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

March 2018

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
March 2018

Acknowledgements

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 On AI, Computational Linguistics, universe and everything

In 1950 Allan Turing in a seminal paper (Turing 1950) published in Mind was asking if "machines can do what we (as thinking entities) can do?" He questioned what intelligence was and whether it could be manifested in machine actions indistinguishable from human actions.

He proposed the famous "Imitation Game" also known as the "Turing test" in which a machine would have to exhibit intelligent behaviour equivalent or indistinguishable from that of a human. The test was stating the following rules. The machine (player A) and a human (player B) are engaged in a written *natural language* conversation with a human judge (player C) which has to decide whether each conversation partner is human or a machine. The goal of players A and B is to convince the judge (player C) that they are human.

This game underpins the question whether "a computer, communicating over a teleprinter, (can) fool a person into believing it is human?", moreover whether it can generate human(-like) cognitive capacities (Stevan Harnad 1992). Essential parts of such cognitive capacities and intelligent behaviour that the machine needs to exhibit are of course the linguistic competences of comprehension (or "understanding") and generation of "appropriate" responses (for a given input from the judge C).

Artificial Intelligence (AI) field was born from dwelling on Turing's questions. The term was coined by McCarthy for the first time in 1955 referring to the "science and engineering of making intelligent machines" (McCarthy et al. 2006).

The general tendency is to program machines to do with language what humans do. Various fields of research contribute to this goal. Linguistics, amongst others, contributes with theoretical frameworks systematizing and accounting language in terms of for morphology, phonology, syntax, semantics, discourse or grammar in general. In computer science are developed increasingly more efficient algorithms and machine learning techniques. Computational linguistics provides ingenious methods of encoding linguistically motivated tasks in terms of formal data structures and computation goal. In addition, specific algorithms and heuristics operating within reasonable amounts of time with satisfiable levels of accuracy are tailored to accomplish those linguistically motivated tasks.

Computational Linguistics (CL) mentioned in 1950 in the context of automatic translations (Hutchins 1999) of Russian text into English stared developing before the field of Artificial Intelligence. Only a few years later CL became a sub-domain of AI as an interdisciplinary field dedicated to developing algorithms and computer software for intelligent processing of text (leaving the very hard questions of intelligence and human cognition somehow aside that up to now still need massive inputs on human mind from cognitive, psycho-linguistic and other related sciences). Besides *machine translation* CL incorporates a broader range of tasks such as *speech synthesis and recognition, text tagging, syntactic and semantic parsing, text generation, document summarisation, information extraction*, etc.

This thesis contributes to the field of CL and more specifically it is an advancement in *Natural Language Parsing* (NLP), one of the central CL tasks informally defined as the process of transforming a sentence into (rich) machine readable syntactic and semantic structure(s). Developing a program to automatically analyse the text in terms of such structures by involving computer science and artificial intelligence techniques is a task pursued for several decades and still continues to be a major challenge today. This is especially so when the target is a *broad language coverage* (?) and even more when the desired analysis goes beyond simple syntactic structures towards richer functional and/or semantic descriptions useful in the latter stages of *Natural Language Understanding* (NLU).

In computational linguistics, broad coverage natural language components now exist for several levels of linguistic abstraction, ranging from tagging and stemming, through syntactic analyses to semantic specifications. In general, the higher the degree of abstraction, the less accurate the coverage becomes and the richer the linguistic description the slower the parsing process is performed.

These working components are already widely used to enable humans to explore and exploit large quantities of textual data for purposes that vary from the most theoretical ones such as understanding how language works or the relation between form and meaning, to very pragmatic purposes such as developing systems with natural language interfaces, machine translation, document summarising, information extraction and question answering systems and that is just to name a few.

These software programs originally were designed by and for the domain experts but over time the fruits of the technological advancement became available to millions of ordinary people. In a world as ours, where technology is so ubiquitous and pervasive into almost all aspects of our life, natural language processing and understanding becomes of great value and importance regardless whether it materializes as a spell-checker, (not so) clever machine translation, voice controlled car or phone and so on.

1.2   The goal of the thesis

This thesis aims at a reliable modular method for parsing unrestricted English text into a feature rich constituency structure using Systemic Functional Grammars (SFGs).

Before describing the parsing method, the following aspects need to be clarified first: the theoretical framework and its descriptive power, the depth and meaningfulness of the analysis, the computational complexity of the process and the level of accuracy and how it is measured. I address each of these aspects in the following chapters and before advancing any further I would like to illustrate through an example of what parsing is.

1.3   An example of Systemic Functional analysis

Traditional linguistic teaches us how to do the syntactic analysis of a sentence. So let's consider Example 1 in order to perform one. We need to focus on clustering words together into constituents guided by the intuitive rule "which word stands closest to another one" within the sentence. The word "He" is closest to "gave" and it seems that they go together especially that if we try to group "He" and "the" they do sense at all. Then, using sequential proximity criteria, "gave" must be related to "the" but they do not stick together at all it is rather "the" and "cake" that are a unity. So "the" has a stronger relation to "cake" than "gave". But actually "cake" seems connected to "gave" and not the other way around, so the direction of connection seems to matter just as much as its strength. So we see that the sequential proximity sometimes indicated

relatedness between words but often it does not. The strength and direction as well are crucial and tt is some sort of meaning making that governs the grouping process. This grouping into syntactic constituents can be expressed using bracketed notation as in Example 2.

(1)   He gave the cake away.

(2)   ((He) (gave) ((the) (cake)) (away) (.))
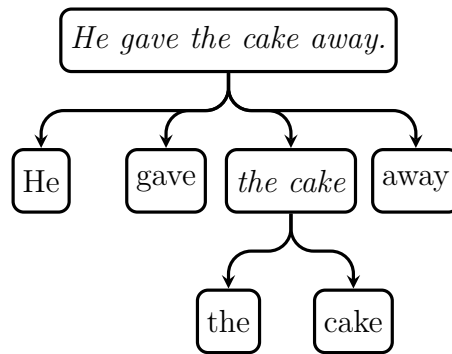
Fig. 1.1 Representation of the Example 2 as constituency tree

Figure 1.1 depicts the constituency division of the clause which is identical to the bracket notation in Example 2. The nodes represent grammatical constituents and the edges stand for the part-whole composition. This constitutes the simplest structure that could be expected from a parser without any specification of constituent class, function or any other grammatical functions. We know as well from works on parsing with formal generative grammars that such composition can always be expressed as a tree (or parse tree).

Each constituent can be decorated with its grammatical features . For example the word "he", we know, is third person pronoun, masculine gender and singular number; or the word "gave" is a verb and the predicate of the sentence, and so on. The structure in Figure 1.2 depicts a syntactic constituency tree in which every node is richly decorated with syntactic and semantic features. The blue part of each node represents grammatical class and function fundamental for establishing a valid constituency structure; the red part represents the semantic functions (called Transitivity in SFL); and the green part other grammatical features. In practice, the feature set is much richer than what nodes in Figure 1.2 carry, the current limitation being simply the space constraints. Generating automatically graphs like the one in Figure 1.2 is the purpose of the current work.
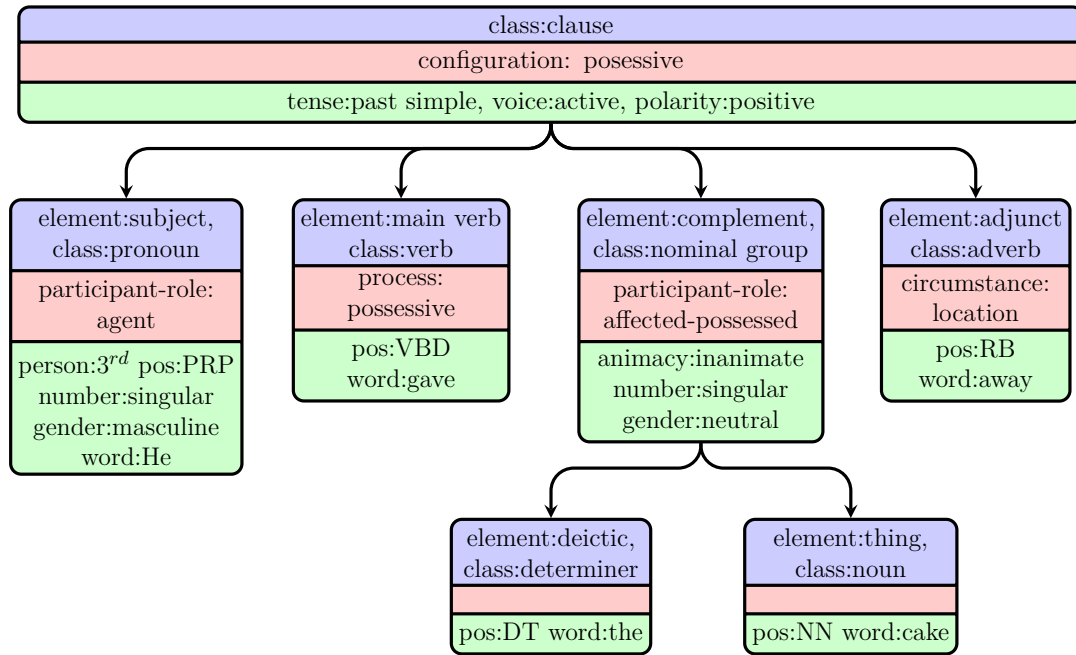
Fig. 1.2 Representation of Example 2 as feature rich constituency graph

1.4 The linguistic framework

In this thesis I chose a Systemic Functional Linguistic (SFL) stance because of its versatility in describing and explaining in *multiple semiotic dimensions* (Halliday 2003b) (i.e. paradigmatic, syntagmatic, meta-functional, stratification and instantiation dimensions) and at different *delicacy levels* of the *lexico-grammatical cline* (Halliday 2002; Hasan 2014) the (occurring) linguistic phenomena.

SFL regards language as a social semiotic system where any act of communication is regarded as a conflation of *linguistic choices* available in a particular language. Choices, which are organised on a paradigmatic rather than structural axis and materialised as *system networks*. Moreover, in SFL perspective language has evolved to serve particular *functions* influencing their the structure and organisation of the language. However, being around the paradigmatic systems, the functional organization differs significantly from other functional approaches (Prague School, Lexical Functional Grammar, Head-driven phrase structure grammar, Role and Reference Grammar, Functional Discourse Grammar and others). Halliday refers to the language functions as metafunctions or lines of meaning offering a trinocular perspective on language through *ideational*, *interpersonal* and *textual* metafunctions.

In SFL, language is first of all an interactive action serving to enact social relations under the umbrella of *interpersonal metafunction*.Then it is a medium to express the

embodied human experience of the inner and outer worlds via *ideational metafunction.* Finally the two weave together into a coherent discourse flow whose mechanisms are explained through the *textual metafunction.*

To account for the complexity and phenomenological diversity of human language the SFL theory provides descriptions along *syntagmatic, (meta)functional, paradigmatic, stratification and instantiation axes.*

There are two models of SFG: *Sydney Grammar* (SG) Halliday & Matthiessen (2013) developed by Halliday and Matthiessen, the founding fathers of *Systemic Functional Linguistics* (SFL), and *Cardiff Grammar* (CG) Fawcett (2008), an extension and a simplification of Sydney Grammar. Each of the two grammars has advantages and shortcomings which I present in analyse and select based on theoretical soundness and suitability to the goals of the current project.

Cardiff and Sydney grammars had been used as language models in natural language generation projects within the broader contexts of social interaction. Some researchers attempted to reuse the grammars for the purpose of syntactic parsing within the borders of NL generation coverage, such as that of Kasper (1988), O'Donoghue (1991), O'Donnell (1993), Souter (1996), Day (2007).

1.5   The SFG complexity problem

Bateman (2008) thoroughly explains the reasons for such tremendous complexity after the attempts of Kasper (1988), Kay (1985), O'Donoghue (1991), O'Donnell (1993) and Day (2007), just a few to mention, none of which managed to parse broad coverage English with full SFG and without aid of some sort. Each had to accept limitations either in grammar or language size and eventually using simpler syntactic trees as a starting point of the parsing process. So what is it about?

Automatic analysis of text can be seen as a problem of searching though the space of possible solutions for an appropriate or even optimal solution. Here we speak of the Systemic Functional Grammar as a linguistic resource that shapes the search space and the way it access to that space is available. The systemic lexicogrammar is organised paradigmatically and was proven to be a good structure for natural language generation task but it turns out to be unusable for the reverse problem, that of natural language analysis. The principal issues is that of handling the *search space* leading back to Halliday's question "How big is a grammar?" (Halliday 1966).

The size of the search space defined by a grammar depends on the number of system networks and on the kind of connectivity and cross-classification it provides.

For example given 50 system networks, the size of the search space lies somewhere between 51 and 2^{50}. This nevertheless is not such a big deal in the case of generation, as Halliday (1996) says that the "number of choice points [...] is actually rather small" as only few of the actual possibilities produced by a system network need to be explored when generating a clause. "Possible feature selections become relevant only when they are revealed to be relevant by prior paradigmatic choices and it is only those alternatives that need to be considered"(Bateman 2008).

Analysis is not symmetric with generation and the paradigmatic context of choice that is available during generation is no longer accessible in parsing. It is not known any longer which features of a systemic network are relevant and which are not. That is: in generation, the simple traversal of the network finds only the compatible choices because that is what the network leads to; whereas in analysis it is not evident in advance which path to follow therefore the task is virtually to explore entire search space in order to discover which features apply to the text (Bateman 2008).

In the analysis task first difficulty that needs to be addressed is discovering from a sequence of words what possible groups are combinable into grammatical groups, phrases or clauses. This is a task of bridging a sequence of words input and the grammatical description of *instantial syntagmatic organizations* involving *configurations of grammatical functions*. In a second stage these grammatical functions can serve as paradigmatic context for further traversing the system network and extend to the full set of systemic features. Moreover they will play a crucial role in restricting and organising the search space for relevant and applicable network parts during analysis task.

Addressing the gap of easily accessible syntagmatic account within SFG framework, can be done by first, providing information about what grammatical function operate at each rank, second which grammatical functions can be filled by which classes of units and third by providing relative and absolute account of ordering within each unit structure. This sort of information can guide building of a constituency backbone structure. As a second stage, as mentioned before, the unit classes and grammatical functions can operate as "hooks" on system network to guide the traversal in the same way the paradigmatic context available in the generation process.

Alternatively the problem of structure construction can be outsources as parsing with other grammars especially that there has been a lot of progress recently. Then the problem changes into creating a transformation mechanism to obtain the SF constituency structure rather than build it from scratch. Starting the SFG parsing

process from a simple syntactic tree reduces the computational complexity by providing a set of reliable selections within the system network.

The second stage of constituent enrichment by network traversal can be further aided by checking an arbitrary set of patterns for preselecting even more features recoverable via lexico-syntactic patterns. The pattern recognition plays an essential role in current parsing method for fleshing out the constituent backbone with systemic selections.

1.6 On theoretical compatibility and reuse

In the past decades there have been made significant progresses in natural language parsing framed in one or another linguistic theory each adopting a distinct perspective and set of assumptions about language. The theoretical layout and the available resources influences directly what and how is being implemented into the parser and each implementation approach encounters challenges that may or may not be common to other approaches in the same or other theories.

Pareses for one theoretical framework may face common or different problems across theories, but as well as the solutions. The successes and achievements in any school of though should be regarded as valuable cross theoretical results to the degree the links and correspondences can be established. Therefore reusing components that have been proved to work and yield "good enough results" is a strong pragmatic motivation for deriving herein described parsing method.

Present work lays first some cross theoretical correspondences and then some inter-grammatical links. It demonstrates how selected grammatical frameworks namely *Systemic Functional Grammar*, *Dependency Grammar* and *Governance & Binding Theory* relate to each other and to which degree they are compatible to undergo a conversion process and to show that simple patterns carrying grammatical information can be used to enrich syntactically and semantically the parse structures. And here is a brief motivation for selecting these frameworks.

In the last years *Dependency Grammar* (Tensiere 2015) became quite popular in natural language processing world favoured over phrase structure grammars. The grammatical lightness and the the modern algorithms implemented into dependency parsers such as Stanford Dependency Parser (Marneffe et al. 2006), MaltParser (Nivre 2006), MSTParser (McDonald et al. 2006) and Enju (Miyao & Tsujii 2005) are increasingly efficient and highly accurate.

I employ Stanford Dependencies (Marneffe & Manning 2008b,a; Marneffe et al. 2014) as a starting point which provides information about functional dependencies between words and grants direct access to the predicate-argument relations which are not readily available form the phrase structure parses and can be used off the shelf for real world applications. Regardless of being a simple grammatical framework which accounts for the syntactic relations between words, Stanford dependency grammar is structurally and functionally compatible to SFG.The account it provides for the word dependencies can be viewed also in functional terms and I expand this idea in Chapter 3. It is a much more suitable foundation for building the SFG syntactic structure than phrase-structure trees, as well as for making more delicate grammatical distinctions (a process highlighted in Section 1.9 and explained in detail in Chapter 6).

The current parsing process requires accounting for *null elements* which are not covered by the dependency grammar. As a solution I turned to a part of Chomsky's Transformational Grammar(Chomsky 1957), Government and Binding Theory (GBT) (Chomsky 1981; Haegeman 1991), to identify and create the Null Elements to support the semantic parsing. I introduce GBT and provide inter-grammatical links between towards dependency grammar in Chapter 4.

Current work is the first one to parsing with a dependency backbone, all previous ones using context-free grammars. There are other good candidates to serve as backbone (CCG, TAG, FCG, HPSG etc.) but a broad investigation of parsers and the compatibility of their linguistic theoretical frameworks to SFL is outside the scope of this thesis.

1.7 Previous works on parsing with Systemic Functional Grammars

There have been various attempts to parsing with SFGs. This section covers the most significant attempts to parse with a Systemic Functional Grammar. The firs attempt was made by Winograd (Winograd 1972) which was more than a parser, it was an interactive a natural language understanding system for manipulating geometric objects in a virtual world.

Starting from early 1980s onwards, Kay, Kasper, O'Donell and Bateman tried to parse with Nigel Grammar (Matthiessen 1985), a large and complex natural language generation (NLG) grammar for English used in Penman generation project. Other attempts by O'Donoghue (1991), Weerasinghe (1994), Souter (1996), Day (2007) aim

for corpus based probability driven parsing within the framework of COMMUNAL project starting from late 1980s.

In a very different style, Honnibal (2004); Honnibal & Curran (2007) constructed a system to convert Penn Treebank into a corresponding SFGBank. This managed to provide a good conversion from phrase structure trees into systemic functional representation covering sentence mood and Thematic constituency (a kind of analysis in SFL which is not considered in current work). Transitivity has not been covered there because of its inherently semantic nature but it is in the current work.

1.7.1   Winograd's SHRDLU

SHRDLU is an interactive program for understanding (if limited) natural language written by Terry Winograd at MIT between 1968-1970. It carried a simple dialogue about a world of geometric objects in a virtual world. The human could as the system to manipulate objects of different colours and shapes and the ask questions about what has been done or the new state of the world.

It is recognised as a landmark in natural language understanding demonstrating that a connection with artificial intelligence is possible if not solved. However, his success was not due to the use of SFG syntax but rather due to small sizes of every system component to achieve a fully functional dialogue system. Not only it was parsing the input but it was developing an interpretation of it, reason about it and generate appropriate natural language response.

Winograd combined the parsing and interpretation processes such that the semantic interpreter was actually guiding the parsing process. The knowledge of syntax was encoded in the procedures of interpretation program. He also implemented an ingenious backtracking mechanism where the the program does not simply go back, like other parsers, to try the next possible combination choice but actually takes a decision on what shall be tried next.

Having data embedded into the program procedures, as Winograd did, makes it non-scalable for example in accommodation of larger grammars and knowledge bodies and unmaintainable on the long term as it becomes increasingly difficult to make changes (Weerasinghe 1994).

1.7.2   Kasper

Bob Kasper in 1985 being involved in Penman generation project embarked on the mission of testing if the Nigel grammar, then the largest available generation grammar,

```
              NG
               |
             NGPS                            nominal-group-simplex
            /      \                                  |
        NGPS        PP  ⇨                              |
        /    \                    ┌──────────┬─────────┴──────┬──────────────────┐
    NGPRE    NGHEAD           DEICTIC    CLASSF        THING        QUALIFIER
     /\                        deictic     noun                accompanying-process
   DN  CLASSIF
```
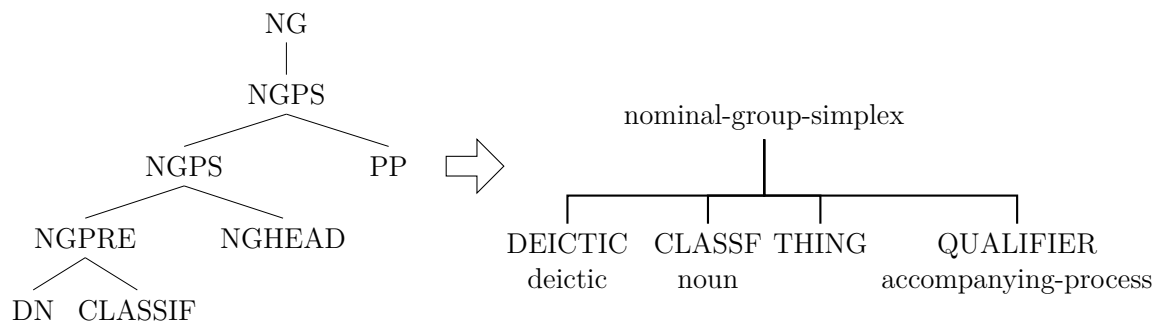
Fig. 1.3 Transformation from phrase structure into systemic constituency structure. Rule example from O'Donnell & Bateman (2005).

was suitable for natural language parsing. Being familiar with Functional Unification Grammar (FUG), a formalism developed by Kay and tested in parsing (Kay 1985) which caught on popularity in computational linguistics regardless of Kay's dissatisfaction with results, Kasper decided to re-represent Nigel grammar into FUG.

Faced with tremendous computational complexity, Kasper (1988) decided to manually create the phrase-structure of the sentences with hand-written rules which were mapped onto a parallel systemic tree structure. Kasper in 1988 was the first one to parse with a context-free backbone. He first parsed each sentence with a Phrase Structure Grammar (PSG), typical to Chomsky's Generative Transformational Linguistics Chomsky (1957). He created a set of rules for mapping the phrase structure (PS) into a parallel systemic tree like the one depicted in Figure 1.3. When all possible systemic tree were created they were further enriched using information from Nigel Grammar (Matthiessen 1985).

Once the context-free phrase-structure was created using bottom-up chart parser it was further enriched from the FUG representation of Nigel grammar. This approach to parsing is called *parsing with a context-free backbone* as phrase-structure is conveyed as simplistic skeletal analysis, fleshed out by the detail rich systemic functional grammar.

Even though Kasper's system is represents the first attempt to parse with full Hallidayan grammar, it's importance is lowered, as O'Donnell & Bateman (2005) point out, by the reliance on phrase structure grammar.

1.7.3 O'Donnell

Since 1990, Mick O'Donnell experimented with several parsers for small Systemic grammars, but found difficulty when scaling up to larger grammars. While working in EAD project, funded by Fujitsu, he recompiled a subset of Nigel grammar into two

resources: the set of possible function bundles allowed by the grammar (along with the bundles preselections) and a resource detailing which functions can follow a particular function (O'Donnell 1993, 1994).

This parser was operating without a syntactic backbone directly from a reasonable scale SFG. However when scaled to the whole Nigel grammar the system became very slow because of the sheer size of the grammar and its inherent complexity introduced by multiple parallel classifications and functional combinations - a problem well described by Bateman (2008). Then O'Donnell wrote his own grammar of Mood that was more suitable for the parsing process and less complex than the recompiled Nigel.

In 2001, while working in a Belgian company O'Donnell came to conclusion that dependency grammars are very efficient for parsing. Together with two colleagues, he developed a simplified systemic grammar where elements were connected through a single function hence avoiding (functional) conflation. Also the ordering of elements was specified relative to the head rather than relative to each other.

More recently, O'Donnell in UAM Corpus Tool embedded a systemic chart parser (O'Donnell 2005) with a reduced systemic formalism. He classifies his parser as a left to right and bottom up with a custom lexicon where verbs are attributed features similar to Hallidayan process types and nouns a unique semantic category like thing-noun, event-noun, location-noun etc.

Because of previously reported complexity problems (O'Donnell 1993) with systemic grammars, the grammatical formalism is reduced to a singular functional layer of Mood-based syntactic structure (Subject, Predicate, Object etc.) ignoring the Transitivity (Actor/Goal, Sensor/Phenomenon etc.) and Textual (Theme/Rheme) analyses. O'Donnell deals away with the conflation except for the verbal group system network. He also employs a slot based ordering where elements do not relate to each other but rather to the group head only simplifying the number of rules and calculation complexity.

In his paper (O'Donnell 2005) does not provide a parser evaluation so its accuracy is still unknown today. The lexicon that was created is claimed to deal with word semantic classes but it is strongly syntactically based assigning a single sense to nouns and verbs ignoring the peculiar aspect of language polysemy. Moreover it is not very clear the framework within which the semantic classes have been generated.

1.7.4 O'Donoghue

O'Donoghue proposes a corpus based approach to parsing using *Vertical Strips* (O'Donoghue 1991). They are defined as a vertical path of nodes in a parse tree

starting from the root down to the lexical items but not including those. He extracted the set of vertical strips from a corpus called Prototype Grammar Corpus together with their frequencies and probability of occurrence. This approach differ from the traditional one with respect to the kind of generalization it is concerned and specifically, the traditional approached are oriented towards horizontal order while the vertical strip approach is concerned with vertical order in the parse tree.

To solve the order problem O'Donoghue uses a set of probabilistic collocation rules extracted from the same corpus indicating which strips can follow a particular strip. He also created a lexical resource indicating for each word which elements can expanded it.

The parsing procedure is a simple lookup of words in the lexical resource selecting all possible elements it can expound and then selecting possible strips starting with the elements expounded by the word. Advancing from left to right for each sentence word more strips compatible with the previously selected ones are selected within the collocation network constrains. The parser finds all possible combinations of strips composing parse trees representing possible output parses.

The corpus from which the vertical strips were extracted is 100,000 sentences large and was generated with Fawcett's natural language generation system and was tested on the same corpus leaving unclear how would the parser behave on a real corpus. In 98% of cases the parser returns a set of trees (between 0 and 56) that included the correct one with an average of 6.6 trees per parse.

Actually, using a larger corpus could potentially lead to a combinatorial explosion in the step that looks for vertical strips. It would decrease the accuracy of the parse because of the higher number of possible trees per parse.

1.7.5   Honnibal

Honnibal (2004; 2007) describes how Penn Treebank can be converted into a SFG Treebank. Before assigning to parse tree nodes synthetic features such as mood, tense, voice and negation he first transforms the parse trees into a form that facilitates the feature extraction.

The scope of SFG corpus was limited to a few Mood and Textual systems leaving aside Transitivity because of its inherently lexico-semantic nature. He briefly describes how he structurally deals with verb groups, complexes and ellipses as functional structures are much flatter than those exhibited in the original Treebank. Then he describes how are identified metafunctional features of unit class, mood function, clause status, mood type, polarity, tense, voice and textual functions.

The drawback of his approach is that the Python script performing the transformation does not derive any grammar but rather implements directly these transformations as functions falling into the same class of problems like Winograd's SHRDLU. By doing so the program is non-scalable for example in accommodation of larger grammars and knowledge bodies and unmaintainable on the long term as it becomes increasingly difficult to make changes.

1.8   Current approach to parsing

The main problem in using SFGs for parsing is that they are very large and complex. Some parsing approaches use a syntactic backbone which is then flashed out with SFG description. Other ones use a reduced set or a single layer of SFG representation the third ones use an annotated corpus as the source of a probabilistic grammar. Regardless of the approach each limits the SFG in a one way or another balancing the depth of description with language coverage: that is either *deep description but a restricted language* or *shallow description but broad language coverage.*

Current approach is aligned with works of Honnibal, Kasper and O'Donnell with respect to using a backbone structure and enriching it with syntactic and semantic features. It relies on parse structures produces in other grammars and then translated to systemic functional constituency structure. The contributions on theoretical compatibility and inter-grammatical transformations are briefed in the Section 1.6 coming up next.

