysis, rhetoric relation parsing, speech act detection, discourse model generation, knowledge extraction and other ones.

All these tasks are needed for creating an intelligent interactive agent for various domains such as call centers, ticketing agencies, intelligent cars and houses, personal companions or assistants and many other.

In marketing research, understanding the clients needs is one of the primary tasks. Mining intelligence from the unstructured data sources such as forums, customer reviews, social media posts is particularly difficult task. In such cases the more features are available in the analysis the better. With the help of statistical methods feature correlations, predictive models and interpretations can be conveyed for task at hand such as satisfaction level, requirement or complaint discovery, etc.

3.2 Impact on future research

Pattern graphs and the matching methods developed in this work can be applied for expressing many more grammatic features than the ones presented in this work. They can serve as language for systematizing grammatical realizations especially that the realization statements play a vital role in SG grammars. The graph matching method itself can virtually be applied to any other languages than English. So similar parsers can be implemented for other languages and and respectively grammars.

Linguists study various language properties, to do so they need to hand annotate large amounts of text to come up with conclusive statements or formulate hypothesizes. Provided the parser with a target set of feature coverage, the scale at which text analysis is performed can be uplifted orders of magnitude helping linguists come with statistically significant and grounded claims in much shorter time. Parsimonious Vole could play the role of such a text annotator helping the research on text genre, field and tenor.

This section describes improvements of the project that are desirable or at least worth considering along with major improvements that arouse in the process of theoretical development and parser implementation.

3.2.1 Verbal group again: from syntactically towards semantically sound analysis

The *one main verb per clause* principle of the Cardiff school that I adopted in this thesis (briefly discussed in Section **??**) provides a basis for simple and reliable syntactic structures. The alternative is adopting the concept of verbal group, simple or complex, as proposed by the Sydney school in (Halliday & Matthiessen 2013: p.396–418, 567–592), which provides a richer semantically motivated description. However, analysis with verbal group complex is potentially complex one and subject to ambiguities.

Ants	*keep*	*biting*	*me*
Subject	Finite	Predicator	complement
Actor	Process: Material		Goal/Medium
	Verbal group complex expansion, elaborative, time-phase, durative $\alpha \longrightarrow = \beta$		

Table 3.1 Sydney sample analysis of a clause with a *verbal group complex*

Ants	*keep*	-	*biting*	*me*
Subject	Finite/Main Verb	Complement		
Agent	Process: Influential	Phenomena		
		Subject(null)	Main Verb	Complement
		Agent	Process: Action	Affected

Table 3.2 Cardiff sample analysis of a clause *embedded* into another

Check the sample analyses in Table 3.1 and 3.2. The two-clause analysis proposed by Cardiff school can be quite intuitively transformed into a single experiential structure with the top clause expressing a set of aspectual features of the process in the lower (embedded) clause just like in the Sydney analysis in Table 3.1.

The class of *influential* processes proposed in the Cardiff transitivity system was introduced to handle expressions of process aspects through other lexical verbs. I consider it as a class of pseudo-processes with a set of well defined and useful syntactic functions but with poor semantic foundation. The analysis with influential process types reminds me to an unstable chemical substance that, in a chain of reactions, is an intermediary step towards some more stable substance. Similarly, I propose merging the two clauses towards a more meaningful analysis, such as the one suggested by Sydney grammar.

Generalization 3.2.1 (Merging of influential clauses)**.** When the top clause has an influential process and the lower (embedded) one has any of the other processes, then the lower one shall be enriched with aspectual features that can be derived from the top one.

This rule of thumb is described in Generalization 3.2.1. Of course, this raises a set of problems that are worth investigating. Firstly, one should investigate the connections and mappings between the influential process system network described in Cardiff grammar and the system of verbal group complex described in Sydney grammar (Halliday & Matthiessen 2013: p.589). Secondly, one should investigate how this merger impacts the syntactic structure.

The benefits of such a merger leads to an increased comprehensiveness, not only of the transitivity analysis – demonstrated by the examples in Tables 3.1 and 3.2 – but also of the modal assessment that includes modality, as demonstrated by the Examples 7 and 8.

(7) *I think* I've been pushed forward; *I don't really know*, (Halliday & Matthiessen 2013: p.183)

(8) *I believe* Sheridan once said you would've made an excellent pope. (Halliday & Matthiessen 2013: p.182)

Examples 7 and 8 represent cases when the modal assessment of the lower clause is carried on by the higher one. In both examples, the higher clause can be replaced by the modal verb *maybe* or the adverb *perhaps*.

3.2.2 Nominal, Quality, Quantity and other groups of Cardiff grammar: from syntactically towards semantically sound analysis

Cardiff unit classes are semantically motivated as compared to more syntactic ones in Sydney grammar. This has been presented in Sections **??** and discussed in **??**.

For instance, Nominal class structure proposed in Cardiff grammar (discussed in Section **??**), uses elements that are more semantic in nature (e.g. various types of determiners: representational, quantifying, typic, partitive etc.) than the syntactic one offered in Sydney grammar (e.g. only deictic determiner). To do this shift we need to think of two problems: (a) how to detect the semantic head of the nominal units and (b) how to craft (if none exists) a lexical-semantic resources to help determining potential functions (structural elements) for each lexical item in the nominal group. In my view building lexical-semantic resources asked at point (b) bears actually a solution for point (a) as well.

I need to stress that some existing lexical resources such as WordNet (Miller 1995) and/or FrameNet(Baker et al. 1998) could and most likely are suitable for fulfilling the needs at point (b) but the solution is not straight forward and further adaptations need to be done for the context of SFL.

The same holds for Adverbial and Adjectival groups (discussed in Section **??**) which, in Cardiff grammar, are split into the Quality and Quantity groups. The existent lexical resources such as such as WordNet (Miller 1995) and/or FrameNet(Baker et al. 1998) combined with the delicate classification proposed by Tucker (1997) (and other research must exist on adverbial groups of which I am not aware at the moment) can yield positive results in parsing with Cardiff unit classes.

Just like in the case of verb groups discussed in previous section, moving towards semantically motivated unit classes, as proposed in Cardiff grammar, would greatly benefit applications requiring deeper natural language understanding.

3.2.3 Taxis analysis and potential for discourse relation detection

Currently Parsimonious Vole parser implements a simple taxis analysis technique based on patterns represented as regular expressions.

In the Appendix is listed a database of clause taxis patterns according to systematization in IFG 3 (Halliday & Matthiessen 2004). Each relation type has a set of patterns ascribed to it which represent clause order and presence or absence of explicit lexical markers or clause features.

Then, in taxis analysis process, each pair of adjacent clauses in the sentence is tested for compliance with every pattern in the database. The matches represent potential manifestation of the corresponding relation.

Currently this part of the parser has not been tested and it remains a highly desirable future work. Further improvements and developments can be performed based on incremental testing and corrections of the taxis pattern database.

This work can be extended to handle relations between sentences taking on a discourse level analysis which is perfectly in line with the Rhetorical Structure Theory (RST) (Mann & Thompson 1988; Mann et al. 1992).

To increase the accuracy of taxis analysis, I believe the following additional elements shall be included into the pattern representation: Transitivity configurations including process type and participant roles, co-references resolved between clauses/sentences and Textual metafunction analysis in terms of Theme/Rheme and eventually New/Given.

3.2.4 Towards speech act analysis

As Robin Fawcett explains (Fawcett 2011), Halliday's approach to MOOD analysis differs from that of Transitivity in the way that the former is not "pushed forward towards semantics" as the latter is. Having a semantically systematised MOOD system would take the interpersonal text analysis into a realm compatible with Speech Act Theory proposed by Austin (1975) or its latter advancements such as the one of Searle (1969) which, in mainstream linguistics, are placed under the umbrella of pragmatics.

Halliday proposes a simple system of speech functions (Halliday & Matthiessen 2013: p.136) which Fawcett develops into a quite delicate system network (Fawcett 2011). It is worth exploring ways to implement Fawcett's latest developments and because the two are not conflicting but complementing each other, one could use Hallidayan MOOD system as a foundation, especially that it has already been implemented and described in the current work.

3.2.5 Process Types and Participant Roles

The PTDB (Neale 2002) is the first lexical-semantic resource for Cardiff grammar Transitivity system. Its usability in the original form doesn't go beyond that of a resource to be consulted by linguists in the process of manual analysis. It was rich in human understandable comments and remarks but not formal enough to be usable by computers. In the scope of current work the PTDB has been cleaned and brought into a machine readable form but this is far from it's potential as a lexical-grammatical resource for semantic parsing.

In the mainstream computational linguistics, there exist several other lexical-semantic resources used for Semantic Role Labelling (SRL) such as FrameNet (Baker et al. 1998), VerbNet (Kipper et al. 2008). Mapping or combining PTDB with these resources into a new one would yield benefits for both sides combining strengths of each and covering their shortcomings.