Current method employs rules for graph traversal in order to build a parallel backbone constituency tree and rules for graph matching to enrich it with systemic features. This aims at keeping the language coverage broad at the expense of higher systemic delicacy. I cover basic Mood systems which is way less than what is available in Nigel grammar and leave it for the future work to extend the grammatical delicacy.

Nonetheless I attempt to cover some the lexico-semantic features as well. Parsing Transitivity system, a task similar to Semantic Role Labelling, requires large lexicogrammatical resource describing verb meanings in terms of their process type and participant roles. The semantically-oriented decomposition of clauses offered by SFL is still sufficiently closely tied to observable grammatical distinctions as to offer a powerful bridge to automatic analysis. Such descriptions are analogous to frame representations (Fillmore 1985) as found in FrameNet (Baker et al. 1998) or VerbNet (Kipper et al. 2008) applied in Semantic Role Labelling Task (Carreras & Màrquez 2005).

O'Donnell approaches this task by providing possible process types directly to the verb by employing self constructed lexicon where each word has syntactic and semantic features. Only recently a resource comparable to FrameNet and VerbNet has been produced in the SFL framework called Process Type Database (PTDB) (Neale 2002). PTDB which provides for each verb a process configuration (similar to a semantic frames) in terms of process type and participant roles. Current work represents the first attempt of using the PTDB to produce semantic (or Transitivity) analysis which combined with pattern patching method has an advantage over O'Donnell's parser. It enables to simultaneously assign, if matched, the systemic features to all clause constituents or not at all.

Another major advantage, as compared to Honnibal's approach is that the grammar and the program are carefully disconnected so that the code is maintainable and scalable with the respect to size of the grammar. This makes it possible to choose rather pragmatically which graph patterns to consider for parsing depending on the scope of task at hand.

1.9   The parsing process overview

Te parser follows a pipeline architecture depicted in Figure 1.4 where staring from an input text gradually a rich systemic functional constituency structure is built. This sections provides an overview to the building process.

In the figure there are three types of boxes. The rounded rectangles represent the parsing steps. They linearly flow from one to the next one via green trapezoids boxes, on the left side, which represent intermediary data. On the right side are positioned double edged orange trapezoids representing some fixed resources used as additional input for some steps. For example *constituency graph creation* step takes a normalised dependency graph for input and produces a constituency graph as output.

The entire process starts with some input English text and ends with production of Rich Constituency Graph. The Input text is first parsed with a dependency parser. For the current work Stanford dependency parser was chosen for its dependency relations, parse accuracy and the continuous efforts put into its development (motivated in Section 1.6).

The dependency graphs often contain errors some of which are predictable, easy to identify and correct. Also some linguistic phenomena are treated in a slightly different manner than proposed in the current thesis. Therefore the dependency graph
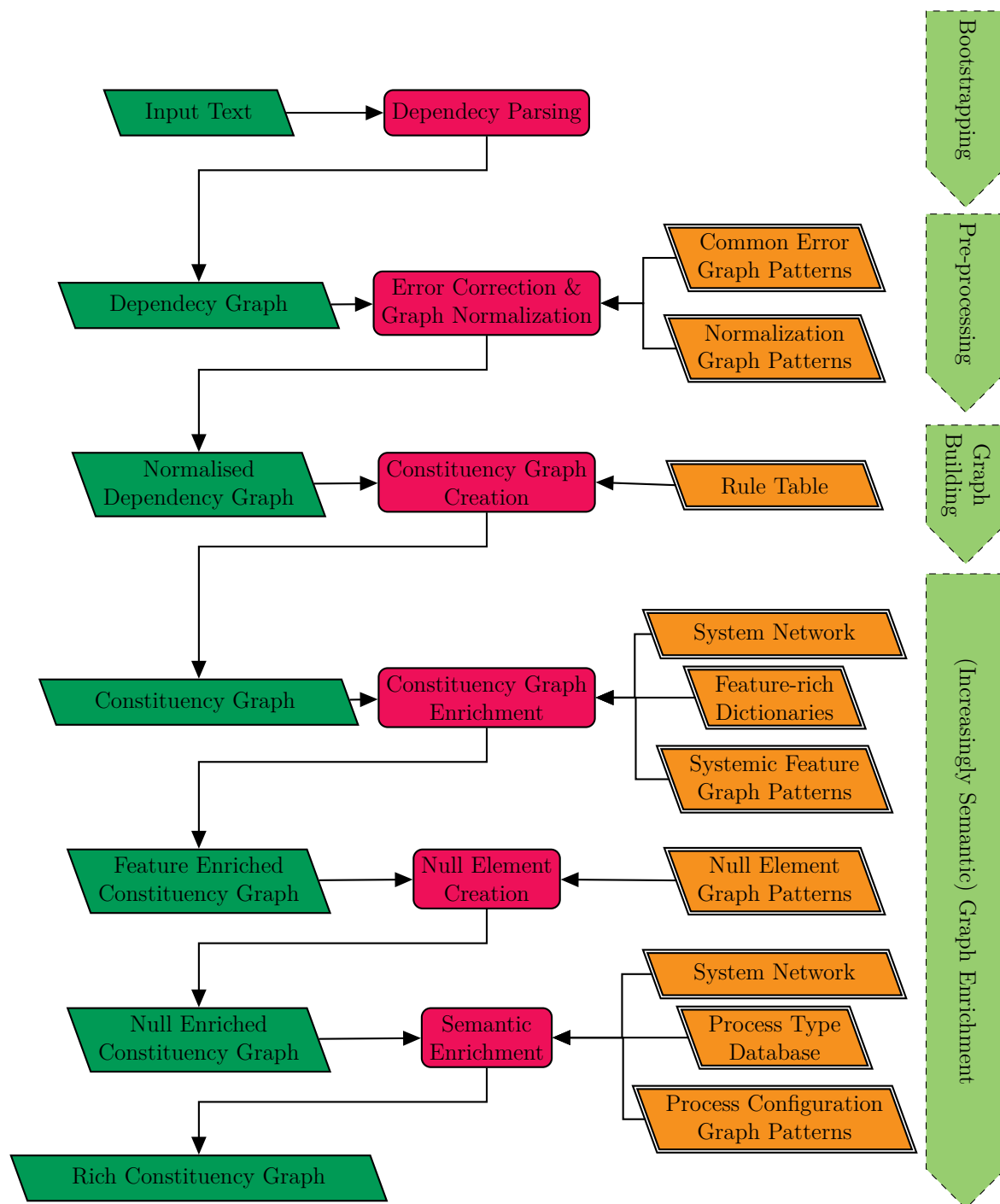
Fig. 1.4 The parsing process pipeline

produced by Stanford parser is *corrected and normalised* using pattern matching against a collections of known errors and one of normalization rules.

Once normalised the dependency graph is ready to guide the *building process* of the systemic functional constituency graph. It represents, in a way, transformation of the dependency graph, and serves a syntactic backbone on which the subsequent enrichment phases are performed.

Next follow two phases where the syntactic backbone is *enriched* with features some of which bear a *syntactic* whereas other a *semantic* nature. In between these enrichment phases there is a construction process which produces structural changes to the backbone adding some *empty constituents* that play a role in semantic enrichment. The enrichment phases use additional resources such as *system networks*, *feature rich lexicons*, *graph patterns* and *semantic databases*. The *null element creation* process also needs a collection of graph patterns for identifying where and what kind of null elements occur. The final result of the process is a *Rich Constituency Graph* of the original text comprising a plenitude of systemic feature selections associated with constituting units of structure.

1.10   Research questions and contributions

This thesis addresses the following questions:

- What is a computationally feasible method to parse with systemic functional grammars with a syntactic backbone?

- To what degree are Stanford Dependencies suitable as a syntactic backbone for Systemic Functional Grammar parsing?

- How can Process Type Database be used as a resource for SFG Transitivity parsing?

- How can Government and Binding Theory be used for detecting external predicate arguments in the context of SFG Transitivity parsing with PTDB?

Also it brings the following contributions:

- The analysis of theoretical and practical compatibility between the syntactic structures of Stanford Dependency and Systemic Functional Grammars along with an implemented method to transform from one structure to another.

- A fast engine for graph pattern matching which can also update and insert new nodes.

- A flexible and expressive method to represent systemic features as graph patterns together with two strategies for choice propagation in the systemic networks.

- A set of pattern graphs covering Mood, Transitivity and other smaller system networks.

- A method to transform PTDB into a set of Transitivity graph patterns.

- Derived principles and generalizations from the Government and Binding Theory (GBT) and represented them as graph patterns used to identify the covert elements of the clause that are explicitly mentioned outside the clause borders. These generalizations serve for semantic parsing where is very helpful to identify the external arguments of verbs.

- Development of a test corpus and evaluation of the parser.

1.11   Thesis organisation

The remaining of this thesis is organised as follows.

Chapter 2 explains in parallel Cardiff and Sydney theories of grammar followed by a discussion of structure units of each grammar. When juxtaposed, weaknesses and strengths of each school emerge in contrast to each other on aspects like *structure, dependency relations, unit classes, systemic networks, rank scale and unit complexing.* I use elements of both grammars therefore I explain my stance on each of the above issues and argument the choices. In similar manner I discuss the syntactic and semantic units of each grammar even if the systemic functional linguistics does not make such distinction it is useful in establishing links to mainstream methods for language processing. Basically, this chapter presents the mixed grammar and its theoretical underpinning through the comparative discussion between two schools in SFL.

Chapters 3 and 4 introduce *Dependency Grammar* and *Governance and Binding Theory (GBT)*. Both frameworks are used as departing points to build the SFG structure. The cross-theoretical correspondences together with specific inter-grammatical links are developed in the same chapters.

Chapter 5 formally defines the structures used in this thesis and the operations on them. Important to mention structures are *feature rich graphs, ordered conjunction sets, feature structures* and *system networks*; whereas important operations are the varieties of *graph matching* and *pattern graph matching*.

Chapters 6 and 7 explain how the parsing process evolves starting from the dependency graph towards a constituency graph and then towards increasingly semantic constituency graph through its feature features. This suite of algorithms and the pipeline has a Python implementation called Parsimonious Vole[1]. A limited empirical evaluation of the parser is provided in Chapter 8. It describes the evaluation methodology, the gold standard used and highlights strengths and weaknesses of the current implementation.

The last part of the thesis sets future directions explore and concludes on the current work (Chapter 9).

---

[1]Parsimonious Vole: https://bitbucket.org/lps/parsimonious-vole

Chapter 2

The systemic functional theory of grammar

There are two variants of Systemic Functional Grammars: the *Sydney Grammar* started in 1961 by Halliday (2002) and the *Cardiff Grammar* proposed by Fawcett (2008) which is a simplification and an extension of Sydney Grammar.

To understand the underlying motives and how exactly they are different we shall start looking at the theories of grammar before we look at the grammars proposed in Sydney and Cardiff SFL schools.

What is the difference between grammar and the theory of grammar, you may ask. If the grammar describes language in terms of categories, functions, relations, structures, meaning choices etc. then the theory of grammar defines what are the concepts that should be used to describer a grammar i.e. categories, functions, relations are and how they related to one another. Having a solid theory of grammar contributes to explaining what language is and how it works. It also frames how language is analysed by either human or machines.

This chapter discusses comparatively Halliday's (Halliday 2002) and Fawcett's (Fawcett 2000) theoretical foundations of SFL.

2.1   A word on wording

Before going into deeper discussion I would like to first make a few terminological clarifications on the terms: grammar, grammatics, syntax, semantics and lexicogrammar. I start with a few definitions adopted in the "mainstream" generative linguistics and then present how the same terms are discussed in systemic functional linguistics.

A. Radford, a generative linguist, in the "Minimalist Introduction to Syntax" (1997), starts with a description of grammar as a field of study, which, in his words is traditionally subdivided into two inter-related areas of study: syntax and morphology.

Definition 2.1.1 (Morphology (Radford))**.** Morphology is the study of how words are formed out of smaller units (traditionally called morphemes) (Radford 1997: p.1).

Definition 2.1.2 (Syntax (Radford))**.** Syntax is the study of how words can be combined together to form phrases and sentences. (Radford 1997: p.1)

Halliday, in the context of rank scale discussion (Halliday 2002: p. 51), refers to the traditional meaning of syntax as the *grammar above the word* and to morphology as *grammar below the word*. Such distinction, he stresses, has no theoretical status. His precursor, Firth, puts it in following terms: "[...] the distinction between morphology and syntax is no longer useful or convenient in descriptive linguistics." (Firth 1957: p.14)

Radford adds that, traditionally, grammar is not only concerned with the principles governing formation of words, phrases and sentences but also with principles governing their interpretation. Therefore *structural aspects of meaning* are said to be also a part of grammar.

Definition 2.1.3 (Grammar (Radford))**.** [Grammar is] the study of the principles which govern the formation and interpretation of words, phrases and sentences. (Radford 1997: p.1)

Interestingly enough, the Definition 2.1.3 makes not mention at all to the lexicon. This is because the formal grammars focus primarily on unit classes and how they are accommodated in various structures and so in formal linguistics the lexicon is disconnected from the grammar. The systemic grammar, on the other hand, along with formal descriptions of grammatical categories and structures, include lexicon as part of grammar to form a *lexicogrammar*.

Another important aspect to notice is that the grammar is defined as a field of study rather than a set of rules. Halliday, since his early papers, became conscious of the confusion made in the literature between a study of a phenomenon with the phenomenon itself. By analogy to language as phenomenon and linguistics as the study of the phenomenon, Halliday adopts the same wording for **grammar** as phenomenon and *grammatics* as the study of grammar; the same distinction holds for *syntax* and *syntactics*.

Definition 2.1.4 (Grammatics (Halliday))**.** Grammatics is a theory for explaining grammar (Halliday 2002: p.369)

E. Moravcsik, another generative linguist stresses the same distinction , in her "An introduction to syntax" (Moravcsik 2006), and presents two ways in which the word "syntax" is used in literature: (a) in reference to a particular aspect of grammatical structure and (b) in reference to a sub-field of descriptive linguistics that describes this aspect of grammar.

In her words "syntax describes the selection and order of words that make well-formed sentences and it does so in as general a manner as possible so as to bring out similarities among different sentences of the same language and different languages and render them explainable. [...] syntax rules also need to account for the relationship between string of word meanings and the entire sentence meaning, on one hand, and relationship between strings of word forms and the entire sentential phonetic form, on the other hand." (Moravcsik 2006: p.25)

In her definition of grammar she includes lexicon and semantics which is somewhat more explicit statement than Radford's "interpretation". She is getting, in Definition 2.1.5, somewhat closer to what grammar stands for in SFL - Definition 2.1.6.

Definition 2.1.5 (Grammar (Moravcsik))**.** ... maximally general analytic descriptions, provided by descriptive linguistics, [are] called grammars. A grammar has five components: phonology (or, depending on the medium, its correspondent e.g. morphology), lexicon, syntax and semantics(Moravcsik 2006: pp.24–25).

Definition 2.1.6 (Grammar (Halliday))**.** To Halliday, lexico-grammar or short grammar is a part of language and it means the wording system - the "lexical-grammatical stratum of natural language as traditionally understood, comprising its syntax, vocabulary together with any morphology the language may display [...]" (Halliday 2002: p.369).

The last point I want to mention is the approach to semantics. Formal grammars aim to account for the realisation variations, that is formation of words, phrases and sentences along with their arrangements and the mentions of semantics merely refer to the "formal aspect of meaning".

By contrast, a systemic grammar is a functional grammar, which means (among other things) that it is semantically motivated, i.e. "natural". So the fundamental distinctions between formal and functional grammars is the semantic basis for explanations of structure.

Also, in SFL, the meaning is being approached from a semiotic perspective, punting the linguistic semantics in perspective with the linguistic expression and the real world situation.

In this respect, Lemke (1993) offers a well formulated theoretical foundation that "human communities are eco-social systems that persist in time through ongoing exchange with their environment; and the same holds true for any of their sub-subsystems [...]" including language. The social practices constituting such systems are both material and semiotic, with a constant dynamic interplay between the two. (Halliday 2002: p.387)

The term *semiotic* means to Halliday oriented towards meaning rather than sign. In other words, the interaction is between *the practice of doing and the practice of meaning.*

As the two sets of practices are strongly coupled, Lemke points out that there is a high degree of redundancy between the material-semiotic interplay. And it perfectly resonates with Firth's idea of *mutual expectancy* between the text and the situation.
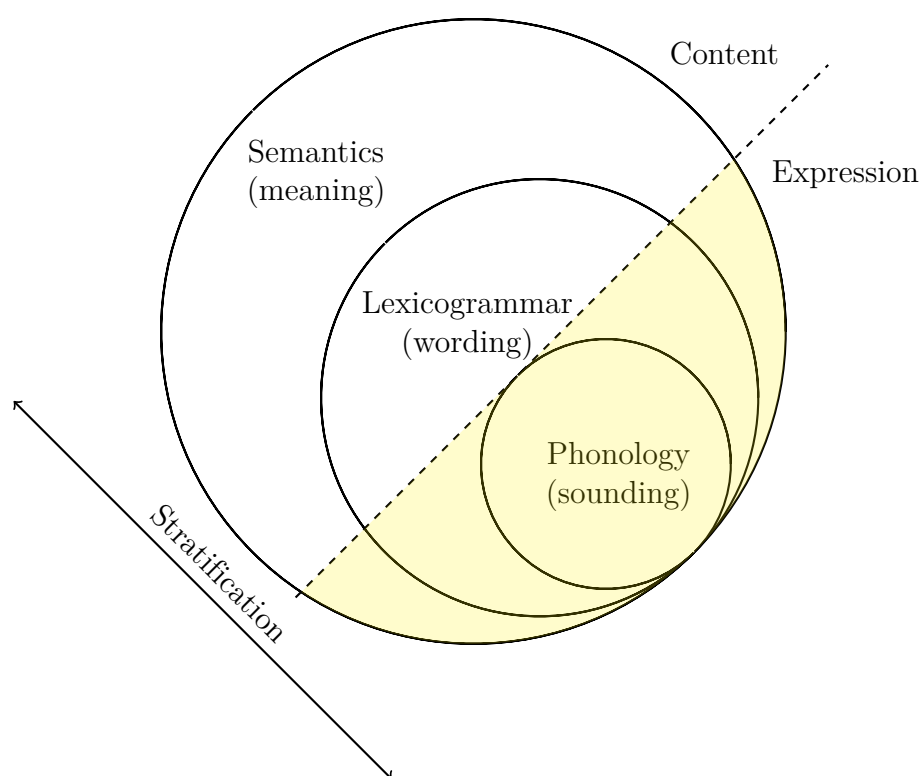
Fig. 2.1 The levels of abstraction along the realisation axis

Having that said, the best way to relate the formal and the systemic functional grammars is by placing the two of them along the stratification axis.

The SFL model defines language as a resource organised into three strata: phonology (sounding), lexicogrammar (wording) and semantics (meaning). Each is defined according to its level of abstraction on the realisation axis. The realisation axis is divided into two planes: the expression and the content planes. The first strata (i.e. phonology) belongs to the *expression plane* and the last two (lexicogrammar and semantics) belong to the *content plane*. But the division is not so clear because some parts of semantics and lexicogrammar transcend from content into expression plane. In this context, the formal grammar could be localised entirely within the expression plane, including the phonology/morphology, syntax, lexicon and semantics but stripped of any explanations in terms of the meaning potential available in the content plane.

2.2   Sydney theory of grammar

Halliday (2002) proposes four fundamental categories of grammar: *unit*, *structure*, *class* and *system*. Each of these categories is logically derivable from and related to other ones in a way that they mutually define each other. These categories relate to each other on three scales of abstraction: *rank*, *exponence*, *delicacy*. Halliday also uses three scale types: *hierarchy*, *taxonomy* and *cline*.

Definition 2.2.1 (Hierarchy)**.** Hierarchy [is] a system of terms related along a single dimension which involves a some sort of logical precedence. (Halliday 2002: p.42).

Definition 2.2.2 (Taxonomy)**.** Taxonomy [is] a type of hierarchy with two characteristics:

1. the relation between a terms and the immediately following and preceding one is constant

2. the degree is significant and is defined by the place in the order of a term relative to following and preceding terms. (Halliday 2002: p.42)

Definition 2.2.3 (Cline)**.** Cline [is] a hierarchy that instead of being made of a number of discrete terms, is a continuum carrying potentially infinite gradations. (Halliday 2002: p.42).

Next I define and introduce each category of grammar and the related concepts that constitute the theoretical foundation for the Sydney Theory of grammar.

2.2.1 Unit

Language is patterned activity of meaningful organization. The patterned organization of substance (*graphic* or *phonic*) along a linear progression is called *syntagmatic order* (or simply *order*).

Definition 2.2.4 (Unit)**.** The unit is a grammatical category that accounts for the stretches that carry grammatical patterns. (Halliday 2002: p.42). The units carry a fundamental *class* distinction and should be fully identifiable in description. (Halliday 2002: p.45).

Generalization 2.2.1 (Constituency principles)**.** The five principles of constituency in lexicogrammar are:

1. There is a scale or rank in the grammar of every language. That of English (typical of many) can be represented as: clause, group/phrase, word, morpheme.

2. Each unit consists of *one or more* units of rank next below.

3. Units of every rank may form complexes.

4. There is potential for rank shift, whereby a unit of one rank my be downranked to function in a structure of a unit of its own rank or of a rank below.

5. Under certain circumstances it is possible for one unit to be enclsoed within another, not as a constituent but simply in such a way as to split the other oe into two discrete parts. (Halliday & Matthiessen 2013: pp.9–10)

The relation between units is that of consistency for witch we say that a unit *consists of* other units. The scale on which the units are ranged is the *rank scale*. The rank scale is a levelling system of units supporting unit composition regulating how units are organised at different granularity levels from clause, to groups/phrases to words and the units of a higher rank scale consist of units of the rank next below. The Table 2.1 presents a schematic representation of the rank scale and its derived complexes.

Generalization 2.2.2 (Rank scale constraints)**.** The rank relations are constrained as follows:

1. downward *rankshift* is allowed i.e. the transfer of a given unit to a lower rank.

2. upward rankshift is not allowed.

Rank scale ↓	Complexing
	Clause complex
Clause	
	Group(/phrase) complex
Group(/phrase)	
	Word complex
Word	
	(Morpheme complex)
(Morpheme)	

Table 2.1 Rank scale of the (English) lexicogrammatical constituency

3. only whole units can enter into higher units.(Halliday 2002: p.44).

The Generalization 2.2.2 taken as a whole means that a unit can include, in what it consists of, a unit of rank higher than or equal to itself but not a unit of rank more than one degree lower than itself; and not in any case a part of any unit. (Halliday 2002: p.42).

2.2.2 Structure

Definition 2.2.5 (Structure)**.** The structure (of a given unit) is the arrangement of *elements* that take places distinguished by order relationship (Halliday 2002: p.46).

Definition 2.2.6 (Element)**.** The element is defined by the place stated as absolute or relative position in sequence and with the reference to the unit next below (Halliday 2002: p.47).
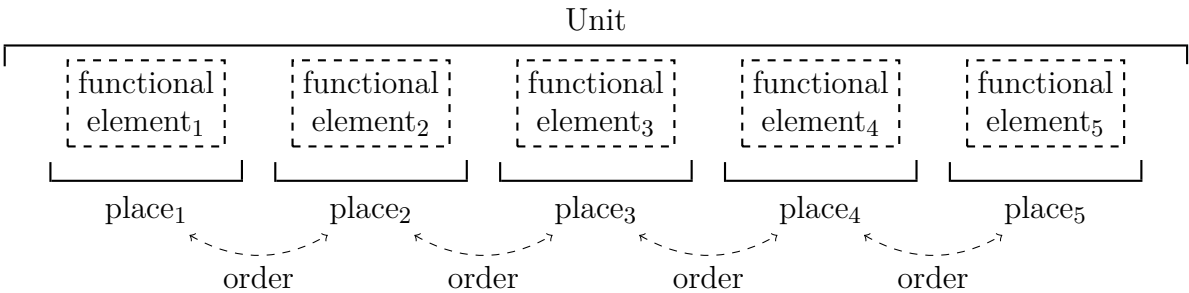
Fig. 2.2 The graphic representation of (unit) structure

We say that an unit is composed of elements located in places and that its internal structure is accounted via elements in terms of functions and places taken by the lower (constituting) units or lexical items. The graphic representation of the unit structure

is depicted in Figure 2.2. The unit structure is referred in linguistic terminology as *constituency* (whose principles are enumerated in Generalization 2.2.1). In the unit structure, the elements resemble an array of empty slots that are *filled* by other units or lexical items.

2.2.3 Class

Definition 2.2.7 (Class)**.** The class is that grouping of members of a given unit which is defined by operation in the structure of the unit next above (Halliday 2002: p.49).

Halliday defines class (Definition 2.2.7) as likeness of the same rank "phenomena" to occur together in the structure. He adopts a top-down approach stating that the class of a unit is determined by the *function* (Definition 2.2.9) it plays in the unit above and not by its internal structure of elements. In SG the structure of each class is well accounted in terms of syntactic variation recognizing six unit classes: *clause*, *prepositional phrase* and *nominal, verbal, adverbial* and *conjunction* groups. Sydney grammar is briefly summarised in the Appendix 9.6.

2.2.4 System

Structure is a syntagmatic ordering in language capturing regularities and patterns which can be paraphrased as *what goes together with what*. However, language is best represented as a set of system networks (Definition 2.2.8) which is a paradigmatic ordering in language describing *what could go instead of what* (Halliday & Matthiessen 2013: p. 22).

This is an essential assumption of systemicists is that the language is best represented in the form of system networks and not as an inventory of structures. The structure of course is a part of language description but it is only a syntagmatic manifestation of the systemic choices (Halliday & Matthiessen 2013: p.23).

Definition 2.2.8 (System)**.** A system is a set of mutually exclusive set of terms referring to meaning potentials in language and are mutually defining. It always means a *closed system* and has the following characteristics:

1. the number of terms is finite,

2. each term is exclusive of all others,

3. if a new term is added to the system it changes the meaning of all other terms. (Halliday 2002: p.41)

A class is a grouping of items identified by operation in the structure. It is not a list of formal items but an abstraction from them. By increase in *delicacy* the class is broken into secondary classes which stand in the relation of exponent to an element of primary structure of the unit next above. This breakdown gives a system of classes that constitute choices implied by the nature of the class. (Halliday 2002: p.41)

2.2.5 Functions and metafunction

Definition 2.2.9 (Function)**.** The functional categories or functions provide an interpretation of grammatical structure in terms of the overall meaning potential of the language. (Halliday & Matthiessen 2013: p.76).

Most constituents of the clause structure, however, have more than one function which is called a *conflation of elements.* For example in the sentence "Bill gave Dolly a rose", "Bill" is the Actor doing the act of giving but also the Subject of the sentence. So we say that Actor and Subject functions are conflated in the constituent "Bill". This is exactly the point where the concept of *metafunction* or *strand of meaning* comes into the picture. The Subject function is said to belong to the *interpersonal metafunction* while Actor function belongs in the *experiential metafunction.*

Halliday identifies three fundamental dimensions of structure in the clause each with distinct meaning: *experiential, interpersonal* and *textual.* He refers to them as *metafunctions* and they account of how language meaning has evolved. Table 2.2 presents metafunctions and their reflexes in the grammar as proposed in (Halliday & Matthiessen 2013: p.85).