Combining PTDB with VerbNet for example, would be my first choice for the following reasons. PTDB is well semantically systematised according to Cardiff Transitivity system however it lacks any links to syntactic manifestations. VerbNet, on the other hand contains an excellent mapping to the syntactic patterns in which each verb occur, each with associated semantic representation of participant roles and some first order predicates. However, the systematization of frames and participant roles could benefit from a more robust basis of categorisation. Also the lexical coverage of VerbNet is wider than that of PTDB.

Turning towards resources like FrameNet and WordNet could bring other benefits. For example FrameNet has a set of annotated examples for every frame which, after transformation into Transitivity system, could be used as a training corpus for machine learning algorithms. Another potential benefit would be generating semantic constrains (for example in terms of WordNet (Miller 1995) synsets or GUM (Bateman et al. 1995, 2010) classes) for every participant role in the system.

PTDB can benefit from mappings with GUM ontology which formalises the experiential model of Sydney school. First by increasing delicacy (at at the moment it covers only three top levels of the system) and second by importing constraints on process types and participant roles from Nigel grammar (Matthiessen 1985). To achieve this, one would have to first map Cardiff and Sydney Transitivity systems and second extract lexical entries from Nigel grammar along with adjacent systemic selections.

3.2.6 Reasoning with systemic networks

Systemic networks are a powerful instrument to represent paradigmatic dimension of language. Besides hierarchies they can include constraints on which selections can actually go together or a more complex set of non hierarchical selection interdependencies. Moreover systemic choices can be also accompanied by the realization rules very useful for generation purpose but they could potentially be used in parsing as well.

In current work system networks are used solely for representation purposes and what would be highly desirable is to enable reasoning capabilities for constraint checking on systemic selections and on syntactic and semantic constituency. For example one could as whether a certain set of features are compatible with each other, or provided a systemic network and several feature selections what would be the whole set of system choices, or being in a particular point in the system network what are the possible next steps towards more delicate systemic choices, or for a particular choice or set of choices what should be present or absent in the constituency structure of the text and so on. All these questions could potentially be resolved by a systemic reasoner.

Martin Kay is the first to attempt formalization of systemics that would become known as Functional Unification Grammar (FUG) (Kay 1985). This formalization caught on popularity in other linguistic domains such as HPSG, Lexical Functional Grammars and Types Feature Structures. One could look at what has been done and adapt the or build a new reasoning system for systemic networks.

With the same goal in mind, one could also look at existing reasoners for different logics and attempt an axiomatization of the systemic networks; and more specifically one could do that in Prolog language or with description logics (DL) as there is a rich set of tools and resources available in the context of Semantic Web.

3.2.7 Creation of richly annotated corpus with all metafunction: interpersonal, experiential and textual

In order to evaluate a parser, a gold standard annotation corpus is essential. The bigger the corpus, covering various the text genres, the more reliable are the evaluation results. A corpus can as well be the source of grammar or distribution probabilities for structure element and potential filling units as is explored by Day (2007), Souter (1996) and other scholars in Cardiff. Moreover such a corpus can also constitute the training data set for a machine learning algorithm for parsing.

A corpus of syntactically annotated texts with Cardiff grammar already exists but, from personal communication with Prof. Robin Fawcett, it is not yet been released to

public because it is considered still incomplete. Even so this corpus covers only the constituency structures and what I would additionally find very useful, would be a set of systemic features of the constituting units covering a full SFG analysis in terms of experiential, interpersonal and textual metafunctions; and not only the unit class and the element it fills.

A small richly annotated set of text had been created in the scope of the current work for the purpose of evaluating the parser. However it is by far not enough to offer a reliable evaluation. Therefore it is highly desirable to create one.

To approach this task one could use a systemic functional annotation tool such as UAM Corpus Tool (O'Donnell 2008a,b) developed and still maintained by Mick O'Donnell or any other tool that supports segment annotation with systemic network tag set structure.

To aid this task one could bootstrap this task by converting other existing corpuses such as Penn Treebank. This task had been already explored by Honnibal in 2004; 2007.

3.2.8 The use of Markov Logics for pattern discovery

Markov Logic (Richardson & Domingos 2006; Domingos et al. 2010) is a probabilistic logic which applies ideas of Markov network to first order logic enabling uncertain inference. What is very interesting about this logics is that tools implementing it have learning capabilities not only of formulas weights but also of new logical clauses.

In current approach I am using graph patterns matching technique to generate a rich set of features for the constituent units. However creating those patterns is a tremendous effort.

Since, graph patterns can be expressed via first order functions and individuals, and assuming that there would already exist a richly annotated corpus, the Markov Logic instruments (for example Alchemy[1], Tuffy[2] and others) can be employed to inductively learn such patterns from the corpus.