Metafunction	Definition(kind of meaning)	Corresponding status in clause	Favored type of structure
experiential	construing a model of experience	clause as representation	segmental (based on constituency)
interpresonal	enacting social relationship	clause as exchange	prosodic
textual	creating relevance to context	clause as message	culminative
logical	constructing logical relations	-	iterative

Table 2.2 Metafunctions and their reflexes in the grammar

Generalization 2.2.3 (Exhaustiveness principle)**.** Everything in the wording has some function at every rank but not everything has a function in every dimension of structure. (Halliday 2002; Halliday & Matthiessen 2013)

With respect to structure and metafunctions, Halliday formulates the general principle of *exhaustiveness* (Generalization 2.2.3) saying that clause constituents have at least one and may have multiple functions in different strands of meaning, however it does not mean that it must have a function in each of them.

This principle implicitly relates to the economic property of language meaning that it naturally evolves towards the shortest and most effective way of expressing a meaning. There is nothing meaningless thus every piece of language must be explained and accounted for in the lexicogrammar.

2.2.6 Lexis and lexicogrammar

In SFL the terms *word* and *lexical item* are not really synonymous. They are strongly related but they refer to different things. The term "word" is reserved (in early Halliday) for the grammatical unit of the lowest rank whose *exponents* are lexical items.

Definition 2.2.10 (Lexical Item)**.** In English, a lexical item may be a *morpheme*, *word* (in traditional sense) or *group (of words)* and it is assigned to no rank. (Halliday 2002: p.60)

Examples of lexical items are all of the following ones: " 's " (the possessive morpheme), "house, walk, on" (words in traditional sense) and "in front of, according to, ask around, add up to, break down" (multi word prepositions and phrasal verbs)

If most linguists treat the grammar and lexis as discrete phenomena, Halliday brings them together as opposite poles of the same cline. We say that they are paradigmatically related through delicacy relation. He refers to this merge as *lexicogrammar* and he expressed his dream that one day linguists will be able to turn whole linguistic form into (lexico)grammar showing that lexis is the most delicate grammar.

Hasan (2014), explores the reality of Halliday's dream in terms of project feasibility and exploring the implications of what would it mean to turn the "whole linguistic form into grammar". This then implies two completely new assumptions: that lexis is not form and that its relation to semantics is unique (challenging the problems of polysemy). It would be the function of the lexicogrammar to map the multiple *meta-functional strata* into a unified structure.

2.3   Cardiff Theory of grammar

This section present the theory of grammar as conceived by Robin Fawcett at University of Cardiff. The biggest difference to Hallidayan theory is renouncing the concept of rank scale which has an impact on the whole theory. As a consequence, to accommodate the lack of rank-scale, Fawcett adapts the definitions of the fundamental concepts and slightly changes the choice of words.

In 2000, Robin Fawcett presents a theory of grammar in contrast to some aspects of Michael Halliday's grammar 2002. One of the main differences is the rejection of the rank scale concept. Another is the bottom-up approach to unit definition as opposed to top-down one advocated by Halliday. These two and few other discrepancies have quite an important implication on the overall theory of grammar and of course the grammar itself.

Fawcett (2000) proposes three fundamental categories in the theory of grammar: *class of unit, element of structure* and *item.* Constituency is a relation accounting for prominent compositional dimension of language. However a unit does not function directly as a constituent of another unit but via a specialised relation. Fawcett breaks down constituency into three relations: *componence*, *filling* and *exponence.* Informally is said that a unit is composed of elements which are either filled by another unit or expounded by an item. He also proposes three secondary relations of *coordination*, *embedding* and *reiteration* to account for the full range of syntactic phenomena.

2.3.1   Class of units

Definition 2.3.1 (Class of Unit)**.** The class of unit [...] expresses a specific array of meanings that are associated with each one of the major classes of entities in semantics [...and] are to be identified by the elements of their internal structure (Fawcett 2000: p.195).

Class of unit is determined based on its internal structure i.e. by its elements of structure (and not by the function it plays in the parent unit).

Fawcett takes a semantic stance in classifying units which in line with Saussurean approach to language. He proposes that in English there are four major semantic classes of entities: situations, things, qualities (of situations and things) and quantities (typically of things but also of situations and qualities) corresponding to major syntactic units of *clause*, *nominal group*, *prepositional group*, *quality group* and *quantity group* (Fawcett 2000: p. 193–194) along with a set of minor classes such as *genitive cluster* and *proper name cluster.*

His classification is based on the idea that the syntactic and semantic units are mutually determined and supported by grammatical patterns. However those patterns are beyond the syntactic variations of the grammar and blend into lexical semantics.

2.3.2   Element of Structure

Definition 2.3.2 (Element of Structure)**.** The elements of structure are immediate components of classes of units and are defined in terms of their *function* in expressing meaning and not in terms of their absolute or relative position in the unit. (Fawcett 2000: pp.213–214).

Generalization 2.3.1. Definition 2.3.2 leads to the following two principles:

1. Every element in a given class of unit serves a function in that unit different from the function of the sibling elements.

2. Every element in every class of unit will be different from every element in every other class of unit. (Fawcett 2000: p.214)

The elements (of structure) are functional slots which define the internal structure of an unit but still they are *located* in *places.* One more category that intervenes between element and unit is the concept of *place* which become essential for the generative versions of grammar.

There are two ways to approach place definition. The first, is to treat places a positions of elements relative to each other (usually previous). This leads to the need of an *anchor* or a *pivotal element* which may not always be present/realised.

The second, is to treat places as a linear sequence of locations at which elements may be located, identified by numbers "place 1", "place 2" etc. This place assignment approach is absolute within the unit structure and makes elements independent of each other. This approach has been used in COMMUNAL and Penman projects.

2.3.3   Item

Definition 2.3.3 (Item)**.** The item is a lexical manifestation of meaning outside syntax corresponding to both words (in the traditional sense), morphemes and either intonation or punctuation (depending whether the text is spoken or written). (Fawcett 2000: pp.226–232).

Items correspond to the leaves of syntactic trees and constitute the raw *phonetic* or *graphic* manifestation of language. The collection of items of a language is generally referred as *lexis.*

Since items and units are of different nature, the relationship between an element and a (lexical) item must be different from that to a unit. We say that items *expound* elements and not that they *fill* elements as units do.

In traditional grammar *word classes* or *parts of speech* are a commonly accepted concept. However in SFL, it plays rather an orientation or an approximation role, precisely because the word classes do not properly correspond to the elements they expound. So terms as *noun* or *adjective* are useful to denote a class of words that expound a certain element of the structure, but such word class to element correspondence shall by no means treated as definite rule.

2.3.4   Componence

Definition 2.3.4 (Componence)**.** Componence is the part-whole relationship between a unit and the elements it is composed of. (Fawcett 2000: p.244).

Note that componence is not a relationship between a unit and its places, the latter, as discussed in Section 2.3.2, simply locationally relate elements of a unit to each other.

Componence intuitively implies a part-whole constituency relationship between the unit and its elements. But this is not the only view. Another perspective is the concept of *dependency* or strictly speaking *sister or sibling dependency* (because the traditional concept of dependency is parent-daughter relation). However the sister dependency is not necessary in the grammar model and is a by-product or second order concept that can be deduced from the constituency structure.

The (supposed) dependency relation between a modifier and the head, in the framework of SFG is, not a direct one that form-centered linguists consider to be. They simply assume that what modifier modifies is the head. Here however the general function of the modifiers is to contribute to the meaning of the whole unit which is anchored by the head. For example, in the nominal group, the modifier contributes to the description of the referent stated by the head. So the head realises one type of meaning that relates the referent while modifier realises another one. Both of them describe the referent via different kinds of meaning, therefore they are related indirectly to each other because the modifier does not modify the head but the referent denoted by the head. (Fawcett 2000: p.216)

Moreover the dependency relations are expressed between system networks and according to Fawcett this is the true place for dependencies in SFL.

2.3.5 Filling and the role of probabilities

Definition 2.3.5 (Filling)**.** Filling is the probabilistic relationship between a element and the unit lower in the tree that operates at that element. (Fawcett 2000: p.238, 251).

Fawcett renounces the concept of rank scale and alternatively proposed the concept of *filling probabilities.* The probabilistic predictions are made in terms of filling relationship between a unit and an element of structure in a higher unit in the tree rather than being a relationship between units of different ranks. This places focus from the fact that a unit is for example a group, to what group class it is.

In this line of though, some elements of a clause are frequently filled by groups, but some other element almost never being rather expounded by items. The frequency varies greatly and is an important factor for predicting or recognizing either the unit class or the element type in the filling relationship.

Filling may add a single unit to the element of structure or it can introduce multiple coordinated units. Filling also makes possible the embedding relation. Both, coordination and embedding relations makes it possible to deal without inter-clausal *hypotaxis* and *parataxis* relations described in Sydney Grammar.

Note also that filling and componence are two complementary relations that occur in the syntactic tree down to the level when the analysis moves out of abstract syntactic categories to more concrete category of items via the relationship of exponence.

2.3.6 A few more concepts

Definition 2.3.6 (Exponence)**.** Exponence is the relation by which an element of structure is realised by a (lexical) item (Fawcett 2000: p.254).

Definition 2.3.7 (Coordination)**.** Coordination is the relation between units that fill the same element of structure (Fawcett 2000: p.263).

Coordination is usually marked by an overt *Linker* such as *and*, *or*, *but*, etc. and sometimes it is enforced by another linker that introduces the first unit such as *both*.

Coordination is through by Fawcett as being not between syntactic units but between mental referents. It always introduces more than one unit which are syntactically and

semantically in similar (somehow) resulting in a *syntactic parallelism* which often leads to *ellipsis.*

Definition 2.3.8 (Reiteration)**.** Reiteration is the relation between successive occurrences of the same item expounding the same element of structure (Fawcett 2000: p.271).

Reiteration often is used to create the effect of emphasis such as for example "she's very very nice!". Like coordination, reiteration is a relation between entities that fill the same element of the unit structure which is problematic in my opinion and I further discuss it in Section 2.4.6.

Definition 2.3.9 (Embedding)**.** Embedding is the relation that occurs when a unit fills an element of the same class of units, i.e. when a unit of the same class occurs (immediately) above it in the tree structure (Fawcett 2000: p.264).

Fawcett opens embedding as a general principle as opposed to exceptional/controlled embedding indicated by Halliday. I will further discuss it in the context of rank-scale concept in Section 2.4.1

Definition 2.3.10 (Conflation)**.** Conflation is the relationship between two elements that are filled by the same unit having the meaning of "immediately after and fused with" and function as one element (Fawcett 2000: pp.249–250).

Conflation is useful in expressing multi-faceted nature of language when for example syntactic and semantic elements/functions are realised by the same unit for example the Subject and the Agent or Complement and Affected. Also conflation relations frequently occur between syntactic elements as well such as for example the Main Verb and Operator or Operator and Auxiliary Verb.

Definition 2.3.11 (Taxis)**.** *Taxis* represents the degree of interdependency between units systematically arranged in a linear sequence where *parataxis* means equal and *hypotaxis* means unequal status of units forming a *nexus* together.

Taxis play a central role in explaining the textual cohesion. Figure 2.3 depicts the Taxis and clause logico-semantic relations in a single system network. Here taxis relations are employed in explaining how clauses form a nexus with certain tactic relations to one another (Halliday & Matthiessen 2013: pp. 438 – 443). These concepts nevertheless are very useful at describing unit relations not only at the group and clause ranks but all the way down to smallest linguistic unit such as morphemes and
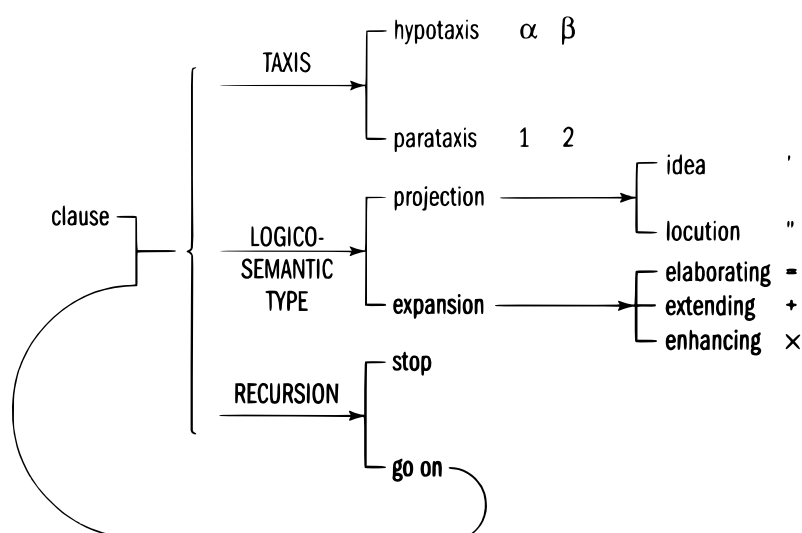
Fig. 2.3 Types of relations between clauses (Halliday & Matthiessen 2013: p.438)

phonemes. I will also employ the concept of taxis in the discussion of dependency relations in Section 3.5. Next I discuss the strengths and weaknesses of the two schools with a pragmatic goal of parsing in mind.

2.4   Critical discussion on both theories

The two sections above cover the definitions and fundamental concepts from each of the two systemic functional theories of grammar. Current work uses a mix of concepts from both theories and this section discusses in detail what and why is being adopted attempting a rather pragmatic reconciliation than a theoretical debate. Next I draw parallels and highlight correspondences between Sydney and Cardiff theories of grammar and where needed alter and present my position on the matter.

2.4.1   Relaxing the rank scale

The *rank scale* proposed by Halliday (2002) became overtime a highly controversial concept in linguistics. Whether it is a suitable for grammatical description or not still continues up to date. The historic development of this polemic is documented in (Fawcett 2000: p.309–338).

I consider rank scale a very useful dimension for unit classification and placement but the definition laid by the Sydney school is too rigid and thus I propose to relax it into a weaker version of it. The relaxation consists of dropping completely the *rank scale constraints* as enunciated in Generalization 2.2.2. An immediate consequence is

that the *embedding* relation can be broadly defined as a naturally occurring phenomena in language at all ranks and not only for clauses as initially proposed.

Halliday's theory allows the downwards rank shift, forbids upwards rank shift and restricts the composition relation to engage only with whole units. Thus the unit may be composed of units of equal rank or a rank higher and cannot be composed of units that are more than one rank lower or parts of other units. The consequence of above is that each element of the clause is filled by a group which has its elements expounded by words.

(3)   some very small wooden ones

The above rules often pose analysis difficulties and complications. For example in nominal groups what seems to be an element is not a single word but a group of words. Consider example 3 where Epithet *"very small"* is not a single word but a group (Halliday & Matthiessen 2013: pp. 390–396). This kind of phenomena introduced a *substructure* of modifiers and heads (see analysis in table 2.3) which complicates the general structure of the nominal group. Accordingly, the Epithet *"very small"* is composed of a head "small" and a modifier "very". This kind of intricate cases can be simplified through rank-shift constraints allowing the elements of a group to be filled by other groups or expounded by words.

some	very	small	wooden	ones
Deictic	*Epithet*		*Classifier*	*Thing*
Modifier				*Head*
	Sub-Modifier	*Sub-Head*		

Table 2.3 Sydney analysis of Example 3

An approach to describe units outside the rank-scale was suggested by Fawcett (2000) and Butler (1985). Because units are carriers of a grammatical pattern, they can be described in terms of their internal structure instead of their potential for operation in the unit above.

Fawcett proposes complete abandonment of rank scale replaces it with the filling probabilities to guide the unit composition simply mapping elements to a set of legal unit classes that may fill it. The above example, in CG the *"very small"* is analysed as a quality group that plays the role of Modifier (CG) in the nominal group as in table 2.4.

some	very	small	wooden	ones
Quantifying Determiner	*Modifier*		*Modifier*	*Head*
	Quality Group			
	Degree Tamperer	*Apex*		

Table 2.4 Cardiff analysis of Example 3

I maintain the idea of ranking the syntactic units because it is a pertinent classification with a clear correspondence to the types of meaning structures in the ideational space.

However I drop the constituency constraints and hence allowing the flexibility for elements to be filled by other units or in other words allow unit embedding. This approach removes the need of sub-structures in the unit elements reducing thus the structural complexity.

The rank system constrains had consequences on the phenomena of embedding defined by Halliday in Definition 2.4.1, which I consider way too restrictive.

Definition 2.4.1 (Embedding (strict)). Embedding is the mechanism whereby a clause or phrase comes to function as a constituent within the structure of a group, which is itself a constituent of a clause. (Halliday & Matthiessen 2013: p.242)

The weakening of constituency constrains makes embedding a normal (as defined in broad sense 2.3.9 by Fawcett) rather than an exceptional (as defined in a strict sense in 2.4.1 by Hallliday) phenomena. And I agree with Fawcett's definition because the human language allows construction of units that contain other units within them regardless of their class.

2.4.2 The unit classes

Fawcett drops the concept of rank system (discussed in Section 2.4.1) and through a bottom-up approach redefining the class as a "class of unit" as in 2.3.1.

He adopts Saussurean perspective on language which states that semantics and syntax are strongly intertwined with each other so major semantic classes of entities correspond to the major syntactic units. This lead Fawcett to take a semantic basis for classifying syntactic units into: clause, nominal group, prepositional group, quality group and quantity group (Fawcett 2000: p. 193–194) along with a set of minor classes such as genitive and proper name clusters.

The problem with this approach is that these classes are beyond the syntactic variations of the grammar and blend into lexical semantics which makes it difficult to apply to parsing, at least nowadays with current state of word classification.

In the current project I turn to Sydney classification of syntactic units that is close in line with traditional syntactic classification (Quirk et al. 1985). I adopt the clause as a unit plus the four group classes of Sidney grammar depicted in Figure 2.4.

(a) The group classes
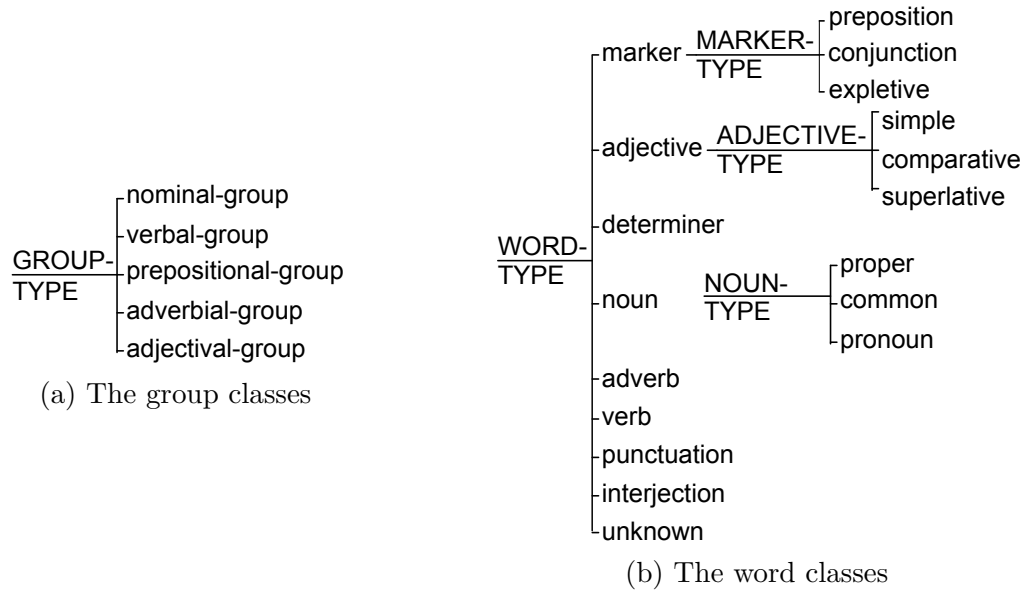
(b) The word classes

Fig. 2.4 The group and word classes

2.4.3   The structure

The *unit* and *structure* are two out of the four fundamental categories in the systemic theories of grammar. Sydney and Cardiff theories vary in their perspectives on *unit* and *structure* influencing how units are defined and identified.

For Halliday, the *structure* (Definition 2.2.5) characterises each unit as a carrier of a pattern of a particular order of *elements*. The order is not necessarily linear realisation sequence but a theoretical relation of relative or absolute placement. This perspective has been proved useful in generation exercises where unit placement evolves in the realisation process.

The Cardiff School take a bottom up approach and defines class in terms of its internal structure describing a relative or absolute order of elements. This sort of syntagmatic account is precisely what is deemed useful in parsing and is the one adopted in this thesis. It is well established algorithmically how to recognise classes and construct them bottom up. So in our case easier to let the unit class emerge

from recognition of constituent part-of-speech (word classes) and dependency relations between words or sequence of lower unit classes. In other words the unit class is defined by the unit structure and not by it's function in the parent unit, as Sydney school predicates, and this is precisely the reason why creation of constituency structure is computationally accessible.

2.4.4 Syntactic and semantic Heads

In SFG the heads may be semantic or syntactic. In most cases they coincide but there are exceptions when they differ or even miss. This is especially an important topic in the discussions of the nominal group structure on which Halliday & Matthiessen (2013) offer a thorough examination but Fawcett (2000) offers us a more generic perspective on this issue.

Consider the example of nominal group "a cup of tea" analysed in three different ways in the Table 2.5. The Sydney Grammar offers two analyses in which the semantic and the syntactic heads differ. In the *experiential* analysis the head is "tea" which functions as *Thing*, while in the *interpersonal* analysis the head is "cup" which functions as *Head*.

Cardiff Grammar does not make the Head/Thing distinction because the functional elements are already established based on semantic criteria. discussed in subsection 2.5.3. Nevertheless the logical analysis of SG resonates closely with the traditional "semantically blinded" grammars because it always provides a syntactic Head even if differs from the "pivotal element" of the group.

		a	**cup**	**of**	**tea**
Sidney Grammar	experiential		*Numerative*		*Thing*
	interpersonal	*Modifier*	*Head*	*Qualifier*	
Cardiff Grammar		*Quantifying Determiner*		*Selector*	*Head*

Table 2.5 Example of dispersed semantic and syntactic heads

Fawcett argues that none of the constituting elements of the unit is mandatory realised even the so called *"pivotal element"* which is the group defining element. The logical structure heads are always realised and correspond dependency relations established in the DG. Depending on the unit class logical structure heads are conflated with specific functions, for instance in nominal group the Head is usually conflated with the Thing, in quality group with the Apex, in quantity group with the Amount, in clause with the Main Verb and so on. But in language it is not unusual to have

nominal groups with the Thing missing or elliptic clauses with missing the Main verb so no rigid correspondence can be established between the Head, unit class and the corresponding pivotal element of the group. So because the unit class depends on its internal structure leading to a circular interdependency between the unit class and the unit structure. To solve this issue Fawcett argues for bototm-up approach where head-modifier relations are identified between lexical items and then between units (i.e. groups and clauses) serving as cues to identify elements of the higher unit and therefore it's class. Usually the class membership of head is raised to the unit class although sometimes the presence or absence of certain elements (during the reconstruction process) may alter the unit class to a different from the logical head.

(4)   The old shall pass first.

Consider the nominal group "The old" which is the subject in example 4. The head of the nominal group is the adjective "old" and not a noun as it would normally be expected. The noun modified by the adjective "old" is left covert and it shall be recoverable from the context. We can insert a generic noun "one" to form a canonical noun group:"the old one". In such cases when the head noun is missing, the logical head is conflated with other element in this case the Epithet. The group class is not raised from the word class to quality group but is identified by internal structure of the whole group and in this case the presence of determiner signals a nominal class. I point out through this example that the class of the head is not always is raised to establish the group class but the whole underlying structure determines the group class.

2.4.5   Systems and systemic networks

Fore example consider polarity system represented in figure 2.5. It contains two choices either positive or negative. And when one says it is positive one means not negative which is obvious and self evident how the two choices are mutually exclusive.
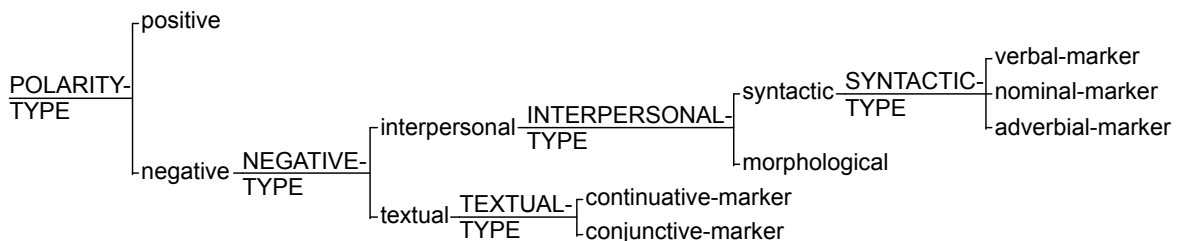
Fig. 2.5 System network of POLARITY

In language it may be often the case that a choice in a system may lead to re-entering the same system again to make another choice again forming this way recursive systems. Alternatively, we can say that a system allows multiple choices for the same unit. These perspectives are the sides of the same coin where the recursion perspective is useful for natural language generation while multiple choice perspective suits better the parsing task and next I explain why by continuing the discussion on polarity system.

The negative polarity in English clauses can be realised in several ways: via a noun group with intrinsic negative polarity feature like *"nobody"* (5), negation particle of the verb *"n't"* (6) or adverb with intrinsic negative polarity (7).

(5)   Nobody with any sense is going.

(6)   I don't have to mow my lawn.

(7)   Never expect her to come back.

Consider now, the cases of double negation from the example 8 where two kinds of negations are realised in the same clause: the negation by verb particle *"n't"* and the pronoun *"nobody"*.

(8)   Nobody with any sense isn't going.

(9)   I don't have nobody to mow my lawn.

The systems can be recursive and thus choices are not always mutually exclusive. Even though the system network clearly distinguishes one type of negation from another multiple negations can still occurring simultaneously. Note that this more delicate distinctions in kind of negation, still is a negation and for any of them it is impossible to co-occur with positive polarity. The issue here is not semantic about whether the clause is positive or negative but what kinds of grammatical choices can be identified within the clause. The problem of whether the double negation shall be interpreted as positive is not necessarily as relevant as the task of identifying the two instances of negation.

Halliday states that the speaker makes only one choice from a system. If this rule is interpreted as two choices from the same system at a time being impossible then it clearly does not cover the recursive systems and needs weakening to accommodate border cases. I propose relaxing the constraint of *mutual exclusivity* to *disjunction*. Correspondingly, two types of systemic networks emerge differing by *the relation among choices*: the original Hallidayan XOR systems (such as POLARITY TYPE in figure 2.5) and the OR systems for accommodating cases of multiple feature selections (such as SYSTEMIC TYPE in the same figure).

As system is expanded in delicacy to forms a systemic network of choices. Choice of a feature in one system becomes the entry condition for choices in more delicate systems below. I turn now to discuss the relationship types of relationships forming entry conditions to more delicate systems. For instance, an increase in delicacy can be seen as a taxonomic "is a" relationship between features of higher systems and lower systems like in the case of POLARITY TYPE and NEGATIVE TYPE in figure 2.5.
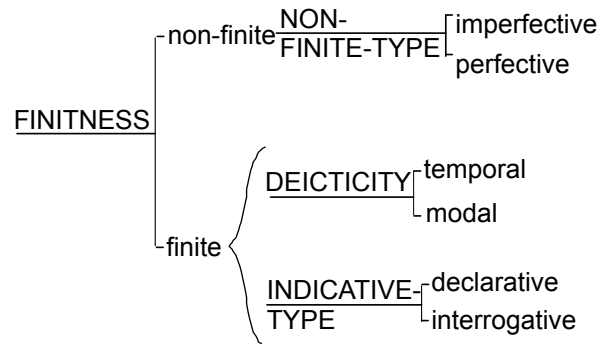
Fig. 2.6 A fraction of the finiteness system where increase of delicacy is not "is a" relation

The activation relation among systems in the cline of delicacy is not always taxonomic. Another relation is "enables selection of" without a any sub-categorisation implied. For example see FINITENESS system in figure 2.6 where in case that the finite option is selected then what this choice enables are not subtypes of finite but merely other options that become available i.e. DEIXIS and INDICATIVE TYPE. The latter is there because selection of finite implies also selection of indicative feature in a FINITNES's sibling system MOOD-TYPE comprised of options indicative vs. imperative.