This approach resembles the Vertical Strips (VS) of O'Donoghue (1991). The similarity is the probabilistic learning of patterns from the corpus. The difference is that VS patterns are syntactic segment chains from the root node down to tree leafs while with ML more complex patterns can be learned independently of their position in the syntactic tree. Moreover such patterns can be bound to specific feature set.

[1] http://alchemy.cs.washington.edu/
[2] http://i.stanford.edu/hazy/hazy/tuffy/

3.3 A final word

References

Allen, James F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11). 832–843. doi:10.1145/182.358434. <http://portal.acm.org/citation.cfm?doid=182.358434>.

Austin, J L. 1975. *How to do things with words*, vol. 3 (Syntax and Semantics 1). Harvard University Press.

Baker, Collin F, Charles J Fillmore & John B Lowe. 1998. The Berkeley FrameNet Project. In Christian Boitet & Pete Whitelock (eds.), *Proceedings of the 36th annual meeting on association for computational linguistics*, vol. 1 ACL '98, 86–90. University of Montreal Association for Computational Linguistics. doi:10.3115/980845.980860. <http://portal.acm.org/citation.cfm?doid=980845.980860>.

Bateman, John A, Renate Henschel & Fabio Rinaldi. 1995. The Generalized Upper Model . Tech. rep. GMD/IPSI. <http://www.fb10.uni-bremen.de/anglistik/langpro/webspace/jb/gum/gum-2.pdf>.

Bateman, John A, Joana Hois, Robert Ross & Thora Tenbrink. 2010. A linguistic ontology of space for natural language processing. *Artificial Intelligence* 174(14). 1027–1071. doi:10.1016/j.artint.2010.05.008. <http://linkinghub.elsevier.com/retrieve/pii/S0004370210000858>.

Bondy, John Adrian, Uppaluri Siva Ramachandra Murty et al. 1976. *Graph theory with applications*, vol. 290. Citeseer.

Day, Michael David. 2007. *A Corpus-Consulting Probabilistic Approach to Parsing : the CCPX Parser and its Complementary Components*: Cardiff University dissertation.

Domingos, Pedro, Stanley Kok, Daniel Lowd & Hoifung Poon. 2010. Markov Logic. *Journal of computational biology a journal of computational molecular cell biology* 17(11). 1491–508. doi:10.1089/cmb.2010.0044. <http://www.ncbi.nlm.nih.gov/pubmed/21685052>.

Fawcett, Robin. 2000. *A Theory of Syntax for Systemic Functional Linguistics*. John Benjamins Publishing Company paperback edn.

Fawcett, Robin P. 2011. A semantic system network for MOOD in English (and some complementary system networks).

Gale, D. & L. S. Shapley. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly* 69(1). 9–15. <http://www.jstor.org/stable/2312726>.

Gusfield, Dan & Robert W. Irving. 1989. *The stable marriage problem: Structure and algorithms*. Cambridge, MA, USA: MIT Press.

Halliday, Michael A. K. 2003. Systemic theory. In Michael A. K. Halliday & Jonathan J. Webster (eds.), *On language and linguistics. Volume 3 of collected works of M.A. K. Halliday*, 490. New York: Continuum.

Halliday, Michael A.K. & Christian M.I.M. Matthiessen. 2013. *An Introduction to Functional Grammar (4^{th} Edition)*. Routledge 4th edn.

Halliday, Michael A.K. & M.I.M. Matthiessen, Christian. 2004. *An introduction to functional grammar (3^{rd} Edition)*. Hodder Education.

Honnibal, Matthew. 2004. Converting the Penn Treebank to Systemic Functional Grammar. *Technology* 147–154.

Honnibal, Matthew & Jr James R Curran. 2007. Creating a systemic functional grammar corpus from the Penn treebank. *Proceedings of the Workshop on Deep ...* 89–96. doi:10.3115/1608912.1608927. <http://dl.acm.org/citation.cfm?id=1608927>.

Iwama, Kazuo & Shuichi Miyazaki. 2008. A survey of the stable marriage problem and its variants. In *International conference on informatics education and research for knowledge-circulating society*, 131–136. IEEE.

Kay, Martin. 1985. Parsing In Functional Unification Grammar. In D.Dowty, L. Karttunen & A. Zwicky (eds.), *Natural language parsing*, Cambridge University Press.

Kipper, Karin, Anna Korhonen, Neville Ryant & Martha Palmer. 2008. A large-scale classification of English verbs. *Language Resources And Evaluation* 42(1). 21–40. doi:10.1007/s10579-007-9048-2.

Mann, William C., Christian M. I. M. Matthiessen & Sandra A. Thompson. 1992. Rhetorical Structure Theory and Text Analysis. In William C Mann & Sandra A Thompson (eds.), *Discourse description: Diverse linguistic analyses of a fund-raising text*, vol. 16 Pragmatics & Beyond New Series, 39–79. John Benjamins Publishing Company.

Mann, William C & Sandra A Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text* 8(3). 243–281. doi:10.1515/text.1.1988.8.3.243.

Marneffe, Marie-Catherine & Christopher D. Manning. 2008. Stanford typed dependencies manual. Tech. Rep. September Stanford University. <http://nlp.stanford.edu/downloads/dependencies{_}manual.pdf>.

Matthiessen, M.I.M., Christian. 1985. The systemic framework in text generation: Nigel. In James Benson & Willian Greaves (eds.), *Systemic perspective on Discourse, Vol I*, 96–118. Ablex.

Miller, George A. 1995. WordNet: a lexical database for English.

Neale, Amy C. 2002. More Delicate TRANSITIVITY: Extending the PROCESS TYPE for English to include full semantic classifications. Tech. rep. Cardiff University.

O'Donnell, Michael. 1993. Reducing Complexity in Systemic Parser. In *Proceeedings of the third international workshop on parsing technologies*, .

O'Donnell, Mick. 2008a. Demonstration of the UAM CorpusTool for text and image annotation. In *Proceedings of the acl-08:hlt demo session* June, 13–16.

O'Donnell, Mick. 2008b. The UAM CorpusTool: Software for Corpus Annotation and Exploration. In Bretones Callejas & Carmen M. (eds.), *Applied linguistics now: Understanding language and mind*, vol. 00, 1433–1447. Universidad de Almería.

O'Donoghue, Tim. 1991. The Vertical Strip Parser: A lazy approach to parsing. Tech. rep. School of Computer Studies, University of Leeds.

Pollard, Carl & Ivan Sag. 1987. *Information-Based Syntax and Semantics*. CSLI.

Richardson, Matthew & P. Domingos. 2006. Markov logic networks. *Machine learning* 62(1-2). 107–136. doi:10.1007/s10994-006-5833-1.

Searle, John R. 1969. *Speech Acts: An Essay in the Philosophy of Language*, vol. 0. Cambridge University Press. <http://books.google.com/books?id=t3{_}WhfknvF0C{&}pgis=1>.

Souter, David Clive. 1996. *A Corpus-Trained Parser for Systemic-Functional Syntax*: University of Leeds Phd. <http://etheses.whiterose.ac.uk/1268/>.

Tucker, Gordon H. 1997. A functional lexicogrammar of adjectives. *Functions of Language* 4(2). 215–250.

West, Douglas Brent et al. 2001. *Introduction to graph theory*, vol. 2. Prentice hall Upper Saddle River.

SFL Syntactic Overview

.1 Cardiff Syntax

Elements found in all groups: Linker (&), Inferer (I), Starter (st), Ender (e)
Units: Sentence (Σ), Clause (Cl), Nominal Group (ngp), Prepositional Group (pgp), Quality Group (qlgp), Quantity Group (qtgp), Genitive Cluster (gencl)

.1.1 Clause

Relative Order of Elements in the Unit Structure:
& |B |L |F |A |C |O |S |O |N |A |I |X |M |Mex |C |A |V |E
Clause May fill: Σ (85%), C (7%), A (4%), Q (2%), f (0.5%), s, qtf, S, m, cv, po
Elements of the Clause: Adjunct (A), Binder (B), Complement (C), Formulaic Element (F), Infinitive Element (I), Let Element (L),Main Verb (M), Main Verb Extension (Mex), Negator (N), Operator (O), Subject (S), Vocative (V), Auxiliary Verb (A), X extension (Xex), Linker (&), Starter (St), Ender(E)

.1.2 Nominal Group

Possible Relative Order of Elements in the Unit Structure:
& |rd |v |pd |v |qd |v |sd |v |od |v |td |v |dd |m |h |q |e
Filling probabilities of the ngp: S (45%), C (32%), cv (15%), A (3%), m (2%), Mex, V, rd, pd, fd, qd, td, q, dt, po
Elements of the ngp: Representational determiner (rd), Selector (v), Partitive Determiner (pd), Fractionative Determiner (fd), Quantifying Determiner (qd), Superlative Determiner (sd), Ordinative Determiner (od), Qualifier-Introducing Determiner (qid), Typic Determiner (td), Deictic Determiner (dd), Modifier (m), Head (h), Qualifier (q)