In this subsection I defined the system and systemic network, presented two system types by the relationship between their choices and distinguished two kinds of activation relations between systems on the cline of delicacy.

2.4.6   coordination as unit complexing

In SG unit complexes fill an important part of the grammar along with the *taxis relations* which express the interdependency relations in unit complexes. *Parataxis* relations bind units of equal status while the *Hypotaxis* ones bind the dominant and the dependant units. Fawcett bypasses the taxis relations replacing them with coordination and embedding (Fawcett 2000: p. 271) seemingly an oversimplified approach leading to abandonment of unit complexing as well. While embedding accounts for the depth and complexity of syntax his approach to coordination is problematic. I further discuss

and argue for utility of unit-complexes for the coordination but this idea can be further extended to other phenomena which involve fixed idiomatic structures such as comparatives or conditionals.

Coordination is challenge not only for SFL but for other linguistic theories as well. CG treats this phenomena as two or more units filling or expounding the same element. For example, in table 2.6 "his shirt" and "and his jeans" are two nominal groups that are siblings and both of them fill the same complement.

Ike	*washed*	*his*	*shirt*	*and*	*his*	*jeans*
Subject	Main Verb	Complement				
		Nominal Group		Nominal Group		
		Deictic Determiner	Head	&	Deictic Determiner	Head

Table 2.6 Coordination analysis in Cardiff Grammar

In SG the coordination is analised as a *complex unit* held together through paratactic relations ensuring that only one unit fills an element of the parent constituent in our example the complement of the clause. The table 2.7 illustrates an example analysis involving the complex unit approach.

Ike	*washed*	*his*	*shirt*	*and*	*his*	*jeans*
Subject	Predicate/Finite	Complement				
		nominal group complex				
		1		+2		
		Deictic	Thing	&	Deictic	Thing

Table 2.7 Coordination analysis in Sydney Grammar

The opinions are divided by whether to invite the notion of a complex unit to handle coordination or not. If we dismiss the unit complex then an element could be filled by more than one units and if we adopt it then the complexing relations need to be accounted along with unit class and what is its structure.

I would argue for adoption of such unit type for two reasons. First, only units are accounted for structure while the elements can only be filled by an unit. Allowing multiple units to fill an element requires accounting at least for order if not also for the relation between the units. The structure as it is described in theories of grammar by Halliday (Halliday 2002) and Fawcett (Fawcett 2000) is defined in the unit and not the element. A unit has a specific possible structure in terms of places of elements however if an element is filled by two units simultaneously it constitutes a violation of the above principle as the order of those units is not accounted for but it matters.

(10) (Both my wife and her friend) arrived late.

(11) * (And her friend both my wife) arrived late.

(12) I want the front wall (either in blue or in green).

(13) * I want the front wall (or in green either in blue).

If the order would not have mattered then we could say that the conjunctions from the example 10 can be reformulated into 11 and the one from 12 into 13. But such reformulations are grammatically incorrect. Obviously the places do matter and they need to be described in the unit structure.

Secondly, the lexical items that signal the conjunction are not a part of the conjuncted units. This is contrary to what is being described in Cardiff and Sydney grammars. Fawcett present the Linker elements (&) which are filled by conjunctions as parts of virtually any unit class placed in the first position of the unit. Halliday omits to discuss in IFG (Halliday & Matthiessen 2013) the place of Linkers but implicitly proposes the same as Fawcett through his examples of paratactic relations at various rank levels where the lexical items signalling conjunction are included in the units they precede.

For example in the "or in green" the presence of "or" signalls the presence at least of one more unit of the same nature and does not contribute to the meaning of the prepositional group but to the meaning outside the group requiring presence of a sibling. Even more, the lack of a sibling most of the time would constitute an ungrammatical formulation. I say sometimes because it is perfectly acceptable to start a clause/sentence with a conjunction most often "but". But even in those cases it still invites the presence of a sibling clause/sentence preceding the current one to be resolved at the discourse level.

So conjunctions and pre-conjunctions shall not be placed as elements of the conjuncted units because they do not contribute to their meaning.

Adopting the unit complex and in particular coordination unit requires two clarifications (1) does the unit complex carry a syntactic class, and if so according to which criteria is it established? (2) Does it have any intrinsic features or not?

Zhang states in her thesis that the coordinating constructions do not have any categorial features thus there is no need to provide a new unit type. Instead the categorial properties of the conjuncts are transferred upwards (Zhang 2010). For example if two nominal groups are conjuncted then the complex receives the nominal class. This principle holds for most of the cases however there are rare cases when

the units are of different classes. Consider the example 14 where the conjuncts are a nominal group "last Monday" and a prepositional group "during the previous weekend".

(14)   I lost it (either last Monday or during the previous weekend).

In this case there are two unit types that can be raised and it is not clear how to resolve this case. Options are to leave the class unspecified, transfer the class of the first unit upwards, or semantically resolve the class as both represent temporal circumstances even if they are realised through two different syntactic categories. Another option is to leave the class generic and assign the conjunction unit the class of *"coordination complex"* without sub-classifying it according to the constituent units below, i.e. without upward unit class transfer.

I address the second question regarding the intrinsic features of the complex unit. The coordination complex can have categorial features which none of the constituting units has. In the example 15 the conjunction of two singular noun groups requires plural agreement with the verb. Even though semantic interpretation that only one item is selected at a time, syntactically both items are listed in the clause and attempting third person singular verb forms like in 16 is grammatically incorrect.

(15)   A pencil or a pen **are** equally good as a smart-phone.

(16)   * A pencil or a pen **is** equally good as a smart-phone.
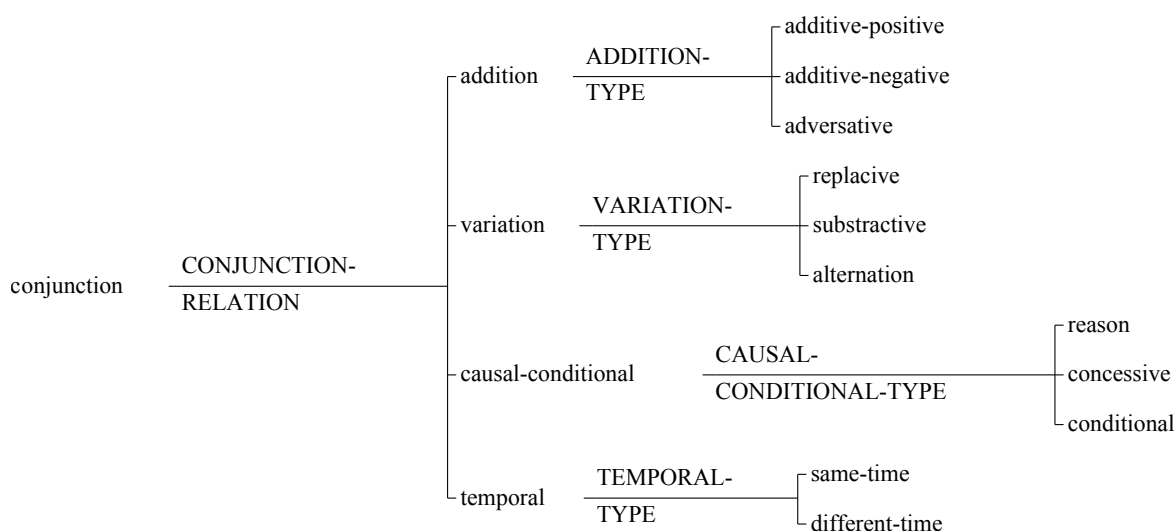
Fig. 2.7 Systemic network of coordination types

In the case of nominal group conjunction we can see that the plural feature emerges even if each individual unit is singular. For other unit classes it is not so obvious whether there are any linguistic features that emerge to the conjunction level. The

meaning variation is rather semantic as for example conjunction of two verbs or clauses might mean very different things like consecutive actions, concomitant actions or presence of two states at the same time and so on. This brings us to another feature of the coordination complex - the type of the relationship it constructs. The lexical choice of head conjunct is the indicator of relationship among the conjuncts. Either *and*, *or*, *but*, *yet*, *for*, *nor* or *so* they express different meanings which are well representable as a relationship types systemic network in figure 2.7.

Adopting the unit complexing enables various kinds of constructions and coordination is only one of them. Below we discuss taxis relations and their role in unit complexing.

2.5 Critical discussion of the grammatical units

Now that the important theoretical details have been covered I would like focus on the grammars of the two schools. They have common parts and also differ in large parts on their paradigmatic and syntagmatic descriptions. This section discusses the main units considered in each of the grammars. Like in the previous section I argue on pragmatic grounds for adoption of unit structures from either grammar. This may seem like an inconsistency add below I try to convince you of the opposite below. The argument runs along the line that some unit structures are closer to the syntactic analysis and thus are easier to detect and parse and the other ones may be a level of abstraction higher falling more on the semantic grounds thus becoming more difficult to capture in structural variance and requiring lexico-semantic resources.

For the reasons of limited space I skipped introducing the Sydney and Cardiff grammars and in turn assume that the reader is familiar with the details fo both of them. And for a general overview of the unit structure in each of the grammars please refer to Appendix 9.6. Nevertheless, as it is a parallel contrastive discussion, even if the reader is familiar with one grammar only I hope it becomes clear how does certain phenomena are dealt with in the other one.

2.5.1 The verbal group and clause division

In SG the verbal group is described as an expansion of a verb just like the nominal group is the expansion of the noun(Halliday & Matthiessen 2013: p.396). There are certainly words that are closely related and syntactically dependent on the verb all together forming a unit that functions as a whole. For example the auxiliary verbs, adverbs or

the negation particles are words that are directly linked to a lexical verb. The verb group functions as Finite + Predicator elements of the clause in mood structure and as Process in Transitivity structure.

In CG the verb group is dissolved moving the Main Verb as the pivotal element of the Clause unit. All the elements that form the clause structure and those that form the verb group structure are brought up together to the same level as elements of a clause. The clause structure in CG comprises elements with clause related functions(like Subject, Adjunct, Complement etc.) and other elements with Main Verb related functions(Auxiliary, Negation particle, Finite operator etc.).

Regarding from the Hallidayan rank scale perspective, merging the elements of the verb group into clause structure is not permitted because the units are of different rank scales. However it is not a problem for the relaxed rank scale version presented in subsection 2.4.1. The reason for adopting such an approach is best illustrated via complex verb groups with more than one non-auxiliary verb such as in examples 17–19.

Next I address the impact of this merger on (a) the clause structure (b) the clause boundaries and (c) semantic role distribution within the clause.

(17)   (The commission **started to investigate** two cases of overfishing in Norway.)

(18)   (The commission **started** (**to investigate** two cases of overfishing in Norway.))

(19)   (The commission **started** (**to finish** (**investigating** two cases of overfishing in Norway.)))

In SG "started to investigate" (example 17) is considered a single predicate of investigation which has specified the aspect of event incipiency despite the fact that there are two lexical verbs within the same verbal group. The "starting" doesn't constitute any kind of process in semantic terms but rather specifies aspectual information about the investigation process. The boundaries of the clause governed by this predicate stretch to entire sentence.

Semantically it is a sound approach because despite the presence of two lexical verbs there is only one event. However allowing such compositions leads to unwanted syntactic analysis for multiple lexical verb cases like in example 19.To solve this kind of problems Fawcett dismisses the verb groups and merges their elements into clause structure. He proposes the syntactically elegant principle of *"one main verb per clause"* (Fawcett 2008). Apply this principle to the same sentence yields a structure of two clauses illustrated in example 18 where the main clause is governed by the verb "to start" and the embedded one by the verb "to investigate". Note the conflict between "one main verb per clause" with Halliday's principle that only whole units form the

constituency of others (the (c) principle of rank scale described in subsection 2.4.1). So allowing incomplete groups into the constituency structure would breach entire idea of unit based constituency.

Semantically the clause in SFL is a description of an event or situation as a figure with a process, participants and eventually circumstances where the process is realised through a lexical verb. Looking back to our examples, does the verb "to start" really describes a process or merely an aspect of it? Halliday treats such verbs as aspectual and when co-occurring with other lexical verbs are considered to form a single predicate. Accommodating Fawcett's stance, mentioned above and contradicting Halliday's approach, requires weakening the semantic requirement and allowing aspectual verbs to form clauses that contribute *aspectually or modally* to the embedded ones. I mention also the modal contribution because some verbs like *want, wish, hope*, etc. behave syntactically exactly like the aspectual ones. Moreover, Fawcett introduces into CG Transitivity network "influential" process type including all categories of meanings that semantically function as process modifiers: tentative, failing, starting, ending etc.

Fawcett's "one main verb per clause" principle changes the way clauses are partitioned, leads to abolition of the verbal group and introduces the "influential" process type.

2.5.2   The Clause

It is commonly agreed in linguistic communities that the unit of clause is one of the core elements in human language. It is consider the syntactic unit that expresses semantic units of a situation referring to a potentially rich array of meanings. The clause structure has been studied for long time and the main clause constituents are roughly the same in SFL as the ones in the traditional grammar (Quirk et al. 1985), transformational grammar (Chomsky 1957) an indirectly in dependency grammar (Hudson 2010).

In current work I adopt the CG Clause structure with the *Main Verb* as pivotal element. Though there is no element that is obligatorily realised in English language, the Main Verb is realised with a reliably high degree of probability. Exceptions are the minor clauses (exclamations, calls, greetings and alarms) that occur in conversational contexts and elliptical clauses Halliday & Matthiessen (2013) such as the one in example 20 which are not currently covered.

(20)   They were in the bar, *Dave in the restroom and Sarah by the bar.*

As mentioned before **??** the elements of a structure are defined in terms of their function contributing to the formation of the whole unit. The elements of an English clause are *Subject*, *Finite*, *Main Verb*(a part of the Predicator), up to two *Complements* and a various number of *Adjuncts*. All the elements of the assumed verbal group are part of the clause as well such as Auxiliary Verbs, Main Verb Extensions, Negators etc. (see Appendix 9.6 for a complete list).

2.5.3   The Nominal Group

The nominal group expresses things, classes of things or a selection of instances in that class. In the table 2.8 is presented an example analysis of the nominal group proposed in Sydney grammar (Halliday & Matthiessen 2013: pp. 364–369).

those	*two*	*old*	*electric*	*trains*	*from Luxembourg*
pre-modifier				head	post-modifier
Deictic	Numerative	Epithet	Classifier	Thing	Qualifyier
determiner	numeral	adjective	adjective	noun	prepositional phrase

Table 2.8 The example of a nominal group in Sydeny Grammar

In SG it is constituted by a head nominal item modified by descriptors or selectors such as: *Deictic*, *Numerative*, *Epithet*, *Classifier*, *Thing* and *Qualifier*. Each element has a fairly stable correspondence to the word classes, expected to be expounded by lexical items. The table 2.9 presents the mappings between the functions and the word classes.

Experiential function in noun group	class (of word or unit)
Deictic	determiner, predeterminer, pronoun
Epithet	adjective
Numerative	numeral(ordinal,cardinal)
Classifier	adjective, noun
Thing	noun
Qualifier	prepositional phrase, clause

Table 2.9 The mapping of noun group elements to classes

The elements in CG differ from those of SG. The table 2.10 exemplifies a noun group analysed with CG covering all the possible elements. The table 2.11 provides a legend for CG acronyms along with mappings to unit and word classes that can fill each element.

or	a photo	of	part	of	one	of	the best	of	the	fine	new	taxis	in Kew	,
					pre-modifiers							head	post-modifiers	
&	rd	v	pd	v	qd	v	sd or od	v	dd	m	m	h	q	e

Table 2.10 The example of a nominal group in Cardiff Grammar

symbol	function meaning	class (of word or unit)
rd	representational determiner	noun, noun group
v	selector "of"	preposition
pd	partitive determiner	noun, noun group
fd	fractional determiner	noun, noun group, quantity group
qd	quantifying determiner	noun, noun group, quantity group
sd	superlative determiner	noun, noun group, quality group, quantity group
od	ordinative determiner	noun, noun group, quality group
td	tipic determiner	noun, noun group
dd	deictic determiner	determiner, pronoun, genitive cluster
m	modifier	adjective, noun, quality group, genitive cluster
h	head	noun, genitive cluster
q	qualifier	prepositional phrase, clause
&	linker	conjunction
e	ender	punctuation mark

Table 2.11 The mapping of noun group elements to classes in Cardiff grammar

The elements in CG have are based on semantic criteria supported by lexical and syntactic choices. Consequently some elements cannot be derived based on solely syntactic criteria requiring semantically motivated lexical resources. Semantically bound elements are predominantly determiners *Representational, Partitive, Fractional, Superlative, Typic Determiners* while the rest of the elements: *Head, Qualifier, Selector, Modifier and Deictic, Ordinative and Quantifying Determiners* can be determined solely on the syntactic criteria. The latter correspond fairly well to Sydney version of nominal group which is adopted in present work with the benefits of relaxed rank system replacing the sub-structures with embedded units and simplifying the syntactic structures.

Another simplification is renouncing to distinction between the Head and Thing (Halliday & Matthiessen 2013: p. 390–396) for the semantic ambiguity reasons as the determiners in CG. Thus if the logical Head of the nominal group is a noun then it is labelled as the Thing leaving the semantic discernment as a secondary process and out of the current scope. Otherwise, in cases of nominal groups without the Thing element, if the Head is pronoun(other than personal), numeral or adjective (mainly

superlatives) then they function as Deictic, Numerative or Epithet. So I propose to parse the nominal groups in two steps: first determine the main constituting chunks and assign functions to the unambiguous ones and second perform a semantically driven evaluation for the less certain units.

In other cases the Thing is present but it is different from the Head. Consider example 21. In Sydney grammar they're treated as a nominal groups with qualifiers introduced by the *"of"* preposition. But these nominal groups are not really about the "cup", "some" or "another one" but rather about "tea", "youngsters" and "eruptions".

(21)   (a cup) of (tea)

(22)   (some) of (those youngsters)

(23)   (another one) of (those periodic eruptions)

So then the syntactic Head still remains the first noun in the nominal group, but then by a semantic evaluation the Thing is shifted into the Qualifier introduced by "of" preposition. Cardiff Grammar weakens the assumption that every prepositional phrase acts as Qualifier in a nominal group and it the special case of the preposition "of". It is allowed to act not as the element introducing a prepositional phrase but as a end mark of a determiner-like selector. Thus making the former noun group a determiner in the latter one. This approach shifts the noun group head into the position of semantically based Thing and erases the discrepancy problem between them. Nonetheless this is not straight forward solution as it requires lexical-semantic informed decision. For example in 24 (Head/Thing marked in bold) the preposition "of" introduces Qualifiers.

(24)   He was the **confidant** of the prime minister.

(25)   It was the **clash** of two cultures.

The distinction between cases when the proposition "of" introduces a Qualifier or ends a Selector/Deictic requires a semantic evaluation answering the questions "what is the Thing that this nominal group is about?". While it is easy to just assume that the first noun in the nominal group is the head.

Therefore, I propose to parse the nominal groups in two steps: first determine the main constituting chunks and assign functions to the unambiguous ones and then in the second step to perform a semantically driven evaluation for the less certain units. This evaluation can be performed by further capturing the structure of nominal groups that act as Dyslectics through their lexico-syntactic realisation patterns.

2.5.4   The Adjectival and Adverbial Groups

Following the rationale of head-modifier like in the case of nominal groups, the adjectives and adverbs function as pivotal elements to form groups. The structure of adverbial and adjectival constructions is briefly covered in Sydney grammar in terms of head-modifier logical structures without an elaborated experiential structure like in the case of nominal groups. While the adverbial group is recognised as a distinct syntactic unit the adjectival group is treated as a special case of nominal group specifically as a sub-structure of Epithet or Classifier elements.

(26)   He is *very lucky.*

For example "very lucky" in 26 is analysed as a short form of the nominal group "very lucky one" where "lucky" is the head of the nominal group with a missing Thing element "one". In this example "very" is not nominal modifier, it does not modify the missing nominal head but the adjective "lucky" so they constitute a head-modifier structure filling the Epithet element and as the rank scale system does not allow groups to fill elements of groups then it is described as a substructure of the nominal group.

In SG "The adverbial group has an adverb as Head which may or may not be accompanied by modifying elements"(Halliday & Matthiessen 2013: p. 419). The adverbial groups may fill modal and circumstantial adjunct elements in a clause corresponding to eight semantic classes of: time, place, four types of manner and two types of assessment. The adverbial pre-modifiers express polarity, comparison and intensification along with only one comparison post-modifier (Halliday & Matthiessen 2013: p.420-421).

A thorough systemic functional examination has been provided for the first time by Tucker Tucker (1997, 1998) materialised into a lexical-grammatical systematization of adjectives and the structure of Quality Group in CG. Fawcett uses the quality group structure for Adverbials as well as they follow similar grammatical behaviour. He avoids calling the group according to the word class but rather refers to the semantic meaning of what both groups express, i.e. the quality of things, situations or qualities themselves. The qualities of things have adjectives as their head while the qualities of situations an adverb.

In Cardiff Grammar, the head of the quality group is called *Apex* while the set of modifying elements: *Quality Group Deictic, Quality Group Quantifier, Emphasizing Temperer, Degree Temperer, Adjunctival Temperer, Scope* and *Finisher*. The quality group most frequently fills complements and adjuncts in clauses and fill modifiers and

superlative determiners in nominal groups but there are also other cases found in the data.

Just like in the case of nominal group the adverbial and adjectival groups in Cardiff grammar are semantically motivated. To automatically identify elements of the quality group requires lexico-semantic resources.

Some adverbs are different from others at least because not all of them can be heads of the adverbial group like for example *very, much, less, pretty* also being able to act as adjectival modifiers whereas others cannot. So a naive attempt is to use a list of words to identify the Emphasizing and Degree Temperers.

Other elements of the quality group like the *Scoper* or *Finisher* are more difficult to identify and localize as part of the group only by syntactic cues and/or lists of words because of their inherent semantic nature. The problem is similar to detecting whether a prepositional phrase is filling a qualifier element in the preceding nominal group or it is filling a complement or adjunct in the clause. Not surprisingly the Scopers and Finishers are most of the time prepositional phrases.

Another issue is continuity. The question is whether a grammar shall allow at least at syntactic level for discontinuous constituents or not. And then if so how to detect all the parts of the group even if they do not stand in proximity of each other. For example, comparatives, a complex case of a quality group, could be realised in a continuous or discontinuous forms. Compare the analyses presented in 2.12 and 2.13. In the first case the comparative structure is a continuous quality group. In the second case the comparative is dissociated and analysed as separate adjuncts.

On one hand it is not a problem treating them as two adjuncts, because that is what they are from the syntactic point of view. However, semantically as Fawcett proposes, there is only one quality group with a discontinuous realisation whose Scope element is placed in a thematic position before the subject. For an automatic process

I	*am*	*much*	*smarter*	*today*	*than*	*yesterday*
Subject	Main Verb	Adjunct				
pronoun	verb	quality group				
		Emphasizing Temperer	Apex	Scope	Finisher	

Table 2.12 Comparative structure as one quality group adjunct

to identify a complex quality group is a difficult task. It needs to pick up queues like a comparative form of the adjective followed by the preposition "than" and then look for two terms being compared. Given some initial syntactic structure such patterns could be modelled and applied but only as a secondary semantically oriented process.

Today	I	am	much	smarter	than	yesterday
Adjunct	Subject	Main Verb	Adjunct			
adverb	pronoun	verb	quality group			
			Emphasizing Temperer	Apex	Finisher	

Table 2.13 Comparative structure split among two adjuncts

Since both the adverbial and adjectival groups have similar structures, it is syntactically feasible to automatically analyse them in terms of head-modifier structures in a first phase followed by a complementary process which assigns functional roles to the quality group components.

2.6   Discussion

This chapter introduces the fundamentals of systemic functional linguistics and presents an adaptation of Sydney and Cardiff theories of grammar to the task of parsing.

Because of bottom up approach to unit structure, rank scale relaxation and accommodation of embedding as a general principle Cardiff systemic functional theory is more suitable for parsing than Sydney one. Nonetheless the unit definitions in Cardiff grammar are deeply semantic in nature. Parsing with such units requires most of the time lexical-semantic informed decisions beyond merely syntactic variations. This is one of the reasons why the parsing attempts by O'Donoghue (1991) and others in COMMUNAL project were all based on a corpus.

As there was no corpus available and because the parsing approach is based on syntactic backbone none of the theories could be fully used as such. The second part of the chapter attempts to merge and adapt the grammars and theories of grammar to this parsing approach.

Next chapter lays the theoretical foundations of Dependency Grammar and introduces the Stanford dependency parser used as a departing point in current parsing pipeline. Because there is a transformation step from dependency to systemic functional consistency structure, next chapter also covers a theoretical compatibility analysis and how such a transformation should in principle look like.

Chapter 3

The dependency grammar

The Stanford dependency analysis of a given text constitutes the input for the algorithm developed in the current work. It provides the foundation to build the syntactic backbone used adopted here. This chapter offers an overview of the grammar and the parser developed at the Stanford university. In the last part of the chapter is discussed the cross theoretical connection between the dependency and systemic functional grammars.

3.1 A briefing on the theory of dependency grammar

Traditionally, Latin language tended to be analysed with the *dependency model* based on theories of what was called *government.* It explains the syntagmatic relations of the subtype Firth called *colligation* (i.e. relations between grammatical categories). Government is a way of explaining the rich inflections of language (such as Latin) in terms of how particular words govern, that is to say, determine, the inflection of other words. (McDonald 2008: p.66)

Tensiere (2015) explains how government works by giving example of Latin text analysis where the inflections immediately give aloto of information about relations between different words. For example the verb agree in person and number with its subject. From this point of view the verb governs the subject. The verb also governs complements and adjuncts which can be seen as relations of dependency between a governing element or controller and a governed element or dependant. It is important to note that Tensiere analysed syntax at the level of clause where he identified a verb node as the main controller.

Contrary to Latin, languages like English or Chinese, where there is little or most of the time none of the inflectional marking to identify dependency relations, are much harder to analyse in terms of such relations. This was a motivation for Tensiere to reinterpret dependency relation in semantic terms rather than inflectional marking. As McDonald (2008) points out, this can be regarded as extending syntagmatic relations of the clause to include what Firth was calling *collocation* relations (i.e. links between lexical items). Tensiere framed his theory in terms of syntagmatic relations as expressing a model of experience. He compares the verbal node of the clause to a complete little drama. Like a drama, it obligatorily consists of an action, most often actors and features of settings. Expressed in terms of syntactic structure, the action, actors and settings become the verb, participants and circumstances (Tensiere 2015).

Further, Tensiere explains *categories of language* as *categories of thought.* The human mind shapes the world in its own measure by organising experience into a systematic framework of ideas and beliefs called categories of though. Likewise, the language shapes thought in its own measure by organising it into a systematic framework of grammatical categories (Tensiere 2015). He stresses though, that the latter ones can vary considerably from language to language and that the analysis of syntactic relations shall be carried on not in terms of grammatical categories but rather in terms of functions. He explain through an example that analysis in terms of nouns and verbs i.e. grammatical categories, tells nothing about the tie that links the words, whereas if we turn to notions such as subject and complement it all of the sudden becomes clear: the connections are established, the lifeless words become a living organism and the sentence take on a meaning (Tensiere 2015).

3.2   Stanford dependency grammar (and parser)

Dominant Chomsky (1981) theories define grammatical relations as configurations of *phrase structure* representations, which is nesting of multi word constituents. Other theories such as Lexical-Functional Grammar reject the adequacy of such an approach (Bresnan 2010) and advocate a functional representation for syntax.

When motivating its stance, Marneffe et al. (2006) insists on practical rather than theoretical concerns proposing that structural configurations be defined as grammatical roles (to be read as grammatical functions)(Marneffe et al. 2006). For example she insists that, information about functional dependencies between words grants direct access to the predicate-argument relations which are not readily available form the phrase structure parses and can be used off the shelf for real world applications. She

avoids going into theoretical debates and focuses on the suitability of the grammar for parsing within the context of relation extraction, machine translation, question answering and other tasks.