.1.3 Prepositional Group

Possible Relative Order of Elements in the Unit Structure:
& |pt |p |cv |p |e
Filling Probabilities of the pgp: C (55%), a (30%), q (12%), s (2%) Mex, S, cv, f, qtf
Elements of the pgp: Preposition (p), Prepositional Temperer (pt), Completive (c)

.1.4 Quality Group

Possible Relative Order of Elements in the Unit Structure:
& |qld |qlq |et |dt |at |a |dt |s |f |s |e
Filling probabilities of the qgp: c (38%), m (36%), A (24%), sd (0.5%), Mex, Xex, od, q, dt, at, p, S
Elements of the qlgp: Quality Group Deictic (qld), Quality Group Quantifier (qlq), Emphasizing Temperer (et), Degree Temperer (dt), Adjunctival Temperer (at), Apex (a), Scope (s), Finisher (f)

.1.5 Quantity Group

Possible Relative Order of Elements in the Unit Structure:
ad |am |qtf |e **Filling probabilities of the qtgp:** qd (85%), A (8%), dt (6%), B, p, ad, fd, sd **Elements of the qtgp** Adjustor (ad), Amount (am), Quantity Finisher (qf)

.1.6 Genitive Cluster

Possible Relative Order of Elements in the Unit Structure:
& |po |g |o |e
Filling probabilities of the gencl: dd (99%), h, m, qld
Elements of the gencl: Possessor (po), Genitive Element (g), Own Element (o)

.2 Sydney Syntax

.2.1 Logical

Possible Relative Order of Elements in the Unit Structure:
Pre-Modifier |Head |Post-Modifier

.2.2 Textual

Possible Relative Order of Elements in the Clause Structure:
Theme |Rheme
New |Given |New

.2.3 Interactional

Possible Relative Order of Elements in the Clause Structure:
Residue |Mood |Residue |Mood tag
Adjunct |Complement |Finite |Subject |Finite |Adjunct |Predicator |Complement| Adjunct

.2.4 Experiential

Possible Relative Order of Elements in the Clause Structure:
Circumstance |Participant |Circumstance |Process| Participant |Circumstance
Possible Relative Order of Elements in the Nominal Group Structure:
Deictic |Numerative |Epithet | Classifier| Thing |Qualifier
Possible Relative Order of Elements in the Verbal Group Structure:
Finite |Marker |Auxiliary |Event
Possible Relative Order of Elements in the Adverbial and Preposition Group Structure: Modifier |Head |Post-Modifier
Possible Relative Order of Elements in the Prepositional Phrase Structure:
Predicator |Complement
Process |Range

.2.5 Taxis

Possible Relative Order of Elements in the Parataxis Structure:
Initiating |Continuing
Possible Relative Order of Elements in the Hypoataxis Structure:
Dependent |Dominant |Dependent

Stanford Dependency schema

The Stanford dependency relations as defined in Stanford typed dependencies manual (Marneffe & Manning 2008)

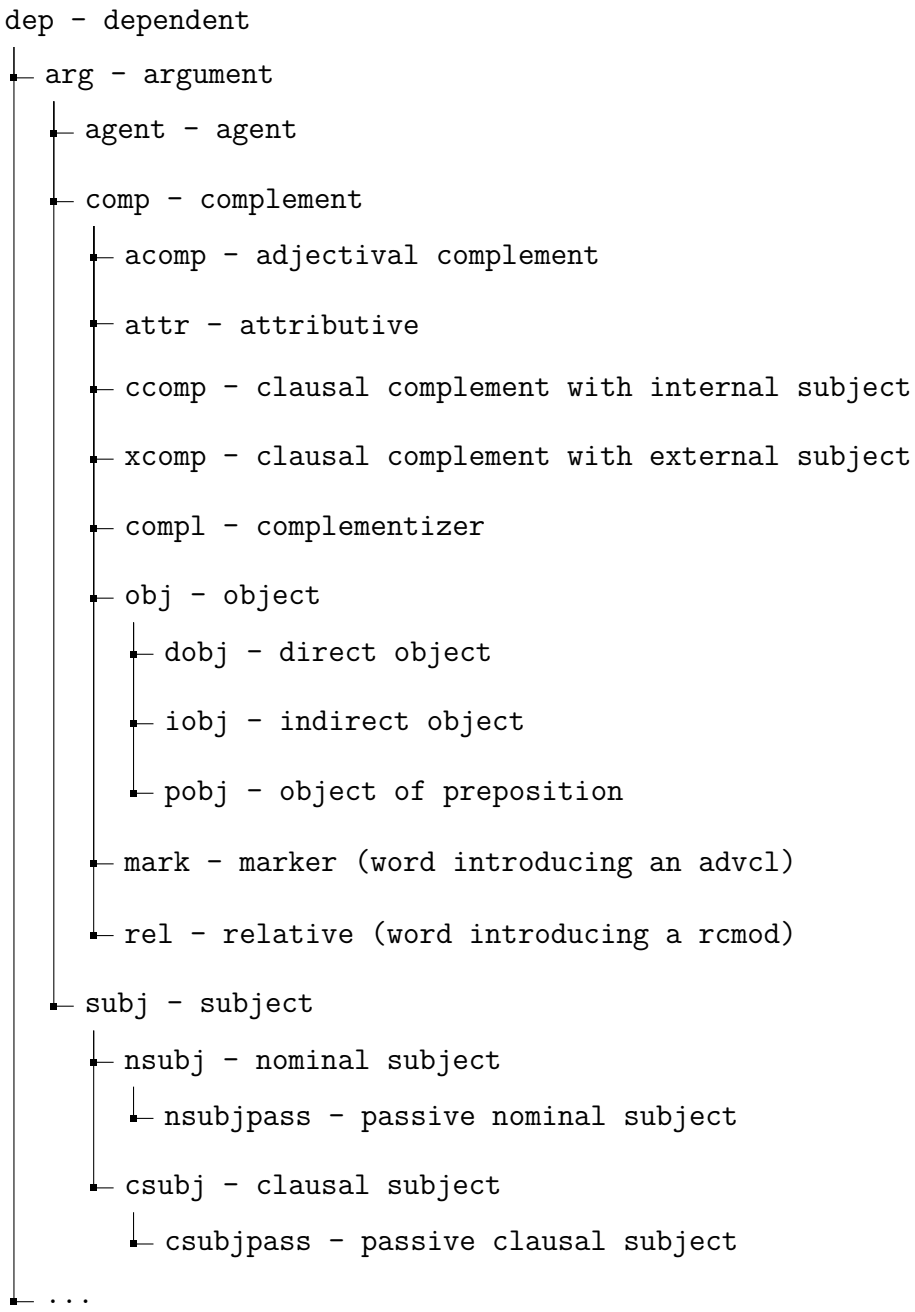


Fig. 1 The Stanford dependency scheme - part one

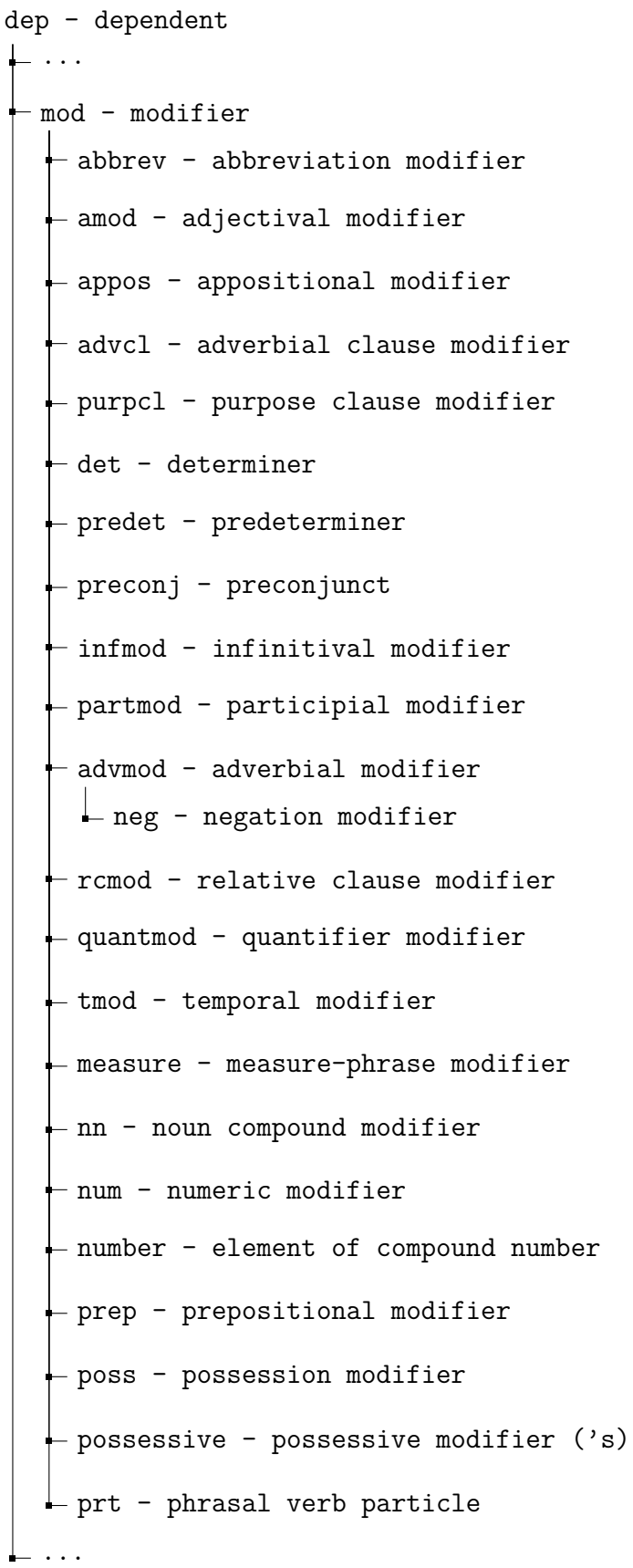


Fig. 2 The Stanford dependency scheme - part two

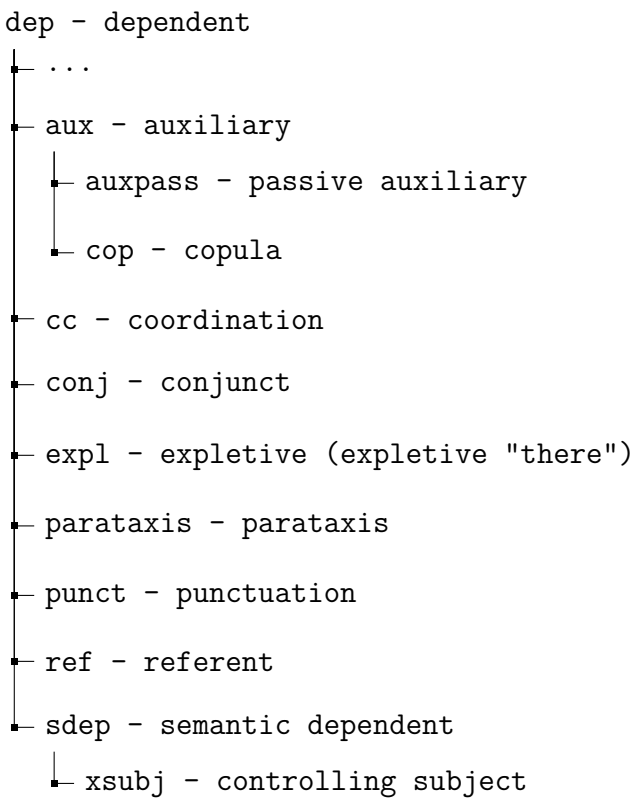


Fig. 3 The Stanford dependency scheme - part three

Penn treebank tag-set

Tag	**Description**	**Example**
CC	conjunction, coordinating	and, or, but
CD	cardinal number	five, three, 13%
DT	determiner	the, a, these
EX	existential there	there were six boys
FW	foreign word	mais
IN	conjunction, subordinating or preposition	of, on, before, unless
JJ	adjective	nice, easy
JJR	adjective, comparative	nicer, easier
JJS	adjective, superlative	nicest, easiest
LS	list item marker	
MD	verb, modal auxillary	may, should
NN	noun, singular or mass	tiger, chair, laughter
NNS	noun, plural	tigers, chairs, insects
NNP	noun, proper singular	Germany, God, Alice
NNPS	noun, proper plural	we met two Christmases ago
PDT	predeterminer	both his children
POS	possessive ending	's
PRP	pronoun, personal	me, you, it
PRP$	pronoun, possessive	my, your, our
RB	adverb	extremely, loudly, hard
RBR	adverb, comparative	better
RBS	adverb, superlative	best
RP	adverb, particle	about, off, up
SYM	symbol	%
TO	infinitival to	what to do?
UH	interjection	oh, oops, gosh
VB	verb, base form	think
VBZ	verb, 3rd person singular present	she thinks
VBP	verb, non-3rd person singular present	I think
VBD	verb, past tense	they thought
VBN	verb, past participle	a sunken ship
VBG	verb, gerund or present participle	thinking is fun
WDT	wh-determiner	which, whatever, whichever
WP	wh-pronoun, personal	what, who, whom
WP$	wh-pronoun, possessive	whose, whosever
WRB	wh-adverb	where, when
.	punctuation mark, sentence closer	.;?*
,	punctuation mark, comma	,
:	punctuation mark, colon	:
(	contextual separator, left paren	(
)	contextual separator, right paren	)

Table 3 Penn Treebank tag set