The functional dependency descriptions is precisely the aspect which makes possible the link between the Stanford Dependency Grammar and Systemic Functional Structures targeted in the current thesis.

The design of Stanford dependency set (Marneffe et al. 2006; Marneffe & Manning 2008a; Marneffe et al. 2014; Natalia Silveira et al. 2014) bears a strong intellectual debt to the framework of Lexical Functional Grammars (Bresnan 2010). Marneffe et al. (2006) started designing the relation typology from GR (Carroll et al. 1999) and PARK (King & Crouch 2003) schemes following a LFG style and according to the principles described in Generalization 3.2.1.

Generalization 3.2.1 (Design principles for Stanford dependency set)**.**

1. Everything is represented uniformly and some binary relation between two sentence words.

2. Relations should be semantically contentful and useful to applications.

3. Where possible, relations should use notions of traditional grammar (Quirk et al. 1985) for easier comprehension by users.

4. The representation should be spartan rather than overwhelming with linguistic details.

When proposing the Stanford dependency, Marneffe et al. (2006) inherits many relations from Lexical Functional Grammars(Bresnan 2010) and departs from the sets described by Carroll et al. (1999) and King & Crouch (2003). She arranges the grammatical relations into a hierarchy rooted in a generic relation *dependent*. This is then classified into a more fine-grained set of relations between a head and its dependent.

3.3   Stanford Parser: collapsed-cc output

The Stanford Dependency Parser is capable of generating several types of dependency representations. The most convenient and informative version is Collapsed-CC-processed. This structure is created after the initial parse is ready and constitutes a simplified and more intuitive representation of the dependency parse. The collapsed

Fig. 3.2 Collapsed preposition dependency
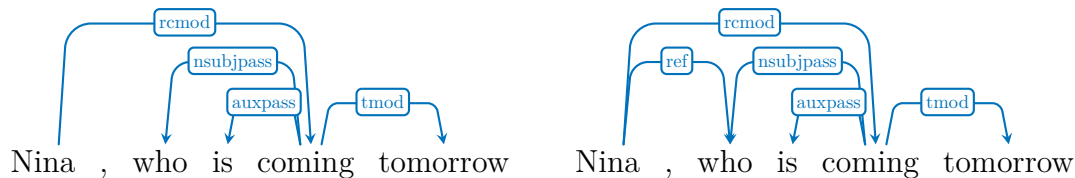
Fig. 3.4 Collapsed preposition dependency

form concerns prepositions, conjunctions and relative clause referents. Dependencies involving prepositions and conjunctions are transformed from basic form in Figure 3.1 to the form where preposition are embedded into the edge relation, as in Figure 3.2. Similar transformation is done for conjunctions.

Besides collapsing prepositions and conjunctions the DG is further processed to introduce more relations (i.e. connections) even if they break the tree structure. For example the *ref* relation does not appear in the basic dependency structure in figure 3.3 but it appears in the processed dependency structure forming a cycle with *rcmod* and *subjpass* relations.

Relations like *ref* introduce cycles but add valuable information useful in various stages of further processing. However, ensuring a tree structure in important for the CG creation stage because it is based on a top down traversal. This is taken care of in the preprocessing stage described in the next stage.

3.4   Penn part-of-speech tag set

Stanford dependency parser starts creation of the parse structure process from the list of tokens annotated with Penn part-op-speech tags. Embedded into the dependency graph, these tags are the part of the syntactic context from which SFG constituency graph is built.

The Penn tagset was developed to annotate the Penn Treebank corpora (Marcus et al. 1993). It is a large, richly articulated tagset that provides distinct coding for classes of words that have distinct grammatical behaviour.

It is based on the Brown Corpus tagset (Kucera & Francis 1968) but differs in several ways from it. First, the authors reduced the lexical and syntactic redundancy. In Brown corpus there are many unique pos tags to a lexical item. In Penn tagset the intention is to reduce this phenomena to minimum. Also distinctions that are recoverable from lexical variation of the same word such as verb or adjective forms or distinctions recoverable from syntactic structure were reduced to a single tag.

Secondly the Penn Corpus takes into consideration the syntactic context. Thus the Penn tags, to a degree, encode the syntactic functions when possible. For example *one* is tagged as NN (singular common noun) when it is the head of the noun phrases rather than CH (cardinal number).

Thirdly Penn POS set allows multiple tags per word, meaning that either it cannot be decided or the annotators may be unsure of which one to choose.

There are 36 main POS and 12 other tags in Pen tagset. A detailed description of the schema, the design and principles and annotation guidelines are described in (Santorini 1990).

3.5   Cross theoretical bridge from DG to SFL

The concept of dependency between pairs of words is long acknowledged in linguistic communities. In traditional terms dependencies are treated as *hypotactic expansions* (see Definition 2.3.11) of word classes (or parts of speech) where the expanded word acts as *heads* and expanding ones as *dependent* establishing parent-daughter structural relations illustrated in Figure 3.5a.

In SFL community the concept of dependency is less salient than the foundational role it plays in the Dependency Grammars. Dependencies are regarded as orthogonal relations between sibling elements of a unit (Figure 3.5b) and link the *heads* to their *modifies* in by Hallidayan *logical structure*(Halliday & Matthiessen 2013).

Figure 3.5 illustrates side by side the parent-daughter and sibling dependency relations. In Figure 3.5a dependency are the only relations between the units of structure whereas in Figure 3.5b there are multiple levels (ranks) of units and the dependency relations are relevant only between siblings at the same level within the structure of an unit. SFL regards dependency relations holding only between elements of a unit whereas the relations that connect the units of lower and higher rank are *constituency relations*. Yet when we look at the two structures they resemble in a way each other and next I show how.

(a) The parent-daughter relations in DG (b) The head-modifier relations in SFG
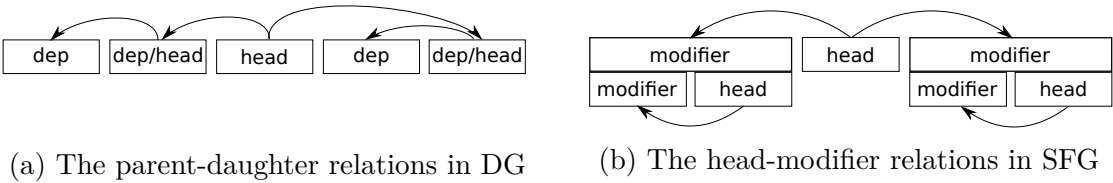
Fig. 3.5 The dependency relations in DG and SFG

In a nutshell, the parent-daughter dependency relations in dependency grammar unpack into multiple function in systemic functional grammar and specifically it is the head-modifier indirect relation, unit-element componence relation and the head of unit a "representativeness" function.

This difference implies that, when translated to a constituency unit (described in Section 6.4), the dependency unit, stands for both a unit and that unit's head element. In other words a DG node corresponds to two functional elements at different rank scales. For example the root verb in dependency graph corresponds to the clause node and the lexical item which fills the Main Verb of the clause. By analogy, the head noun of a Nominal Group anchors the entire unit (as a group) and fills the head element of the group.

text	some	very	small	wooden	ones
units	Nominal Group				
elements	*Quantifying Determiner*	*Modifier*		*Modifier*	*Head*
units		Quality Group			
elements		*Degree Tamperer*	*Apex*		

Table 3.1 SF analysis of Example 3 (reproduced from Table 2.4)
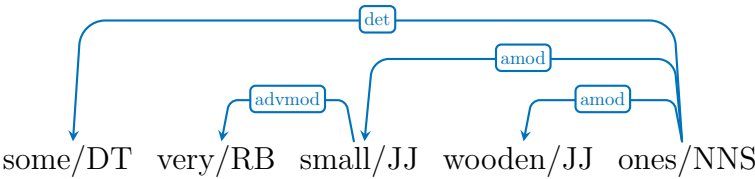
Fig. 3.6 Dependency analysis for Table 3.1

Figure 3.6 and the Table 3.1 represent the analysis of a nominal group from Example 3 ("some small very small wooden ones") in Cardiff grammar and Stanford dependency grammar exhibiting a contrast of the two structures. Consider the dependency relation "det" a link between the noun "ones" and the determiner "some". When translated into SF variant the dependency relation stands within Nominal Group between the

Head element (filled by word "ones") and the Quantifying Determiner element (filled by the word "some"). By definition all elements in a unit are equal in the structure so the Head and Quantifying Determiner are siblings. So the items (words) filling those elements are sibling. How is then the dependency relation established?

In SFL there is the concept of Head and Modifier. There is no direct relationship between them but it is said that what the Modifier modifies is the Head. The relation is not a direct one, the Modifier and Head stand for two different kinds of meaning and what the Modifier modifies is not the Head per se but the referent denoted by the head (and thus construed by the entire unit). It is precisely this modification of the head that is called a (sibling) dependency relation and is seldom mentioned in SFL literature because it is considered implicit and recoverable from the SF constituency structure.

The Head also is the element that anchors the entire unit and plays a constitutive role. In this sense the word "ones" realizes not only the Head function (sided with Determiner "some") but also anchors the entire unit. The relation between the group and it's elements is one of *componence* (Definition 2.3.4) described in Section 2.3.4. Yet in the role of unit anchor we cannot say that there is a componence relation between "one" and "some" because it is merely a proxy to the referent rather than the entire unit. So in this role "one" can be said to be standing in a parent-daughter dependency relation to "some" incorporating the filling and componence relations.

I just showed how the dependency relation in dependency structure (Figure 3.5a) can be unpacked into two relations in systemic functional structure (Figure 3.5b): sibling dependency considered an indirect relation between Head and Modifier (Logical Metafunction) and parent-daughter dependency between unit anchor and the compounding elements, relation which resembles unit componence but is not.

Lets look at a second example of two relations "advmod" from "small" to "very" and "amod" from "ones" to "small". The interesting case here is the item "small" which is the Head of the Quality Group, it anchors the meaning of the whole group and the Quality group fills the Modifier element within the Nominal group. What is not covered in previous example is that the Apex "small" not only is a representative of the entire group but it also is a *representative filler* of the Modifier element within Nominal Group. Using the similar translation mechanism as above, this means that, the incoming dependency needs to be unpacked into three levels: the element within the current group (Modifier), the unit class that element is filling (Quality Group) and finally the head of the filler group (Apex). In fact, to be absolutely correct there is one

more level. The elements of a unit are expounded by lexical items, so fourth relation to unpack is the expounding of the Apex by "small" word.

In this section I laid the theoretical principle for transforming the dependency structure into systemic functional structure. In practice to achieve this level of unpacking the algorithm requires a bottom up and a top down contextualization in terms of elements of structure within a unit and realised sequence of textual units. This implied that the unpacking needs two traversals, a bottom-up and a top one. More on that and the exact algorithm for the translation is provided in Section 6.4.

Next follows the chapter on Governance and Binding Theory needed to account for the unrealised, covert (Null) Elements in the syntactic structure. It is also an opportunity to perform a similar theoretical translation exercise (as in this section) from one theory of grammar into another.

Chapter 4

Governance and binding theory

Government and Binding Theory (GBT) is a theory of syntax based on phrase structure grammar and is a part of transformational grammar developed by Noam Chomsky (1981). It explains how some constituents can *move* from one place to another, where are the places of *non-overt constituents* and what constituents do they refer to i.e. what are their *antecedents*.

Knowing that some constituents are not realised in certain places with specific syntactic functions greatly benefits semantic analysis process. GBT is important for Transitivity parsing which is semantic in nature but still systematised at the level of abstraction that enables capturing the grammatical variation.

GBT approach to explain grammatical phenomena using *phrase structures* (PS) is more distant from SFG than the main approach taken by the dependency grammar. Section 4.2 briefly introduces the theoretical context of GBT and then formulates the principles and generalisations relevant for current work. Then Section 4.3 translates the introduced principles and generalisations into DG rules and patterns. To lay the ground for the two sections, I first place GBT into the context of transformational grammar.

4.1   On Transformational grammar and parse trees

The notion of structure in generative grammar means the way words are combined together to form phrases and sentences. *Merging* is the technical term for the operation of combining two words together into a phrase. In this operation one word will always be more prominent called the *head* of the phrase and the resulting combination is a new constituent which is a *projection* of the head.

These combinations of words and projections can be represented using the *labeled bracketing notation* where the labels denote constituent category. The bracketed notation is a representation equivalent to a hierarchical tree of constituent parts called *parse tree* (aka *syntactic tree*, *phrase structure*, *derivation tree*). The parse tree represents the syntactic structure of a string according to some grammar. The equivalence between a bracketed notation and parse tree is exemplified in the following two representations.

(27) $\left[_{IP}\left[_{D}We\right]\left[_{\bar{I}}\left[_{I}are\right]\left[_{VP}\left[_{V}trying\right]\left[_{IP}\left[_{I}to\right]\left[_{VP}\left[_{V}help\right]\left[_{D}you\right]\right]\right]\right]\right]\right]$

```
              IP
          ___/  \___
         D          I
         |       __/ \__
        We      I       VP
                |     __/ \__
               are   V       IP
                     |     __/ \__
                   trying  I      VP
                           |    __/ \__
                          to   V      D
                               |      |
                             help    you
```
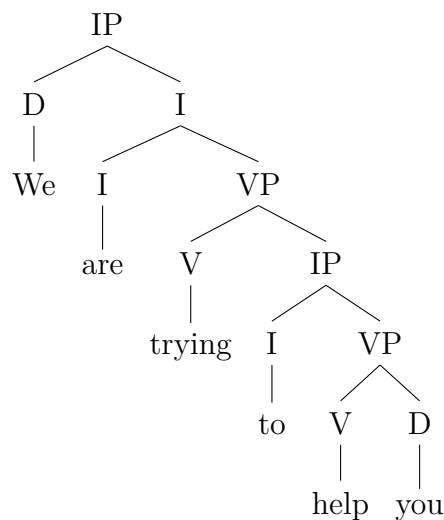
Transformational grammar (TG) or transformational-generative grammar (TGG) is, in the study of linguistics, part of the theory of generative grammar, especially of naturally evolved languages, that considers grammar to be a system of rules that generate exactly those combinations of words which form grammatical sentences in a given language. TG involves the use of defined operations called transformations to produce new sentences from existing ones.

Chomsky developed a formal theory of grammar where transformations manipulated not just the surface strings, but the parse tree associated with them, making transformational grammar a system of tree automata (Stockwell et al. 1973).

A transformational-generative (or simply transformational) grammar thus involved two types of productive rules: phrase structure rules, such as "S -> NP VP" (meaning that a sentence may consist of a noun phrase followed by a verb phrase) etc., which could be used to generate grammatical sentences with associated parse trees (phrase markers, or P markers); and transformational rules, such as rules for converting statements to questions or active to passive voice, which acted on the phrase markers

to produce further grammatically correct sentences (Bach 1966: 59-66). This notion of transformation proved adequate for subsequent versions including the "extended", "revised extended" and Government-Binding (GB) versions of generative grammar, but may no longer be sufficient for the current minimalist grammar, in that merge may require a formal definition that goes beyond the tree manipulation. For the purpose of the current work, however, the GBT describes clearly the mechanisms for identification of *null elements* therefore I find it most suitable and preferred to minimalist program.

4.2   On Null Elements (empty categories)

In linguistics, in the study of syntax, an *empty category* is a nominal element that does not have any phonological content and is therefore unpronounced. Empty categories may also be referred to as *covert nouns*, in contrast to overt nouns which are pronounced. The phenomenon was named by Noam Chomsky in his 1981 LGB framework. Some empty categories are governed by the empty category principle. When representing empty categories in trees, linguists use a null symbol to depict the idea that there is a mental category at the level being represented, even if the word(s) are being left out of overt speech. There are four main types of empty categories: NP-trace, Wh-trace, PRO, and pro.

There are four main types of empty categories: *NP-trace*, *Wh-trace*, *PRO*, and *pro*. The types are differentiated by their two binding features: the anaphoric feature [a] and the pronominal feature [p]. The four possible combinations of plus or minus values for these features yield the four types of empty categories.

[a]	[p]	Symbol	Name of empty category	Corresponding overt noun type
-	-	t	Wh-trace	R-expression
-	+	pro	little Pro	pronoun
+	-	t	NP-trace	anaphor
+	+	PRO	big Pro	none

Table 4.1 Four types of empty categories according to their binding features

In the Table 4.1, [+a] refers to the anaphoric feature, meaning that the particular element must be bound within its governing category. [+p] refers to the pronominal feature which shows that the empty category is taking the place of an overt pronoun.

4.2.1   PRO Subjects

PRO stands for the non-overt NP that is the subject in non-finite (complement, adjunct or subject) clause and is accounted by the *control theory* (CT).

Definition 4.2.1 (Control)**.** Control is a term used to refer to a relation of referential dependency between an unexpressed subject (the control element) and an expressed or unexpressed constituent (controller). The referential properties of the controlled element are determined by those of the controller.(Brensan 1982)

Control can be *optional* or *obligatory.* While *Obligatory control* has a single interpretation, that of PRO being bound to its controller, the *optional control* allows for two interpretations: *bound* or *free.* In 28 the PRO is controlled, thus bound, by the subject "John" of the matrix clause (i.e. higher clause) whereas in 29 it is an arbitrary interpretation where PRO is refers to "oneself" or "himself". In 30 and 31 PRO must be controlled by the subject of the higher clause and does not allow for the arbitrary interpretation.

(28)   John asked how [PRO to behave himself/oneself.]

(29)   John and Bill discussed [PRO behaving oneself/themselves in public.]

(30)   John tried [PRO to behave himself/*oneself.]

(31)   John told Mary [PRO to behave herself/*himself/*oneself.]

Sometimes the controller is the subject (like in Examples 28, 29, 30) and sometimes it is the object (Example 31) of the higher clause. Haegeman (1991: p. 278) proposes that there are verbs of *subject* and of *object control.* However if there is an Complement in the higher clause, as a rule of thumb, it is likely that it is the controller of the PRO in lower clause. Three role cognition verbs like in Example 31 take two complements where one of them is the phenomenon and can be a nominal group or a non-finite clause with PRO subject controlled by the object in higher clause.

The following set of generalizations from Haegeman (1991) are instrumental in identifying places where PRO constituent occurs and its corresponding binding element.

Generalization 4.2.1. Each clause has a subject. If a clause doesn't have an overt subject then it is covert (non-overt) represented as PRO. (Haegeman 1991: p. 263)

Generalization 4.2.2. A PRO subject can be bound, i.e. it takes a specific referent or can be arbitrary (equivalent to pronoun "one") (Haegeman 1991: p. 263) In case of obligatory control, PRO subject is bound to a NP and must be c-commanded by its controller (Haegeman 1991: p. 278).

Generalization 4.2.3. PRO must be in ungoverned position. It means that (a) PRO does not occur in object position (b) PRO cannot be subject of a finite clause.

Generalization 4.2.4. PRO does not occur in the non-finite clauses introduced by *if* and *for* complementizers but it can occur in those introduced by *whether*.

Examples 32 and 33 illustrate generalization 4.2.4

(32)   John doesn't know [whether [PRO to leave.]

(33)   * John doesn't know [if PRO to leave.]

Generalization 4.2.5. PRO can be subject of complement, subject and adjunct clauses. (Haegeman 1991: p. 278)

Generalization 4.2.6. When PRO is the subject of a declarative complement clause it must be controlled by an NP i.e. arbitrary interpretation is excluded. (Haegeman 1991: p. 280)

Generalization 4.2.7. The object of active clause becomes subject when it is passivized and also controls the PRO element in complement clause. (Haegeman 1991: p. 281)

Generalization 4.2.8. PRO is obligatorily controlled in adjunct clauses that are not introduced by a marker. (Haegeman 1991: p. 283)

Adjuncts (clauses or phrases) often are introduced via preposition and these are some of rare cases of adjunct clauses free of preposition. Examples 34 and 35 illustrate marker-free adjunct clauses.

(34)   John hired Mary [PRO to fire Bill.]

(35)   John abandoned the investigation [PRO to save money.]

Generalization 4.2.9. PRO in a subject clause is optionally controlled thus by default it takes arbitrary interpretation.

A default assumption is to assign arbitrary "one" interpretation to each PRO subject in subject clauses. However there are cases when it may be bound (resolved) to a pronomial NP in the complement of the higher clause. Binding element can be either the entire complement or a *pronomial* part of it like the qualifier or the possessor. Example 36 illustrates that PRO has only arbitrary interpretation since cannot be bound to the complement "health". Moreover PRO can also be bound to (a) the possessive element of higher clause - example 37, (b)the complement of the higher clause - example 38 and (c)either the possessives in lower or higher clause, example 39.

(36) PRO_i smoking is bad for the $health_j$.

(37) PRO_i smoking is bad for $your_i$ $health_j$.

(38) PRO_i smoking is bad for you_i.

(39) PRO_i lying to $your_i$ friends decreases $your_i$ $trustworthiness_j$.

4.2.2   NP-traces

The NP-movement in GB theory is used to explain *passivization*, *subject movement* (in interrogatives) and *raising*. The raising phenomena (Definition 4.2.2) is the one that is of interest for us here as it is the one involving an empty constituent.

Definition 4.2.2 (NP-raising)**.** NP-raising is the NP-movement of a subject of a lower clause into subject position of a higher clause.

Consider examples 40 to 43. There are two cases of expletives(40 and 42) and their non-expletive counterparts(41 and 43) where the subject of the lower clause is moved to the subject position of the matrix clause by replacing the expletive. The movement of NPs leaves traces which here are marked as *t*. This phenomena is called *raising* (Definition 4.2.2) or as postal calls it *subject-to-subject raising* Postal (1974).

(40)   It was believed [Poirot to have destroyed the evidence.]

(41) $Poirot_i$ was believed [t_i to have destroyed the evidence.]

(42)   It seems [that Poirot has destroyed the evidence.]

(43) $Poirot_i$ seems [t_i to have destroyed the evidence.]

The subjects "It" and "Poirot" in none of the examples 40-43 receive a semantic role from the main clause. "It" is an expletive and never receives a semantic role while "Poirot" in 41 and 43 takes *Agent* role from "destroy" and is not the *Cognizant* neither of "believe" nor of "seem". So verbs "believe" and "seem" do not theta mark (assign semantic roles) their subjects in none of the examples.

Table 4.2 represents the semantic role distribution for verb senses in examples above. Example 41 is a passive clause with an embedded complement clause. The subjects and complements switch places in passive clauses and so do the semantic roles. That would mean that Cognizant role goes is to be assigned to embedded/complement clause. However Phenomena is the only semantic role that can be filled by a clause, all other roles take nominal, prepositional or adjectival groups.

In example 43 the verb "seem" assigns roles only to complements and as the embedded clauses can take only the phenomenon role, the cognizant is left unassigned

Verb	Process type	canonical distribution of semantic roles
Believe	Cognition	Cognizant + V + Phenomenon
Seem	Cognition	It + V + Cognizant + Phenomenon
Destroy	Action	Agent + V + Affected

Table 4.2 Semantic role distribution for verbs "believe" and "seem"

and so does not assign any thematic role to the subject which then can be filled either by an expletive or a moved NP.

Definition 4.2.3 (Trace, Antecedent and Chain)**.** An empty category which encodes the base position of a moved constituent is referred to as *trace*. The moved constituent is called *antecedent* of a trace. Both the trace(s) and the antecedent form a *chain.*

Raising is very similar to obligatory subject control with a difference in thematic role distribution. In the case of subject control, both the PRO element and it's binder (the subject of higher clause) receive thematic roles in both clauses. However in the case of raising, the NP is moved and it leaves a trace which is theta marked but not to the antecedent. This is expressed in generalization 4.2.10.

Generalization 4.2.10. The landing site for a moved NP is an empty A-position. The chain formed by a NP-movement is assigned only one theta role and it is assigned on the foot of the chain, i.e. the lowest trace. (Haegeman 1991: p. 314)

So, in case of raising, the landing sites for a moved NP are empty subject positions or the ones for expletives. As a result of movement, these positions are filled or the expletives replaced.

4.2.3   Wh-traces

Wh-movement is involved in formation of Wh-interrogatives and in formulation of relative clauses (Haegeman 1991: p. 423). We are interested in both cases as both of them leave traces of empty elements that are relevant for transitivity analysis.

(44)   [What] will Poirot eat?

(45)   [Which detective] will Lord Emsworth invite?

(46)   [Whose pigs] must Wooster feed?

(47)   [When] will detective arrive at the castle?

(48)   [In which folder] does Margaret keep the letter?

(49)   [How] will Jeeves feed the pigs?

(50)   [How big] will the reward be?

Haegeman (1991: p. 375) treats each Wh-element as the head of the Wh-phrase. I do not share this perspective. In some cases they function as heads but there are other cases when they act as determiners, possessors or adjectival modifiers. This issue is extensively discussed by Abney (1987); Quirk et al. (1985); Halliday & Matthiessen (2013).

For clarity purposes I define the Wh-group and Wh-element in 4.2.4. Note that Wh-group is not a new unit class in the grammar but a constituent feature that can span across three unit classes.

Definition 4.2.4 (Wh-group, Wh-element)**.** *Wh-group* is a nominal, prepositional or adverbial group that contains Wh-element either as head or as modifier. The *Wh-element* is a unit element filled by any of the following words or their morphological derivations (by adding suffixes *-ever*, *-soever*): *who, whom, what, why, where, when, how.*

In GBT the *case* occupies an important place in the grammar. Verbs dictate the case of their arguments, a property called case marking. It is established that the Subjects are marked with Nominative case while the Complements of the verb are marked with Accusative case.

In English however case system is very rudimentary as compared to other languages. Hence, *who*, *whom* and their derivatives *whoever* and *whomever* are the only Wh-elements with overt case differentiation. The other Wh-elements *what, when, where, how* do not change their form based on the case.

The example 51 shows that the Accusative (*whom*) is disallowed when its trace is in Subject position requiring Nominative (*who*). In example 52 the reverse holds and the Nominative form is disallowed as the Wh-element moved from Complement position requiring Accusative case.

(51)   Who$_i$/*Whom$_i$ do you think [t$_i$ will arrive first?]

(52)   Whom$_i$/*Who$_i$ do you think [Lord Emsworth will invite t$_i$?]

Another important distinction to be made among English Wh-elements is the *theta-marking*, i.e. the argument and non-argument distinction. The Functional distribution for Wh-elements and Wh-groups is presented in the table **??** below.

Features	Clause functions of the Wh-group		Group functions of Wh-element
	Subject	**Complement**	
person	who, whoever	whom, whomever, whomsoever	head/thing
person, possessive	whose		possessor
person/non-person	which		determiner
non-person	what, whatever		head/thing
	Adjunct		
various circumstantial features	when, where, why, how (whether, whence, whereby, wherein) (and their -*ever* derivations)		head/modifier

Table 4.3 Functions and features of Wh-elements and groups

There are two places where the Wh-Group can land: (a) either in the subject position of the matrix clause changing its mood to interrogative (example 53) or (b) subject position of the embedded clause creating embedded questions (example 54). However regardless of the landing site the movement principle is subject to *that-trace* filter expressed in generalization 4.2.11) and *Subjacency condition* (generalization 4.2.12)

(53)   Whom$_i$ do [you believe [that Lord Emsworth will invite t$_i$]]?

(54)   I wonder [whom$_i$ you believe [that Lord Emsworth will invite t$_i$.]]

Generalization 4.2.11 (That-trace filter)**.** The sequence of an overt complementizer "that" followed by a trace is ungrammatical (Haegeman 1991: p. 399).

The examples 55 to 58 illustrate how the above generalization applies.

(55)   * Whom do you think that Lord Emsworth will invite?

(56)   Whom do you think Lord Emsworth will invite?

(57)   * Who do you think that will arrive first?

(58)   Who do you think will arrive first?

Generalization 4.2.12 (Subjacency condition)**.** Movement cannot cross more than one bounding node, where bounding nodes are IP[1] and NP (Haegeman 1991: p. 402).

Subjacency condition captures the grammaticality of NP-movement and exposes two property of the movement, namely as being *successive* and *cyclic*. Consider the chain creation resulting from Wh-movement in Examples 59 – 61. The Wh-movement leaves (intermediate) traces successively jumping each bounding node.

(59)   Who$_i$ did [he see t$_i$ last week?]

(60)   Who$_i$ did [Poirt claim [t$_i$ that he saw t$_i$ last week?]]

(61)   Who$_i$ did [Poirot say [t$_i$ that he thinks [t$_i$ he saw t$_i$ last week?]]]

What 4.2.12 states is that a Wh-element cannot move further than subject position and so creating the interrogative form or moving outside the clause into subject position of the higher clause and leaving a Wh-trace.

4.3   Placing null elements into Stanford dependency grammar

This section provides a selective translation of principles, rules and generalizations captured in GB theory into the context of dependency grammar. The selections mainly address the identification of null elements which is later used in the semantic parsing process described in Chapter 7

4.3.1   PRO subject

Coming back to the definition of PRO element, it is strictly framed to the non-finite subordinate clauses. In dependency grammar the non-finite complement clauses are typically linked to their parent via *xcomp* relation defined in Marneffe & Manning (2008a) as introducing an open clausal complement of a VP or ADJP without its own subject, whose reference is determined by an external reference. These complements are always non-finite. Following principles 4.2.1 and 4.2.3 the non-finite complement clause introduced by *xcomp* relation shall receive by default a PRO subject (controlled or arbitrary).

The markers (conjunctions, prepositions or Wh-elements) at the beginning of the clause change the dependency relation to *prepc*, *rcmod*, *partmod* and *infmod* and a slight

---

[1]IP-Inflectional phrase corresponds roughly to the concept of clause (matrix or embedded)

variation in clause features and constituency. The only other relation that introduces a complement clause with PRO element is *prepc* and the only preposition is "whether". The other relations will be discussed latter in this chapter since they correspond to other types of empty elements.

I have explained earlier how to identify the place where PRO element shall be created. Before creating it we need to find out (1) whether PRO is arbitrary (equivalent to pronoun "one") or it is bound to another constituent. And if it is bound then decide (2) whether it is bound to subject or object (case in which we say that PRO is subject or object controlled).

Generalization 4.2.6 introduces a mood test for complement clause. Even is it is non-finite in GB theory it still may be declarative or interrogative. Halliday's MOOD system (Halliday & Matthiessen 2004: p. 107-167) does not allow mood selection for non-finite clauses. The reason declarative mood is mentioned is because GB theory the complement clauses can have declarative or interrogative mood. But as soon as we formulate an interrogative complement clause it can be only a Wh interrogative, thus the test is whether there is a Wh-marker (who, whom, why, when, how) or "whether" preposition. The presence of any marker like in example 62 at the beginning of the complement clause will change the dependency relation and the structure of the dependent clause. The only case when the complement clause remains non-finite and without a subject is the case of "whether" preposition and the clause is introduced via *prepc* relation. However if any such marker is missing then the clause is declarative and thus it must be controlled by a NP, so the arbitrary PRO is excluded. The cases of Wh-marked non-finite clauses will be treated in the further section 4.3.4 on Wh-element movement.

(62)   Albert asked [whether/how/when/ PRO to go.]

Based on the above I propose Generalization 4.3.1 enforcing obligatory control for *xcomp* clauses.

Generalization 4.3.1. If a clause is introduced by *xcomp* relation then it must have a PRO element which is bound to either subject or object of parent clause.

(63)   Albert$_i$ asked [PRO$_i$ to go alone.]

(64)   Albert$_i$ was asked [PRO$_i$ to go alone.]

(65)   Albert$_i$ asked Wendy$_j$ [PRO$_j$ to go alone.]

(66)   Albert$_i$ was asked by Wendy$_j$ [PRO$_i$ to go alone.]

The generalization 4.2.7 appeals to the *voice* feature of the parent clause in order to determine what NP is controlling the PRO element in the complement. Consider 63 and it's passive form 64. In both cases there is only one NP that can command PRO and it is the subject of the parent clause "Albert". So we can generalise that the voice does not play any role in controller selection in one argument clauses (i.e. clauses without a nominal complement). In 65 and 66 the parent clause takes two semantic arguments. Second part of principle 4.2.2 states that in case of obligatory control PRO must be *c-commanded* by an NP. In 65 both NPs("Albert" and 'Wendy") c-command PRO element, however according to Minimality Condition(Haegeman 1991: p. 479) "Albert" is excluded as the commander of PRO because there is a closer NP that c-commands PRO. In case of 66 the only NP that c-commands PRO is the subject "Albert" because "by Mary" is a PP (prepositional phrase) and also only NPs can control a PRO as stated in principle 4.2.6. In the process of passivation the complement becomes subject and the subject becomes a prepositional (PP-by) complement then the latter is automatically excluded from control candidates, thus supporting principle 4.2.7.

Generalization 4.3.2. The controller of PRO element in lower clause is the closest nominal constituent of the higher clause.

The adjunct non-finite clauses (examples 34 and 35) shall be treated exactly as the non-finite complement clauses. Generalization 4.2.8 emphasises obligatory control for them. The only difference between the adjunct and complement clauses is dictated by the verb of the higher clause whether it theta marks or not the lower clause. In dependency grammar the adjunct clauses are also introduced via *xcomp* and *prepc* relations, so syntactically there is not distinction between the two and patterns.

The *prepc* relation in dependency grammar introduces a prepositional clausal modifier for a verb (VN), noun (NN) or adjective (JJ). Adjective and noun modification are cases of copulative clauses. However such configuration are not relevant to the context of this work because the dependency graphs are normalised in the preprocessing phase described in the next chapter Section 6.2. Within the normalization process, among others, the copulas are removed and transformed to verb predicated clauses instead of adjective or noun predicated clauses.

The subject clauses are introduced via *csubj* relation. Subject non-finite clauses are quite different from complement and adjunct clauses because, according to generalization 4.2.9 the PRO is optionally controlled. Since it is not possible to bind PRO solely on syntactic grounds the generalization 4.2.9 proposes arbitrary interpretation.

4.3.2 NP-traces

Syntactically, NP raising can occur only when there is a complement clause by moving the subject of a lower clause into position of higher clause. The subject of higher clause c-commands the subject of lower clause. This is exactly the same syntactic configuration like in the case of PRO subjects. In the dependency grammar the lower clause is introduced via *xcomp* (explained in Section 4.2.1).

Poirot was believed to have destroyed the evidence
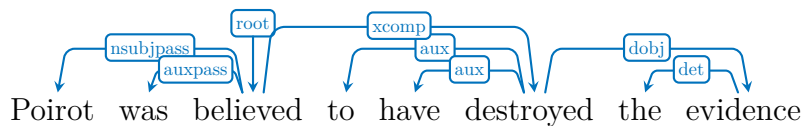
Fig. 4.1 Dependency parse for Example 41

The figures 4.1 and 4.2 represent a dependency parses for examples 41 and 43. To identify the place of the empty category is enough to apply the subject control pattern depicted in 7.3. However, just by doing so it is not possible to distinguish whether the empty subject is a PRO or a NP trace *t*.

Poirot seems to have destroyed the evidence
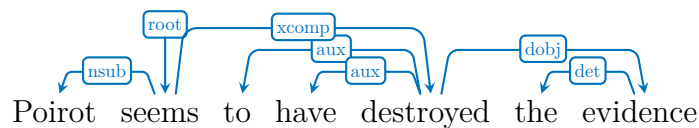
Fig. 4.2 Dependency parse for Example 43

When the empty subject in the embedded clause is detected and instantiated through the *obligatory subject control* pattern it is important to distinguish among the cases.

First of all this distinction is crucial for assignment of thematic roles to the constituents. So the problem is deciding on the type of relationship between the empty constituent and it's antecedent (subject of matrix clause) to which it is bound. In the case of *PRO* constituent, the thematic roles are assigned to both the empty constituent and to its antecedent locally in the clause they are located. So the PRO constituent receives a thematic label dictated by the verb of the embedded clause and the antecedent by the verb of the matrix clause. In the *t*-trace case the thematic role is assigned only to the empty constituent by the verb of the embedded clause and this role is propagated to its antecedent.

Making such distinction directly involves checking role distribution of upper and lower clause verb and deciding the type of relationship between the empty category and

its antecedent. If such a distinction shall be made at this stage of parsing or postponed to Transitivity analysis (i.e. semantic role labeling) is under discussion because each approach introduces different problem.

Before discussing each approach I would like to state a technical detail. When the empty constituent is being created it requires two important details: (a) the antecedent constituent it is bound to and (b) the type of relationship to it's antecedent constituent or if none is available the type of empty element: t-trace or PRO. Now identifying the antecedent is quite easy and can be provided at the creation time but since the empty element type may not always be available then it may have to be marked as partially defined.

The first solution is to create the empty subject constituents based only on syntactic criteria, ignoring for now element type (either PRO or *t*-trace) hence postponing it to semantic parsing phase. The advantage of doing so is a clear separation of syntactic and semantic analysis. The empty subject constituents are created in the places where they should be and it leaves aside the semantic concern of how the thematic roles are distributed. The disadvantage is leaving the created constituents incomplete or under-defined. Moreover the thematic role distribution must be done within the clause limits but because of raising, this process must be broadened to a larger scope beyond clause boundaries. Since transitivity analysis is done based on pattern matching, the patterns rise in complexity as the scope is extended to two or more clauses thus excluding excludes iteration over one clause at a time (which is desirable).

Otherwise, a second approach is to decide the element type before Transitivity analysis (semantic role labeling) and remove the burdened of complex patterns that go beyond the clause borders. Also, all syntactic decisions would be made before semantic analysis and the empty constituents would be created fully defined with the binder and their type but that means delegating semantically related decision to syntactic level (in a way peeking ahead in the process pipeline).

The solution adopted here is mix of the two avoiding two issues: (a) increasing the complexity of patterns for transitivity analysis, (b) leaving undecided which constituents accept thematic role in the clause and which don't.

The process to distinguish the empty constituent type starts by (a)identifying the antecedent and the empty element (through matching the subject control pattern in Figure 7.3), (b) identifying the main verbs of higher and lower clauses and correspondingly the set of possible configurations for each clause(by inquiry to process type database(PTDB) described in the transitivity analysis Section 7.2).

Generalization 4.3.3. To distinguish the *t*-traces check the following conditions:

- the subject control pattern matches the case AND

- the process type of the higher clause is two or three role cognition, perception or emotion process (considering constraints on Cognizant and Phenomenon roles). AND

- among the configurations of higher clause there is one with:

  - an expletive subject OR

  - the Phenomenon role in subject position OR

  - Cognizant in subject position AND the clause has passive voice or interrogative mood (cases of movement).

If conditions from generalization 4.3.3 are met then the empty constituent is a subject controlled *t*-trace. Now we need a set of simple rules to mark which constituents shall receive a thematic role. These rules are presented in the generalization 4.3.4 below.

Generalization 4.3.4. Constituents receiving thematic roles shall be marked with "thematic-role" label, those that do not receive a thematic role shall be marked with "non-thematic-role" and those that might receive thematic role with "unknown-thematic-role". So in each clause:

- the subject constituent is marked with "thematic-role" label unless (a) it is an expletive or (b) it is the antecedent of a *t*-trace then marked "non-thematic-role"

- the complement constituent that is an nominal group (NP) or an embedded complement clause is marked with "thematic-role" label.

- the complement that is a prepositional group (PP) is marked with "unknown-thematic-role".

- the complement that is a prepositional clause is marked with "unknown-thematic-role" label unless they are introduced via "that" and "whether" markers then it is marked with "thematic-role" label.

- the adjunct constituents are marked with "non-thematic-role"

4.3.3   Wh-trances

Let's turn now to how Wh-movement and relative clauses are represented and behave in dependency grammar and SF grammars. Figures 4.3, 4.4 present dependency parses for Wh-movement from subject and object positions of lower clause while 4.5 from adjunct position.
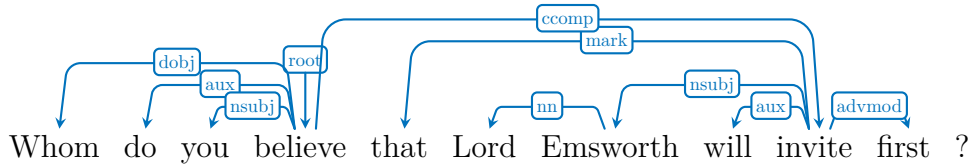
Fig. 4.3 Dependency parse for Example 52
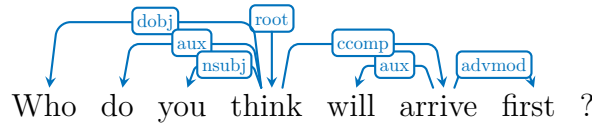
Fig. 4.4 Dependency parse for Example 51

Just like in cases of NP-movement, the Wh-groups move only into two and three role cognition, perception and emotion figures. In contrast, if the NP-antecedents land in expletive or passive subject position then the Wh-antecedents land in subject or subject preceding position functioning as subject, complement or adjunct functions depending on the Wh-Element.

The essential features for capturing the Wh-movement in dependency graphs are (a) the finite complement clause identified by *ccomp* relation between the matrix and embedded clause (b) the Wh-element/group plays a complement function in higher clause which is identifiable by *dobj*, *prep* or *advmod* relations to the main verb (c) the function of the Wh-trace in lower clause is either Subject(e.g. 52), Object(e.g. 51) or Adjunct.

ccomp relation is defined in Marneffe & Manning (2008a) to introduce a complement clauses with an internal subject which are usually finite. I must emphasize the fact that the lower clause must be embedded into the higher one and receive a thematic role in higher clause.

Regardless whether the syntactic function of the traces in the lower clause is Subject or Object, in the higher clause the Wh-group takes Object function and is bound to the main verb via *dobj* relation but is positioned before the main verb and the Subject

(a structure corresponding to Wh-interrogatives). Wh-group can take also Subject function in the higher clause, but then it is not a case of Wh-movement and is irrelevant for us at this point because there is no empty element, i.e. Wh-trace. The attribution of clause function to the Wh-trace is based on either the case of the Wh-group or the missing functional constituent in the lower clause.
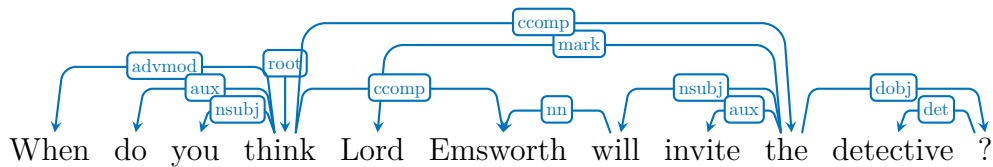
Fig. 4.5 Example dependecy parse with Adjunct Wh-element

In case of Wh-traces with Adjunct function in the lower clause, like in figure 4.5, their antecedents also receive adjunct function in the higher clause. Adjunct Wh-group cannot bind to the clause it resides in if the clause has (generic) present simple tense and thus it has to be antecedent for a trace in lower clause which has other tense or modality than present simple. The reader can experiemnt with changing tense in the example 4.5.

(67)   Who$_i$ believes that Lord Emsworth will invite a detective?

(68)   To whom$_i$ did Poirot say t$_i$ that Lord Emsworth will invite a detective?

Not always the Wh-groups are movements from lower clause. It is possible that the trace of the moved element to reside in the higher clause (complement) or even have a case of no movement when Wh-element takes the subject function in the higher clause. 67 and 68 are good examples of a *short* movement (in clause movement). However the short movement in dependency grammar has no relevance because the dependency grammar is order free and the functions are already assigned accordingly so short movement is not a subject of interest for the current chapter.

4.3.4   Chaining of Wh-trances
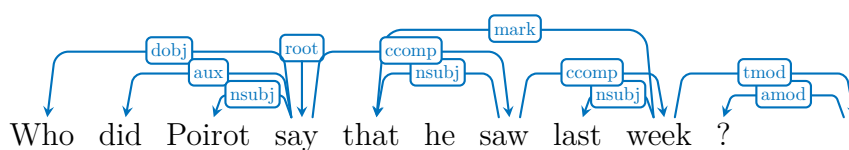
Fig. 4.6 Dependency parse for Example 61

Recall the *cyclic* and *successive* properties of Wh-movement from previous section underlined by example 61 and its dependecy parse in figure 4.6. GBT suggests that the Wh-movement leaves traces in all the intermediary clauses. In dependency grammar these properties shall be treated instrumentally for determining intermediary hops in search for the foot of the chain and none of the intermediary traces shall be created. There simply is no further purpose for them as they do not receive a thematic role in the intermediary clauses.

4.3.5 Wh-trances in relative clauses

In GB theory the Wh-elements that form relative clauses (*who, whom, which, whose*) are considered moved. In dependency grammar such movement is redundant since the Wh-element and its trace are collapsed and take the same place and function. The Wh-elements function either as subject or complement. When the relative clause is introduced by a Wh-group there is no empty element to be detected, rather there is anaphoric indexing relation to a noun it refers to. Focusing now on the relative clauses, there are three more possible constructions that introduce them: (a) a prepositional group that contains a Wh-element, (b) "that" complementizer which behaves like a relative pronoun (c) the Zero Wh-element which is an empty element and which functions the same way as a overt Wh-element. The table 4.4 lists possible elements that introduce a relative clause, their features and the functions they can take.

Next I discuss how to identify, create and bind the traces of Wh-elements to their antecedents.

Relativizing element	Feature	(Clause) Function	Examples
who	person	subject	... the woman who lives next door.
whom	person	complement (non defining clause)	... the doctor whom I have seen today.
which	non-person	subject/ complement	... the apple which is lying on the table. ... the apple which George put on the table.
whose	possessive, person	possessor in subject	... the boy whose mother is in a nurse.
(*any of the above in prepositional group*)	person/ non-person	thing/possessor in subject	... the boy to whom I gave an advice. ... the cause for which we fight.
that	person/ non-person	subject	... the apple that lies on the table.
Zero Wh-element	person/ non-person	subject	... the sword sent by gods.

Table 4.4 The Wh-elements introducing a relative clause.

Compare the dependency parse with an overt Wh-element in Figure 4.7 and the covert one in 4.8.
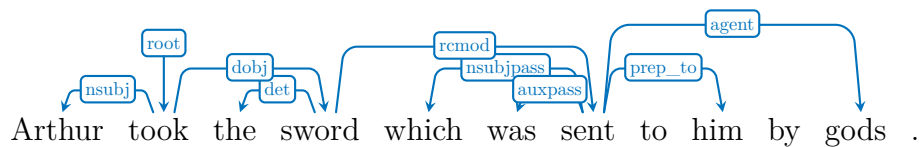
Fig. 4.7 Dependency parse for "Arthur took the sword which was sent to him by gods."
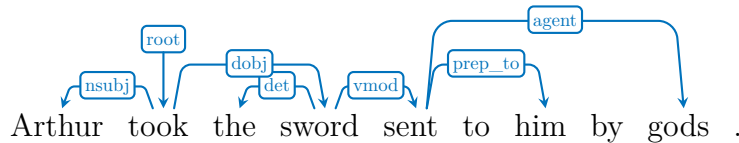
Fig. 4.8 Dependency parse for "Arthur took the sword sent to him by gods."

Relative clauses in dependency graphs are introduced by *rcmod*, *partmod*, *infmod* and *vmod* relations. The *rcmod* introduces relative clauses containing a Wh-group while the *partmod* and *infmod* introduce finite and non-finite relative clauses with Zero Wh-element. After the Version 3.3 of Stanford parser the *partmod* and *infmod* relations have been merged into *vmod*. So the dependency relation is the main signaller if empty Wh-elements even though they are subject to corrections in preprocessing (Section 6.2) phase to ensure a uniform treatment of each relation type.

The Zero Wh-element behaves exactly like the *PRO element* in the case of non finite complement clauses discussed in Section 4.2.1. It receives thematic roles in both the higher clause and in lower clause and is not a part of a chain like the cases of NP/Wh-movement.

4.4   Discussion

This chapter treats the identification of the *null elements* in syntactic structures. Section 4.2 presents how GBT theory handles null elements and then Section 4.3 shows how the same principles translate into Stanford dependency graphs.

Identification of null elements is important for the semantic role labeling process described in Chapter 7 because usually the missing elements are participant roles (theta roles) shaping the semantic configuration. The semantic configurations (gathered in a database) are matched against the syntactic structure and missing elements lead to failing (false negatives) or erroneous matches (false positives). Therefore to increase the accuracy of semantic role labeling spotting null elements is a prerequisite.

This Chapter also contributes to establishing cross theoretical connections that is among current thesis objectives. Specifically it provides translations of necessary principles and generalizations from GB theory into the context of dependency grammar. These results are directly used in Section 7.1 for generating graph patterns.

Chapter 5

On Graphs, Feature Structures and Systemic Networks

The parsing algorithm 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 algorithm. It 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 and operational requirements arose. These requirements are as follows:

- 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) and 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 (needed for polysemy and uncertainty)

- conjunctive descriptions (needed for multiple feature selections in recursive systems)

- (conditional) pattern specifications (i.e. define patterns of graphs)

- operational pattern specifications (i.e. a functional description to be executed in pattern graphs)

5.1 Basic Definitions

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

At this point you may ask why graphs and not trees since in the field of computational linguistics trees has been taken as the de facto data representation for parse structures. Firstly, the trees are a special kind of graphs and anything expressed as a graph could as well be expressed as 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 this chapter. This expressiveness is needed when dealing with interconnection of various linguistic theories.

Definition 5.1.1 (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 5.1.2 (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 $u, v \in V$)

Carl Pollard & Ivan Sag (1987) have formally described the useful concepts for grammatical representations of HPSG. Taking on board and extending the typed feature structure theory and extending it in several useful forms. Including definitions of hierarchy, feature structure recursion, logical evaluation and compositions and unification the key operation parsing feature structured grammars. In this thesis however I simplified feature structures to only a few concepts. This si due to mixing concept of *graph* as structure carriers with the concept of *feature structure* as mainly description carrier. This mix does not exist in (Carl Pollard & Ivan Sag 1987) who use feature structure as both structure and description carriers. Nevertheless, besides a few instrumental concepts, I proceed defining feature structure as follows:

Definition 5.1.3 (Feature Structure (FS))**.** A *feature structure* F is a finite set of attribute-value pairs $f_i \in F$. A *feature* $f_i = (a, v)$ is a mapping between an identifier a

(a symbol) and a value v which is either a symbol, an ordered set or another feature structure. The function $att(f_i)$ is a function returning the feature identifier $att(f_i) = a$ and the function $val(f_i)$ is a function returning the ascribed value of a feature $(f_i) = v$

Definition 5.1.4 (Ordered Set). An *ordered set* $S = \{o_1, o_2, ..., o_n\}$ is a finite well defined collection of distinct objects o_i arranged in a sequence such that $\forall o_{i-1}, o_i \in S : o_{i-1} < oi$

Definition 5.1.5 (Conjunction Set). A *Conjunction Set* $S_{conj} = (S, conj)$ is an ordered 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 5.1.6 (AND-Set). $AND - Set$ (also called *conjunctive 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 5.1.7 (NAND-Set). $AND - Set$ (also called *negative conjunctive 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 5.1.8 (OR-Set). $OR - Set$ (also called *disjunctive 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 5.1.9 (XOR-Set). $OR - Set$ (also called *exclusive disjunctive set*) is a conjunction set $S_{XOR} = \{a, b, c...\}$ that is interpreted as a logical exclusive disjunction of its elements $a \oplus b \oplus c \oplus ...$ 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 type of 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.

I use τ function defined $\tau : S \to \{S_{AND}, S_{OR}, S_{XOR}, S_{NAND}\}$ to return the type of the conjunction set and *size* function defined $size : S \to \mathbb{N}$ to return the number of elements in the set.

Definition 5.1.10 (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, when I refer to a graph I will a feature rich digraph, unless otherwise stated. The parsing algorithm operates with three feature graph types: *Dependency Graphs* (DG) (example figure 5.1), *Constituency Graphs* (CG) also called Mood Constituency Graphs (MCG) (example figure 5.2) and Pattern Graphs (PG) which can also be viewed as *Query Graphs* (QG).

Definition 5.1.11 (Dependency Graph)**.** The *dependency graph* is a feature rich digraph whose nodes V correspond to words, morphemes or punctuation marks in the text and carry but not limited to the following features: *word, lemma*, part of speech (*pos*) and when appropriate the named entity type (net); while the edges E describe the dependency relation (*rel*).
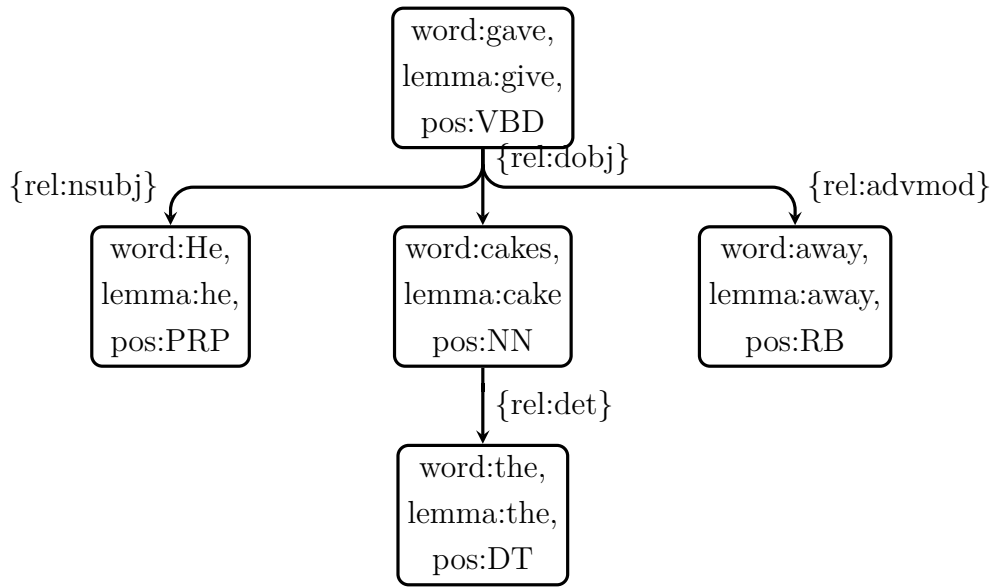
Fig. 5.1 Dependency graph example with FS nodes and edges

Definition 5.1.12 (Constituency Graph)**.** The *constituency graph* is a feature rich digraph whose nodes V correspond to SFL *units* and carry but not limited to the unit class and the function within the parent unit (except for the root node); while the edges E carry mainly the constituency relation between parent and daughter nodes but also potentially other relation types.

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 (see theoretical aspects

of SFL in Chapter 2). The root node (usually a clause) is an exception and it does not act as a functional element because it doesn't have a parent unit. The leaf nodes carry the same features as the DG nodes plus the class feature which correspond to the traditional grammar part of speech.

Apart from the essential features of class and function, the CG nodes carry additional class specific features selected from the relevant system network. For example gender, number and animacy are specific to nominal unit classes such as nouns and nominal group; while tense and modality are specific mainly to clause class.
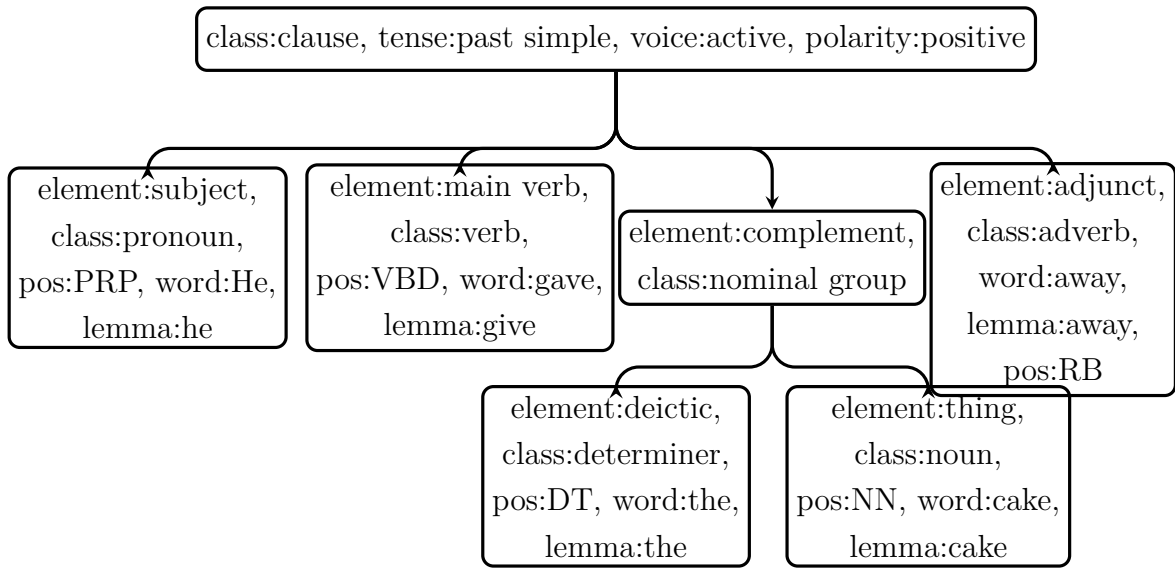
Fig. 5.2 Constituency graph example

Regardless of the graph type, constituency or dependency, it is essential to express patters over those graphs in order to facilitate various operations. The PG (defined in 5.1.13) are special kinds of graphs meant to represent small (repeatable) parts of parse graphs that stand for a certain grammatical feature. The patterning is described as both graph structure and feature presence (or absence) in node or edge. Such pattern graphs can be viewed as the conditional part of a query language if not the entire query statements (i.e. select, insert, update, delete operation descriptions). This kinds of graphs need last four requirements listed in the beginning of this section.

Definition 5.1.13 (Pattern Graph)**.** A *pattern graph* (PG) describes regularities in node-edge configuration and feature structure including descriptions of *negated nodes or edges* (i.e. absence of), logical operators over feature sets (AND, OR, XOR and NAND) and operations once the pattern is identified in a target graph (select, insert, delete and update).

The feature structure of a PG are always *underspecified* as compared to the dependency or constituency graph in the sense that irrelevant attribute-value pairs are omitted sometimes down to an empty FS. However often it is useful to specify more than one value for the same feature as a list of disjunctive values allowing the pattern to cover larger set of possible cases. I will call these FS as being *over-specified.*

An example of PG is depicted in figure 5.6 in the Section 5.6. It deals with pattern graph matching and other operations with in detail. But before that I will first briefly cover generic operations on graphs and the problem of graph matching also known in computer science as the *graph isomorphism* problem.

5.2   More on Pattern Graphs

As already mentioned in the previous section we are dealing with a special case of graph isomorphism precisely because the graphs we consider are feature rich graphs. Specifically, besides the graph structure, the node and edge identity checking is a secondary check to consider.

Before I dive into pattern graph matching and operations with pattern graphs I will first discuss two example of pattern graphs. Consider the case of present perfect continuous tense which traditional grammar defines as in Table 5.1 regardless of the element order.

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

Table 5.1 Present perfect continuous pattern

Examples 69-71 show variations of this tense in a simple clause according to the mood and "has" contraction. Of course there are also voice variations but I skipped them in this example because it adds combinatorially to the number examples. The Figures 5.3-5.5 represent dependency parses for the above examples.

(69)   He has been reading a text.

(70)   He's been reading a text.

(71)   Has he been reading a text?
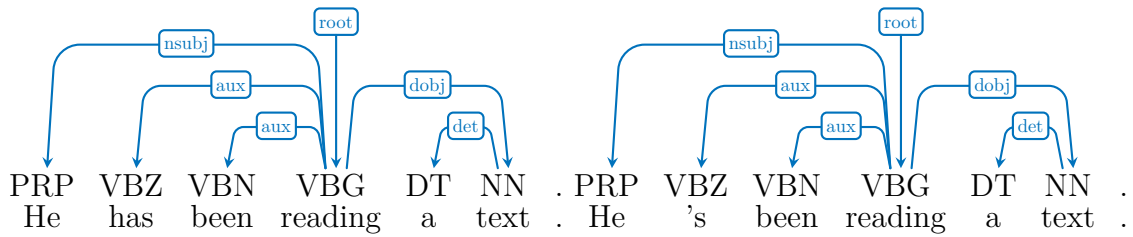
Fig. 5.4 Present perfect continuous: indicative mood, contracted "has"
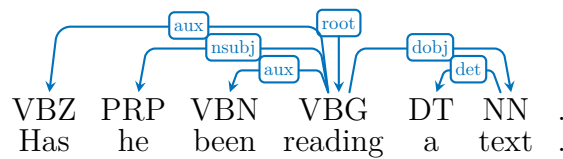
Fig. 5.5 Present perfect continuous: interrogative mood, un-contracted "has"

The present perfect continuous tense can be formulated as a pattern graph (including voice) over the dependency structure as illustrated in Figure 5.6 below.
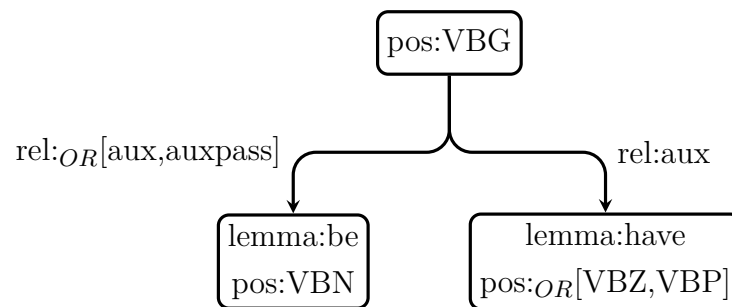
Fig. 5.6 The graph pattern capturing features of the present perfect continuous tense

In this a pattern the main lexical verb is *present participle* indicated via *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 person. Also the *to be* can be either connected by *aux* relation or in case of passive form by *auxpass* relation. Note that the pattern in Figure 5.6 over-specifies the edge type (using the OR set notation) 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*.

One of the fundamental features of language is its sequentiality and directionality. Place and order of constituent elements play an important role in SFG (see Section 2.2.1). Unfortunately capturing the aspect of order is not straight forward in graphs

which inherently lack the capacity to capture linear order specifically. In the simplest form, they just describe connections between nodes and are agnostic to any meaning or interpretation.

To demonstrate how the order is specified in the graph patterns, lets turn now to the clause mood and capture specifically the distinction between declarative and Yes/No interrogative moods. In SFG this features is described in terms of relative order of clause elements. If the finite is before the subject then mood is Yes/No-interrogative whereas when the finite succeeds subject then mood is declarative. The example 71 clearly contrasts in mood with 69 and 70.

Order can be specified in absolute or relative terns, partially or exhaustively. To overcome this problem I introduce three special features that cove all cases: the node *id*, *precede* and *position*. Node id take a token to uniquely identifies a node, the precede feature takes ordered sets to indicate the (partial) precedence of node ids, and the position feature indicates the absolute position of a node.

One way to introduce order among nodes is marking them with an absolute position. The parse graphs i.e. DGs and MCGs are automatically assigned at the creation time the absolute position of the node in the sentence text via the feature *position*. Only the leaf nodes can have a position assigned. The leaf nodes position corresponds to the order number in which they occur in the sentence text while the non-leaf node's position is calculated to the lowest of it's constituent nodes. The absolute position description rarely is used in the PGs, mainly for stating that a constituent is the first or the last one in the sentence.

Another way to specify node order is through relative positions for which the node id and the precedence features are introduced.

Continuing the example of mood features Figures 5.7 and 5.8 illustrate the use of relative and absolute node ordering constrains for declarative mood. In PGs, the relative order is preferred to absolute one but either is usable. Figure 5.9 depicts the PG for the Yes/No interrogative mood using the relative node ordering.
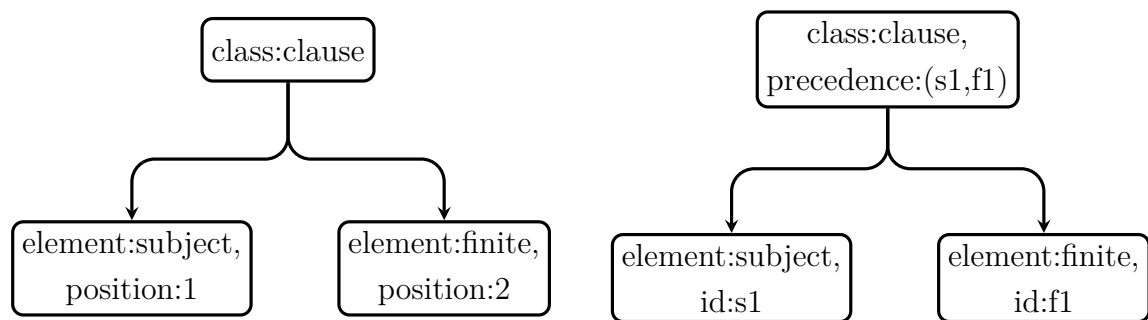
Fig. 5.8 Declarative mood pattern graph
with absolute element order
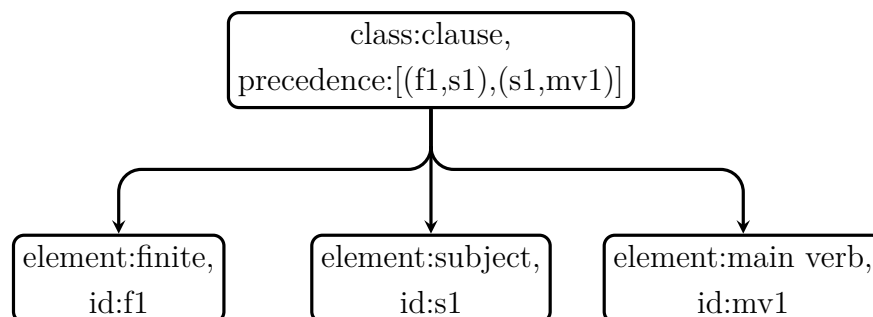
Fig. 5.9 Pattern graph for Yes/No-interrogative mood with a redundant main verb node

From the usability point of view there are few technicalities I shall emphasize. First, the precedence feature can be used on any node. When matched, the precedence declarations are collected from all nodes into a single set before being checked. However I recommend as a good practice to specify the order of nodes on the parent constituent.

Second, the notation in Figure 5.9 follows the Python bracketing meaning i.e. the round brakes signify tuples while the square ones lists (ordered sets). So the main verb element is redundant and is introduced to demonstrate multiple order specifications. However the order can be either specified as a set of binary tuples or as an ordered set (i.e. a Python list). So precedence:[(f1,s1),(s1,mv1)] is equivalent to precedence:[f1,s1,mv1].

Thirdly the ordering can be defined in absolute terms via position or in relative terms. Note that in the case of PGs the absolute ordering of nodes is interpreted relatively so PG in figure 5.7 is identical to 5.8.

Patterns like the ones explained above can be created for many other grammatical features and tested via graph pattern matching operation whether the feature is found in the parse graph or not. Once the pattern is identified it can act as a triggering condition to various operation affecting the parse graph or other external structures

which I describe in the next sections. For example once the pattern 5.6 is identified then the clause can be marked with the *tense* feature or in the case of dependency structure clause corresponding to the dominant verb node.

5.3 Graph Operations

Definition 5.3.1 (Atomic Query)**.** *Querying* $q_V(F, G)$ or $q_E(F, G)$ over the nodes V or edges E of a graph G is an operation that returns a set of nodes or edges filtered by the conditional feature structure F.

For example, for a given dependency graph, to select all the determiners we query the nodes with condition that part of speech has DT value i.e. *pos=NP* or to select all the edges connection a noun to it's determiner then the query is formulated for all edges whose relation type is *rel=det*.

The graph traversal defined in 5.3.2 is another important operation. For example DG traversal is used in bootstrapping the CG as a parallel structure. Another usage is conditional sub-graph selection of a given DG or CG. For example in the semantic enrichment phase (see Section 6.5), to ensure that the semantic patterns are applied iteratively to each clause, from a multi-clause MCG graphs are selected each individual clause sub-graphs without including the embedded (dependent) clauses. Using sub-graphs for performing the pattern matching, like in the case of semantic enrichment, decreases drastically the complexity of the graph isomorphism problem (described in Section 5.4) thus increasing the overall performance.

Definition 5.3.2 (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 the neighbouring each other in either *breadth first* (t_{BF}) or *width first* (t_{WF}) orders.

Definition 5.3.3 (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.

The graph traversal can be used in various ways, either for searching a node, an edge or finding a sub-graph that fulfils certain conditions on its nodes and edges if it is a conditional traversal. A traversal can be also used to execute generative operations on parallel data structure.

Definition 5.3.4 (Generative Traversal). *Generative traversal $m(M, G)$ of a graph G via a operation matrix M is an operation resulting in creation of another graph H by contextually applying generative operations.* The operation matrix M is a set of tuples (ctx, o, p) that link the visited node context ctx (as features of the node, the edge and previously visited neighbour) to a certain operation o that shall be executed on the target graph H with parameters p.

Now that generative traversal is defined, you may point out that similarly (or 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.

But such traversal operations cannot be easily defined. While traversal context may be sufficient for generative operations on a new structure, it is insufficient for executing affecting operations on the traversed graph. To overcome this limitation, instead of using traversal context I take a different approach: the *pattern graphs*, defined in previous section combined with generic graph matching algorithm. This mechanism offers a similar algorithmic independence of mapping structural context to operation(s) triggered by it.

5.4   Graph Matching

The graph matching problem is known in computer science as *graph isomorphism problem* which is an NP-complete problem. A peculiar characteristic of such problems is that given a solution then it can be very quickly *verified* in polynomial time but the time required to find a solution increases exponentially with the size of the problem. Therefore to prove whether or not such problems can be solved quickly is one of the main unsolved problems in computer science and the performance of of algorithms solving NP-complete problems is an important issue and requires careful investigation.

Definition 5.4.1 (Graph Matching). *For two given graphs G and H where $H \leq G$, graph matching (or graph isomorphism) is an operation of finding an sub-graph $G_1 \subseteq G$ that is structurally isomorphic to H.*

To the moment no algorithm exist to solve the graph isomorphism problem in polynomial time, however the latest available algorithms such as VF2 Cordella et al. (2001, 2004) or QuickSI Shang et al. (2008) performs the task quickly when the addressed graphs are of limited size.

Next I present some estimate calculations and compare to benchmarking study of Lee et al. (2013).

The graphs used in benchmarking tests described by (Lee et al. 2013) on AIDS dataset are composed of 2–215 nodes and 1–217 edges on which VF2 algorithm performs the isomorphism problem on average in 20–25 milliseconds for sub-graphs sizing between 4–24 edges. The NASA dataset (used in the same benchmarking study) which contains 36790 graphs sizing between 2–889 nodes and 1–888 edges the VF2 algorithm performs on average in 250 milliseconds for sub-graphs of 4 edges.

To put it into the context we have to answer the questions: how big the sentence graphs are, what size are the patterns and what would be a rule of thumb estimation of performance?

According to (Koeva et al. 2012), on average, an English sentence is composed of 12-20 words(n) with about 1.6 clauses per sentence . The parse graph of an average English sentence is a tree or very close to a tree whose number of nodes is within the limits between $n+1$ and $2n-1$ for a n number of leaf nodes (in our case the words). So for a sentence of 20 words the parse tree would be maximum 39 nodes.

Lets assume the size of a sentence is ten times the average i.e. 200 words and a maximum estimate of 399 nodes in the parse tree. The patterns used in current work are 1–5 edges. Overall in the parsing algorithm, the graph matching is mostly applied at the clause level which on average in English is of 6 8 words yields an average maximum of 15 nodes per parse graph which is 0.38 of the average sentence and 0.01 of the unusually big sentence. As used in the current implementation and given relatively small graphs, the performance VF2 algorithm fits well within reasonable time limits.

5.5   Rich Graph Matching

In order to accommodate feature rich graphs (FRG), VF2 algorithm is extended to perform custom identity checks. In the original implementation two node V_1 and V_2 are said to be equal if the nodes are of simple data types (e.g. integer or string) and they carry the same value. In our case, feature structures are attached to edges and stand for graph nodes. And there are cases when two nodes, even if the have somehow different structures, to be considered the same.

Therefore identity of complex structures becomes an elastic concept. Of course strict identify checking function is important, but we can derive more power from a nuanced check instead of a strict identity.

The functions that decide the identity of two objects (i.e. which is to say that they are the same) are called *morphisms*.

Definition 5.5.1 (Morphism)**.** A morphism (also called *identity function*) $f : X \rightarrow Y$ is a structure preserving map from one object X to the other Y where the objects are complex structures such as sets, feature structures or graphs.

Definition 5.5.2 (Identity Morphism)**.** for every object X, there exists a morphism $id_X : X \rightarrow X$, called *identity morphism* on X, such that for every morphism $f : A \rightarrow B$ we have $id_B \circ f = f = f \circ id_A$

In this work, for *identity morphism* I use interchangeably the terms *identity checking function* or *identity function.*

Definition 5.5.3 (Isomorphism)**.** The morphism $f : X \rightarrow Y$ is called *isomorphism* if there exists a morphism $g : Y \rightarrow X$ such that $f \circ g = id_X$ and $g \circ f = id_X$

I already have defined (somehow ahead) what is graph isomorphism; and that the graph matching should always be an isomorphic function. However the nodes and edges shall be rather loosely identical or they shall only be considered identical but not necessarily be so. Therefore the node and edge morphism functions shall rather be *asymmetric* or simply a morphisms without any assumptions of exact identity. Now I can define the rich graph matching as follows.

Definition 5.5.4 (Rich Graph Matching)**.** For two given graphs G and H where $H \leq G$ and two *morphism functions* f_V and f_E, the *rich graph matching* is the function that finds a structural isomorphism between H and $G_1 \subseteq G$ provided that for all nodes $V_i \in H$ their *morphism function* $V_j \in G_1$ satisfies the identity function $f_V(V_i) = V_j$

So, the functions that compare whether two node or edge are the same in fact compare their feature structures and in fact the sameness is defined in accordance with the goals of the particular task. That's why identity checking function is provided as a parameter to the rich graph matching algorithm.

How is the node and edge identity checking done (or how are the morphism functions defined) is covered in the next section. What is important to mention here is the complexity of such nuanced checks since we have discussed the complexity of VF2 algorithm only on graphs where the edges and nodes are simple data structures.

The comparison of two FS is a PTIME problem that is efficiently solvable in polynomial time. Of course, this (slightly) increase the complexity of the matching process as a whole but still this lies well within the limits of practical computability.

The current implementation of VF2 algorithm includes the custom morphism functions for nodes and edges. So far I have not encountered performance issues with it. For the future however it would be of tremendous value to know exactly how much is the original VF2 algorithm burdened by such extra checks and what are the reasonable upper limits for the node size (i.e the complexity of the node feature structure). And perform some stress testing for the whole enterprise.

5.6   Pattern Graph Matching

An extension and particular case of rich graph matching is the *pattern graph matching* where H (Definition 5.5.4) is a pattern graph (Definition 5.1.13) and the identity checking function(s) are not strict but permissive to feature over-specification of node and edge feature structures.

I will define now how the identity checking function (identity morphism) used for pattern graph matching. It operates on feature structure values, which can be either atomic types *simple* of one of the conjunctive sets: S_{AND}, S_{OR}, S_{XOR} and S_{NAND}. I use both set theoretic notations for inclusion $\subseteq$, intersection $\cup$ and element belonging to a set $\in$ and the logical notations for conjunction $\wedge$ and tautology $\top$. The unary function $T(x)$ returns the type of x element.

When checking the identity of two feature values, three cases can be asserted: $x = y$ i.e. x is definitely equal to y, $x \neq y$ i.e. x is definitely different from y and $x \sim y$ i.e. x is maybe (or could be) equal to y.

I define below two identity morphisms: (a) *permissive* $I_{permissive}$ (defined by Equation 5.2) which includes the uncertain cases and (b) *strict* I_{strict} (defined by Equation 5.1) which excludes the uncertain cases. The main difference between the two morphism functions is whether on the right side (the instance graph) any uncertainty is accepted. This is to say any of the disjunctive sets s_{OR} and S_{XOR}.

Definition 5.6.1 (Strict Pattern Graph Matching). *Strict pattern graph matching* is a rich graph matching where a morphism for the pattern graph H is found in the target graph G given that $H \leq G$ and that for any node $p \in H$ there is a node $r \in G$ satisfying the strict identity morphism $I_{strict} : p \to r$

$$I_{strict} : p \to r \models \begin{cases} p = r, & \text{if } T(p) = simple \land T(r) = simple \\ p \in r, & \text{if } T(p) = simple \land T(r) = S_{AND} \\ p \subseteq r, & \text{if } T(p) = S_{AND} \land T(r) = S_{AND} \\ p \cap r \neq \varnothing, & \text{if } T(p) = S_{OR} \land T(r) = S_{AND} \\ r \in p, & \text{if } T(p) = S_{OR} \land T(r) = simple \\ r \in p, & \text{if } T(p) = S_{XOR} \land T(r) = simple \\ p \cap r = \varnothing, & \text{if } T(p) = S_{NAND} \land T(r) \in \{S_{AND}, S_{OR}, S_{XOR}\} \\ r \notin p, & \text{if } T(p) = S_{NAND} \land T(r) = simple \\ \top, & \text{if } T(p) = S_{NAND} \land T(r) = S_{NAND} \end{cases} \quad (5.1)$$

Definition 5.6.2 (Permissive Pattern Graph Matching). *Permissive pattern graph matching* is a rich graph matching where a morphism for the pattern graph H is found in the target graph G given that $H \leq G$ and that for any node $p \in H$ there is a node $r \in G$ satisfying the permissive identity morphism $I_{permissive} : p \to r$

$$I_{permissive} : p \to r \models \begin{cases} p = r, & \text{if } T(p) = simple \land T(r) = simple \\ p \in r, & \text{if } T(p) = simple \land T(r) \in \{S_{AND}, S_{OR}, S_{XOR}\} \\ p \subseteq r, & \text{if } T(p) = S_{AND} \land T(r) = S_{AND} \\ p \cap r \neq \varnothing, & \text{if } T(p) = S_{OR} \land T(r) \in \{S_{AND}, S_{OR}, S_{XOR}\} \\ r \in p, & \text{if } T(p) = S_{OR} \land T(r) = simple \\ p \cap r \neq \varnothing, & \text{if } T(p) = S_{XOR} \land T(r) \in \{S_{OR}, S_{XOR}\} \\ r \in p, & \text{if } T(p) = S_{XOR} \land T(r) = simple \\ p \cap r = \varnothing, & \text{if } T(p) = S_{NAND} \land T(r) \in \{S_{AND}, S_{OR}, S_{XOR}\} \\ r \notin p, & \text{if } T(p) = S_{NAND} \land T(r) = simple \\ \top, & \text{if } T(p) = S_{NAND} \land T(r) = S_{NAND} \end{cases}$$
$$(5.2)$$

Of course these two are not the only identity morphisms that can be defined for the pattern matching. The implementation accepts any binary function that returns a truth value meaning that the two arguments shall be considered the same or not. Moreover the identity function can be provided for edges as well, but I skip it in the

current thesis because I do not use. In the future it would be useful to define the edge morphism functions and identify the use cases that employ them.

Now that I have defined how patterns are identified in the graphs, lets take a look at more advanced applications of it. In the next section I explain how the graph isomorphism can be enacted once they are identified.

5.7 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. So 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 *affordance patterns* (Definition 5.7.1). Next I explain how to embed the operations into the graph pattern and how they are used in the algorithm.

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 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. So far the implemented operations are *insert*, *delete* and *update*. But anyone that finds appropriate can extend it with any other operations that may be useful.

Definition 5.7.1 (Affordance Graph Pattern)**.** An *affordance graph pattern* is a graph pattern that, at least one node or edge, has *operation* and *arg* features.

I say that the affordance patterns are enacted once they are tested for isomorphism in another graph and if one is found then all the defined operations are executed accordingly.

Definition 5.7.2 (Affordance Graph Enacting)**.** For an affordance graph H and a target graph G, *affordance graph enacting* is a two step operation that first performs the permissive or strict pattern graph matching and if any isomorphism graph $G_1 \subseteq G$ is identified and second for every node $p \in H$ with an operation features, executes that operation on the corresponding node $r \in G_1$ of the isomorphism.

5.7.1   Pattern-Based Node Selection

It is often needed to select nodes from a graph that have certain properties and are placed in a particular configuration. This operation is very similar to the *atomic graph query* defined in 5.3.1. The main difference is ability to specify that the node is a part of the a certain structural configuration which is not possible via the atomic query.

For example let's say that we are interested in all nodes in a dependency graph that can take semantic roles specifically subjects, and complements of the clause. For the sake of simplicity example I exclude prepositional phrases and embedded clauses that sometimes can also take semantic roles. The pattern identifying such nodes looks like the one in Figure 5.10. It selects all the nodes that are connected via *nsubj, nsubjpass, iobj, dobj* and *agent* edges to a VB node.
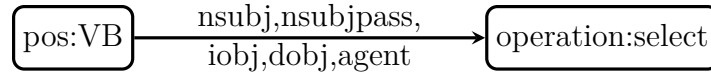
Fig. 5.10 Graph pattern that selects all the nodes that can receive semantic roles

5.7.2   Pattern-Based Node (FS) Update

There are other cases when the FS of the nodes needs to be updated either by adding or altering a feature value. This can be achieved via *pattern-based update* operation. For example, consider the example analysis 5.2 and the task to assign *Agent* feature to the subject node and *Possessed* feature to the complement. PG depicted in figure 5.11 fulfils exactly this purpose.
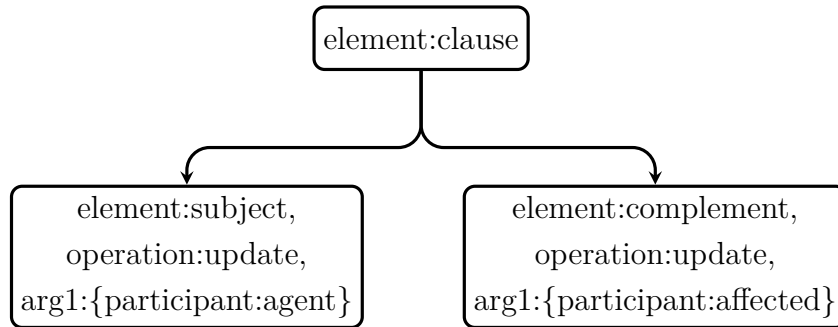
Fig. 5.11 Graph pattern for inserting the agent and affected participant role features to subject and direct object nodes.

Consider the very same pattern, but applied to a sentence in the Table 5.3. The clause has two complements and they are by no means distinguished in the pattern

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

Table 5.2 MCG with a transitive verb

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 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 5.12 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 5.3 MCG with a di-transitive verb
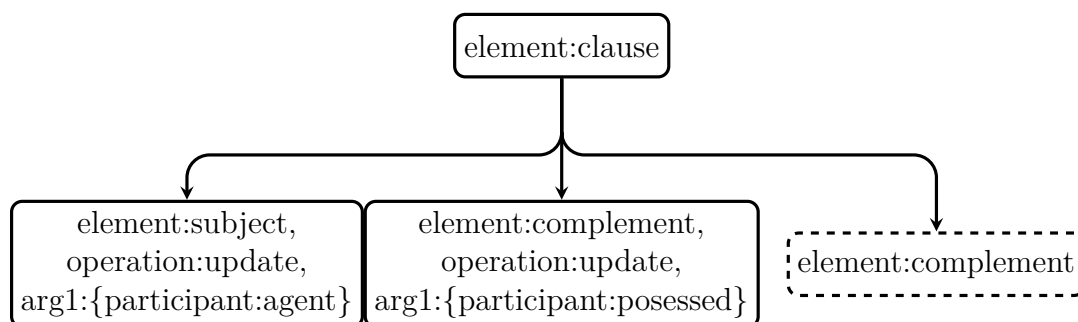
Fig. 5.12 PG for inserting agent and possessed participant roles to subject and complement nodes only if there is no second complement.

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.

5.7.3   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 4.

(72)   Albert asked [∅ to go alone].

Consider the Example 72. 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 4.2. The table 5.4 provides a constituency analysis for the example and the null elements (in italic) are appended for the explicit grammatical account. In the Section 4.3 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).

class:clause					
element: subject	element: main verb	element: complement, class:clause			
		element: subject	element: main verb		element: adjunct
Albert	asked	*Albert*	to	go	alone.

Table 5.4 The constituency analysis that takes null elements into consideration
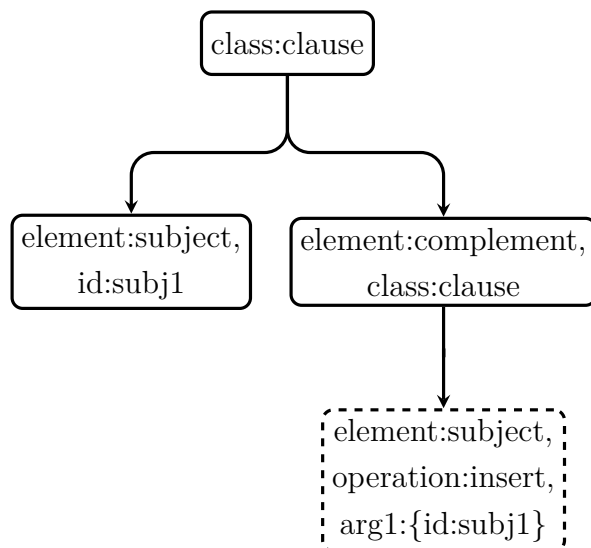
Fig. 5.13 A graph pattern to insert a reference node

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.

5.8   Systems and Systemic Networks

In the Section 2.2.4 I present the basic definition of System and System Network as it is formulated in the SF theory of grammar. The that definition in te context of theory of grammar is shaped into a more practical definition closer to what may be operationalized and represented with computer systems. In addition I cover few more useful concepts for implementation of system networks applied to enrichment of constituents with systemic features. The graphical notations introduced by Halliday &

Matthiessen (2013) are useful in reading and writing system networks in this thesis. Below is a system network with a simple entry condition (Figure 5.14), a system network grouping that share the same entry condition (Figure 5.15), a system network with a disjunctive and conjunctive entry conditions (Figure 5.16 and 5.17).

$$a \rightarrow \left[\begin{matrix} x \\ y \end{matrix} \right.$$
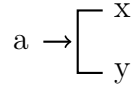
Fig. 5.14 A system with a single entry condition: if a then either x or y

$$a \left\{ \begin{matrix} \rightarrow \left[\begin{matrix} x \\ y \end{matrix} \right. \\ \rightarrow \left[\begin{matrix} p \\ q \end{matrix} \right. \end{matrix} \right.$$
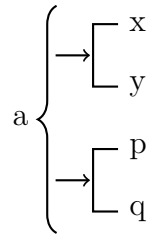
Fig. 5.15 Two systems grouped under the same entry condition: if a then both either x or y and, independently, either p or q

$$\left. \begin{matrix} a \\ c \end{matrix} \right] \rightarrow \left[\begin{matrix} x \\ y \end{matrix} \right.$$
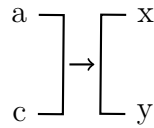
Fig. 5.16 A system network with a disjunctive entry condition: if either a or c (or both), then either x or y

$$\left. \begin{matrix} a \\ b \end{matrix} \right\} \rightarrow \left[\begin{matrix} x \\ y \end{matrix} \right.$$
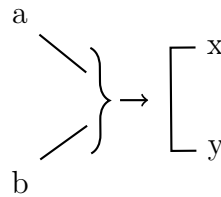
Fig. 5.17 A system with a conjunctive entry condition: if both a and b then, either x or y

Now that the graphical notations are introduced, I would like to start with the abstract concept of *hierarchy* defined in a computer scientific way by Carl Pollard & Ivan Sag (1987). This is a formal rephrasing of Definition 2.2.1 that Haliday provides.

Definition 5.8.1 (Hierarchy)**.** A hierarchy is finite bounded complete partial order $(\varDelta, \prec)$.

The next concept that required higher order of formalization os that of a System first established in Definition 2.2.8. For precision purposes, this one has a narrower scope without considering the system networks or precondition constraints which are introduced shortly afterwards building upon current one.

Definition 5.8.2 (System). A *system* $\Sigma = (p, C)$ is defined by a finite disjoint set of distinct and mutually defining terms called a *choice set C* and an *entry condition p* establishing the delicacy relations within a system network; subject to the following conditions:

1. the choice set is a S_{OR} or S_{XOR} conjunction set.

2. the entry condition is a S_{OR}, S_{XOR} or S_{AND} conjunction set.

3.
$$\infty > size(C) \geq \begin{cases} 2, & \text{if } T(C) = S_{XOR} \\ 3, & \text{if } T(C) = S_{OR} \end{cases}$$

There is a set of functions applied to system: $label(\Sigma) = l$ 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 entry condition, and the $size(\Sigma)$ return the number of elements in the system choice set.

Definition 5.8.3 (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

Systems are rarely if ever used in isolation. SF grammars often are vast networks of interconnected systems defined as follows.

Definition 5.8.4 (System Network). A *system network $SN = (r, SS)$* is defined as a hierarchy within set of systems SS where the order is that of systemic delicacy where:

1. S_i is an arbitrary system within the hierarchy $S_i \in SS$

2. $r \in S_i$ is the unique root of the system network with empty precondition $precondition(r) = \varnothing$

3. $p_i = precondition(S_i)$ the entry condition of system S_i.

4. $\tau : f \times S_i \to S_j$ a transition function from a feature $f \in precondition(S_i)$ to a less delicate system $S_j, f \in choices(S_j)$. We say that $S_j \prec S_i$

subject to the following conditions:

1. $\forall x \in \cup\{P_i | \forall P_i \in SN\}, \exists y \in \cup\{choices(S_i) | \forall S_i \in SN\} : x = y$ every precondition value is among the choice values

2. $\forall x \in \cup\{P_i | \forall P_i \in SN\}$ there is a path π (i.e. a sequence of systems) such that $\tau(x, \pi) = r$ (ensuring the connectedness of entire systemic network and a unique root)

3. $\nexists x \in \cup\{P_i | \forall P_i \in SN\}$ and $\nexists \pi$ such that $\exists S_j = \tau(x, \pi)$ and that $S_j \in \pi \vee x \in values(S_j)$ (ensuring the system network is no cyclical)

Now you may ask a pertinent question: what is the basis on which is the systemic selection made? To answer it I must first introduce two types of constraints. First, The systems are interconnected with each other by a set of preselection (entry) conditions forming systemic networks (Definition 5.8.4). Second, is an aspect not always mentioned in the SFL literature, the systemic *realisation statements* which are shaping the context where the system is applied. These aspects are covered in Section 5.9 talking about execution of system networks.

The notation for writing system networks from (Halliday & Matthiessen 2013) uses colon (:) to symbolize entry condition leading to terms in systems, slash (/) for systemic contrast (disjunction) and ampersand (&) for systemic combination (conjunction). So a sample network will be written as follows:

(73) $\varnothing : i_1/i_2/i_3$

(74) $i_1 : i_4/i_5$

(75) $i_2 \,\&\, i_4 : i_6/i_7$

However in this thesis we need to account for the disjunction type and system name. So we adopt a slightly different notation of three slots separated by colon (:) where the first slot signifies the system name, second the set of system features and the third is the entry condition. Examples 76 to 78 show three systems definitions (without selection functions i.e. no realization statements).

(76) $S_1 : OR(i_1, i_2, i_3) : \varnothing$

(77) $S_2 : XOR(i_4, i_5) : OR(i_1)$

(78) $S_3 : XOR(i_6, i_7) : AND(i_2, i_4)$

The system network can be represented as a graph where each node is a system and edges represent precondition dependencies. All system features must be unique in the network i.e. $\forall S_1, S_2 \in SN : choice\_feature\_set(S_1) \cap choice\_feature\_set(S_2) = \varnothing$ and there must be no dependency loops in the system definitions.
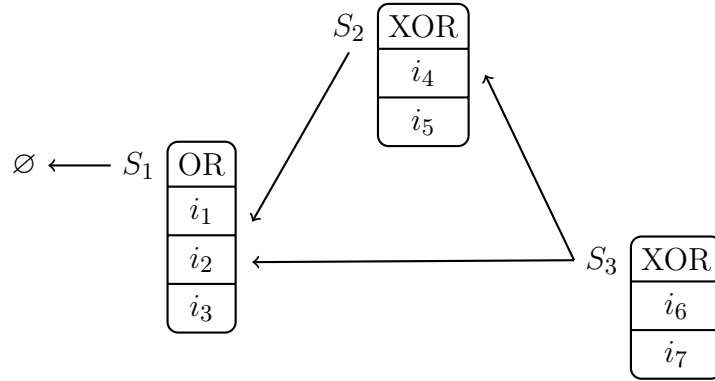
Fig. 5.18 Example System Network presented as graphs

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 and *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.

When the features are selected from systems within a network they form a path. It is often useful to check whether a set of arbitrary features belong to a *consistent* and *complete selection path*. Next I introduce a few concepts useful in addressing this task.

First a system network can be reduced to a graph of features called feature network (Definition 5.8.5 sometimes referred to as *maximal selection graph*) interconnected by system entry conditions.

Definition 5.8.5 (Feature Network). We call *Feature Network $FN(N, E)$* a directed graph whose nodes N are the union of choice sets of the systems in the network and edges E connect choice features with the entry condition features. Formally it can be expressed as follows:

1. $N = \bigcup choices(\Sigma_i)$ where $\Sigma_i \in SN$ for $0 < i size(SN)$

2. $E = \{(f_m, f_n)\}$ where $f_m \in choices(\Sigma_i), f_n \in precondition(\Sigma_i)$

The Feature Network in fact is an expansion of the System Network. The former is a network of interconnected features while the latter a network of systems.
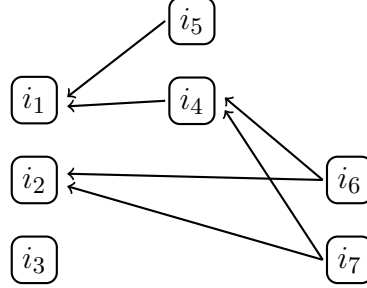
Fig. 5.19 Example Feature Network

Definition 5.8.6 (Selection Path)**.** A *Selection Path $SP(N, E)$* is a connected subgraph of the Feature Network representing system network instantiation through choice making traversal.

Definition 5.8.7 (Complete Selection Path)**.** A *Complete Selection Path* is a selection path starting from the network root and ending in one of the leafs.

We use terms related to age to underline order in which systems activated i.e. older systems must be chosen from before younger ones.

Definition 5.8.8 (System Network Instance)**.** A *System Network Instance SNI* of a constituent node n is a directed graph representing the union of all Complete Selection Paths applicable to a constituent.

Let's come back to Figure 5.19. As you can notice this is a handy device for efficiently checking the path completeness (whether the path is from head to tail of a feature network), consistency with respect to the order of elements (whether such a path exits). There is one aspect that cannot be checked in feature network and it is the conjunctive entry conditions which require that both system networks precede any choice in the current one. In other words, a conjunctive entry states that two paths merging into one and they can only be checked in isolation as two distinct paths, which happen to share a common portion. This shorcoming will be dressed in future work.

In this section there were mentions to selection, instantiation and traversal processes but no specific definition were provided. Next, let's turn our attention towards the system network instantiation through traversal and selection.

5.9   Systemic Network Execution

Every node from a constituency graph is enriched with feature selections grammatically characterising it. This is an important stage in the parsing algorithm discussed in Section 6.5. The enrichment stage is in fact system network instantiation and ascription of complete selection paths to each constituent node .

Executing a system network is an incremental process that builds selection paths by making choices in the system networks. There are two ways to *instantiate* (or execute) a system network: either by *forward activation* or *backward induction* processes which both imply a different order of network traversal.

When it comes to traversing system networks and making choices there is a specific mechanism responsible for this instantiation process. The *choice makers* are selector functions associated to (some) systems. Selector functions implement realization statements corresponding to a system S_i and represents the instantiation mechanism turning the generic set of alternative choices into a concrete choice for a specific context.

Each node in a constituency graph carries features whose names and values are constrained to the set of systems defined in the grammar. In this sense, systems represent constraint definitions for what features may be used and what values those features can take. The algorithm has to evaluate these constraints in order to select the set of relevant features for a given constituent. Traversing system by system within the systemic network, with a known previously selected set of features and a given syntagmatic structure a selector function is executed to make the systemic choice.

Definition 5.9.1 (Selector Function)**.** A *selector function* $\sigma_{ctx} : S \to R$ is defined from a system S to a feature structure R within a given context *ctx* where:

1. the context $ctx = (G, fn)$ is a binary tuple of a constituency graph G and a focus node $fn \in G$ belonging to it

2. *preselection feature set* (PFS) is the already assigned set of features to the focus node $pfs = featureSet(fn)$

3. $size(R) \in \{0, 1\}$ meaning that there is either no choice made and an empty feature structure is returned or there is a choice made and a feature structure is returned with one feature bearing values from the system choice set

subject to the following condition:

1. if $size(R) = 1$ then for the only $f_i \in R$ it holds that $att(f_i) = name(S) \land val(f_i) \subset choices(S) \land val(f_i) \neq \varnothing$

If the PFS is an *OR set* then it requires that any of the features (at least one) must be in a Selection Path (Definition 5.8.6). If the PFS is an *AND set* then it requires that all of the features must be in a Selection Path.

5.9.1   Forward Activation

Forward activation is a process that enables systems to be executed (chosen from by selection function) only after choices from an older system has been already added to a selection path. In other words the selection path is constructed from older to younger systems/features.

We say that a system S_y *activates* another system S_o if and only if $\forall S_o, S_y : S_o < S_y, precondition(S_y) \cap choices(S_o) \neq \varnothing$. Activation process is the process that ensures advancement from an older to a younger system. This implies checking and ensuring entry condition is satisfied and executing the selection function. If the entry condition of the younger system is simple then the choice in the old system suffices, however if the entry condition is a complex conjunction, then first the older sibling systems have to be selected from before entering the younger one.

Algorithm 1: Forward Activation Algorithm

---

   input : sp (current selection path), sn (system network), node (constituent),
             cg (constituency graph)
1 def forward_activate(sp, sn, node, cg):
2   **for** system **in** sn *systems activated by the last* sp *feature***:**
3   get choice set by executing system selection function (given system, node, cg)
4   append sp by the choice set
5   **if** sp *has changed***:**
6   forward_activate (updated sp, sn, node, cg)

Algorithm 1 outlines how the forward activation is executed recursively. The systems that are active at a particular moment of the depend on the configuration of the *selection_path. activated_systems* function returns a set of systems from the system network whose preconditions are satisfied and their choices are not in the selection path (or the system has not yet been executed) $\forall S \in SN : precondition(S) \subset selection\_path$, $choice\_set(S) \cap selection\_path = \varnothing$.

For each activated system, its selector function is executed returning a selection set. The result selection is is used to extend the the selection_path thus potentially fulfilling preconditions of younger systems. If the path has been changed then the same

procedure is applied recursively to the updated path until no more changes are done to the selection_path.

5.9.2 Backwards Induction

Backwards induction is a process opposite to forward activation. If a system is executed yielding a selection set then the preconditions of this system are induced as valid selections in the older systems defining those precondition features, and so on until a system is reached with no preconditions.

Algorithm 2: Naive Backwards induction

   input : sp (current selection path), sn (system network), node (constituent),
               cg (constituency graph)

1 def backwards_induction_naive(sp, sn, node, cg):
2   **for** system **in** sn *systems preconditioning selection* sp *features***:**
3   get choice set by executing system selection function (given system, node, cg)
4   **for** *induced_system* **in** *dependecy_chain(act_sys,sn)***:**
5   choice_set.add(precondition_set(induced_sys))
6   selection_path += create_selection_path_from(choice_set)
7   **return** *selection_path*

The naive approach to is represented in Algorithm 2 which executes the selection functions of leaf systems and the yielded selections induce choices in the older systems through the precondition chain down to the oldest systems of the network.

So for example if SYNTACTIC-TYPE system in Figure 5.20 is executed and yields *verbal-marker* feature then the Algorithm 2 will add to the selection path the chain $negative \rightarrow interpersonal \rightarrow syntactic \rightarrow verbal - marker$.

This approach works very well in classification networks or networks covering a concise vocabulary such as determiners or pronouns. Such network has selection functions on the leaf systems only. However if in the middle of the selection path there are systems with selection functions then the there may exist a conflict between what is induced through precondition of younger systems and what is yielded by the selection function.

In fact confronting the preconditions with selection function is a good technique to verify whether the SN is well constructed. Following the previous example let's imagine that INTERPERSONAL-TYPE system has it's own selection function and it yields the *morphological* feature same time when the *verbal-marker* is selected in the

SYNTACTIC-TYPE. Since the precondition of the latter system is the selection of *syntactic* feature, then we have a mismatch in either the way systems are constructed and the precondition of the latter system needs to be changed or the selection function is poorly implemented in the former system.

The Algorithm 3 implements the verification of whether the induced features match those from the selection function.

Algorithm 3: Backwards Induction with verification mechanism

---

1 **def** `backwards_induction_verified`(*list_of_leafs, sn, constituent, mcg*)**:**
2   **for** *act_sys* **in** *list_of_leafs***:**
3   choice_set = execute_selection_function(act_sys)
4   **if** *choice_set ≠ ∅***:**
5   induced_system_set = find_dependent(act_sys,sn)
6   **for** *induced_sys* **in** *induced_system_set***:**
7   ind_choice_set = selection_function(induced_sys)
8   minimal_valid_set = precondition_set(induced_sys) ∩ choice_set(act_sys)
9   **if** *minimal_valid_set ⊆ ind_choice_set***:**
10   selection_path += create_selection_path_from(choice_set)
11   **else:**
12   **raise** *Execption: The precondition set different from selection function result*
13   `backwards_induction_verified`(*induced_system_set, sn, constituent, mcg*)
14   **return** *selection_path*

It is a recursive algorithm that executes a system S_1, and also the systems $S_2..S_n$ which S_1 depends on an then verifies if $\forall S_i \in dependent\_systems(S_1) : precondition\_set(S_1) \cap choice\_set(S_i) \subseteq selection\_function(S_i)$
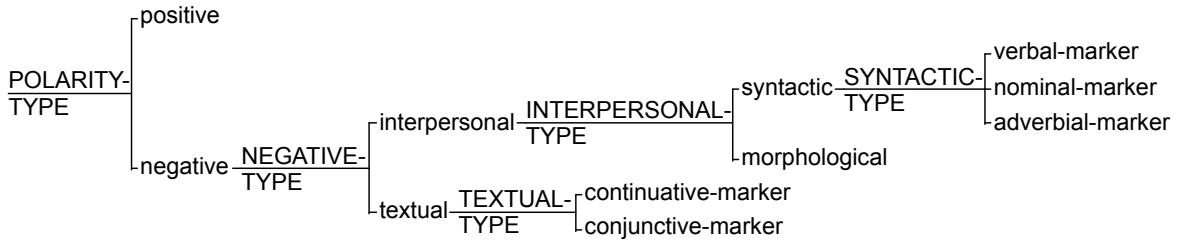
Fig. 5.20 Polarity System

Take for instance the POLARITY system in Figure 5.20. Its default selection is *positive* feature unless there is a negative marker. So we must asses NEGATIVE-TYPE

system to resolve POLARITY system. But NEGATIVE-TYPE also must be postponed because we do not know if there is a negative marker unless we run tests for each marker type (i.e. presence of a "no" particle, negative subject or adjunct etc.). So we postpone selection decision and activate further the INTERPERSONAL-TYPE and TEXTUAL-TYPE systems and base the assessment on the selections yielded by the latter two systems. The same story is with INTERPERSONAL-TYPE which can make selections based on what SYNTACTIC-TYPE system yields. If SYNTACTIC-TYPE and TEXTUAL-TYPE systems yield no selection then we return recursively to INTERPERSONAL-TYPE and to NEGATIVE-TYPE and yield no selection in those systems as well. However if, for instance, *verbal-marker* is detected in the clause then the *syntactic* feature is yielded by the INTERPERSONAL-TYPE and *interpersonal* by the NEGATIVE-TYPE and thus *negative* is yielded by the POLARITY-TYPE.

Moreover the negative markers can be of various types and more than one can occur simultaneously without any interdependence between them so the algorithm needs to check presence of every type of negative marker i.e. verbal, nominal adverbial, conjunctive and continuative markers.

That being said the intermediary systems and features i.e. interpersonal, textual, syntactic, morphological are there for the classification purpose only and do not carry any particular algorithmic value making the network from Figure 5.20 reducible to the one in Figure 5.21.
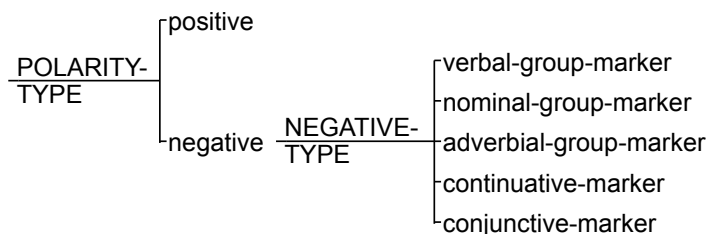
Fig. 5.21 Condensed Polarity System

Execution of system networks is subject to constituent *enrichment phase* of the parsing algorithm. Reducing the POLARITY network to the one in Figure 5.21 would lead to loss of information which may be relevant for choice-making in other systems (e.g. MODALITY) so it is useful to expand the selection set with dependent features to achieve feature rich constituents.

5.10   Discussion

This chapter describes the elemental data structure and the kinds of operations that current implementation applies to generate the SFG parse structures. It lays down the foundations for next chapter which focuses on the parsing pipeline and algorithms.

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 excellent candidate for expressing *systemic realization rules*.

Robin Fawcett recurrently emphasises the role of realization rules in the composition of system networks. He often stresses "no system networks without realization rules". They are important because they formally express ways in which a feature is identified or realised. 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 2003a: 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. 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.

Moreover the expression of rules is proposed in terms of atomic operations such as lexify, preselect, insert, order, etc. Which may not always be fully transparent to the grammarian. Expressing realization rules as operations contextualised in fragments of parse structure is a promising way to ease the grammar authoring process. They could then be used directly by the parser to recognise such structures making the corpus annotation and grammar construction an in-parallel evolving process.

The data structures and operations described in this chapter can be a suitable approach to address the problem of missing realisation rules from the system networks. To do so however requires creation of a system network authoring tool (such as the one available in UAM Corpus Tool (O'Donnell 2008b)) which besides systemic network editor should contain also a graph pattern editor allowing association of graph patterns to systemic features and .

In current parser the pattern graphs are represented as compositions of Python dictionaries and lists such as the one below.

```
{
    NODES: {
        "cl": [
            {C_TYPE: 'clause',
             VOICE: ACTIVE},
            {CONFIGURATION: ['two-role-action', ['Ag', 'Ra', 'Cre']], }],
        'pred': [
            {C_TYPE: [PREDICATOR, PREDICATOR_FINITE], },
            {VERB_TYPE: "main", PROCESS_TYPE: 'two-role-action'} ],
        'subj': [
            {C_TYPE: SUBJECT, },
            {PARTICIPANT_ROLE: 'Ag'}],
        'compl1': [
            {C_TYPE: [COMPLEMENT, COMPLEMENT_DATIVE], },
            {PARTICIPANT_ROLE: 'Ra'}],
        'compl2': [
            {C_TYPE: [COMPLEMENT, COMPLEMENT_ADJUNCT, ], },
            {PARTICIPANT_ROLE: 'Cre'}],
    },
    EDGES: [
    ['cl', 'pred', None],
    ['cl', 'subj', None],
    ['cl', 'compl1', None],
    ['cl', 'compl2', None],]
}
```

This Python dictionary contains two top keys: NODES defined as with node identifiers each associated with a set of systemic features and EDGES defined as a list with three tuples of source, target and eventually a dictionary of features. The nodes contain a list of two dictionaries. The first dictionary enlists the features that the backbone structure should already carry, and against which the pattern matching is performed. The second dictionary contains the set of features that the node shall receive in case of a successful match of the entire pattern.

Writing such structures is cumbersome and requires in depth knowledge of the parser and employed system networks therefore the need for an editor is even higher. Unfortunately building such an editor is out of the scope of the current work and is among the priorities in the future developments just as switching to better technology for working with graphs such as Semantic Web suite of tools. This and other future work are described in the Section 9.4.

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 6

Mood parsing: the syntax

6.1 Algorithm overview

6.2 Preprocessing – canonicalization of DGs

The Stanford Parser applies various machine learning(ML) techniques to parsing. It's accuracy increased over time to $\approx 92\%$ for unlabeled attachments and $\approx 89\%$ for labeled ones (in the version 3.5.1). This section addresses known error classes of wrongly attached nodes or wrongly labelled edges and nodes.

As the Stanford parser evolved, some error classes changed from one version to another (v2.0.3 – v3.2.0 – 3.5.1). Also the set of dependency labels for English initially described in (Marneffe & Manning 2008a,b) changed to a cross-linguistic one (starting from v3.3.0) described in (Marneffe et al. 2014).

Beside stable errors, there are two other phenomena that are modified in the preprocessing phase: *copula* and *coordination*. They are not errors per se but simply an incompatibility between how Stanford parser represents them and how they need to be represented for processing by the current algorithm and grammar.

In this section I describe a set of transformation operations on the dependency graph before it is transformed into systemic constituency graph. The role of preprocessing phase is bringing in line aspects of dependency parse to a form compatible with systemic constituency graph creation process by (a) correcting known errors in DG, (b) cutting down some DG edges to form a tree (c) changing Stanford parser's standard handling of copulas, coordination and few other phenomena. This is achieved via three transformation types: (a) *relabelling of edge relations*, (b) relabelling node POS, and (c) reattachment of nodes to a different parent.

6.2.1   Loosening conjunction edges

Stanford parser employs an extra edge for each of the conjuncts such that there is one indicating the syntactic relationship to the child or parent nodes (just like for any other nodes) and additionally one that shows the conjunction relationship to its sibling nodes. This process removes the parent or child relations except for the first conjunct and leaves only the sibling relationships.

     Some common patterns occurring between noun, verb and adjective conjuncts are depicted below in figures 6.1 - 6.5.
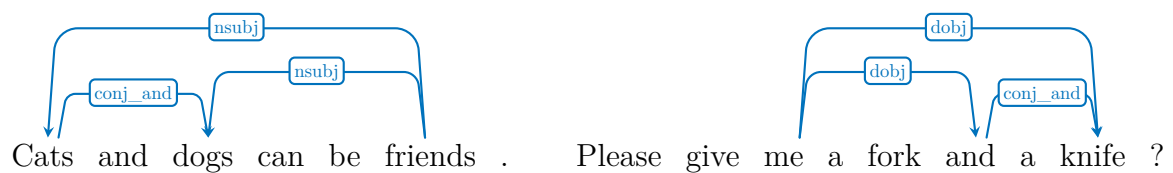
Fig. 6.2 Conjunction of noun objects
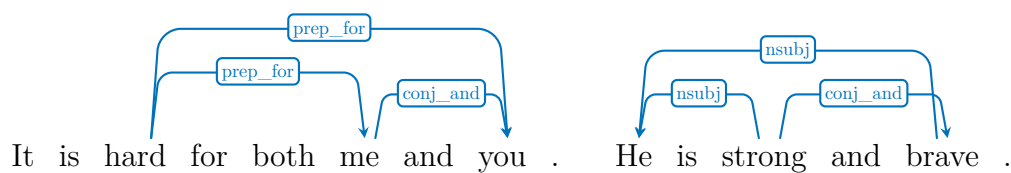
Fig. 6.4 Conjunction of copulatives sharing the subject
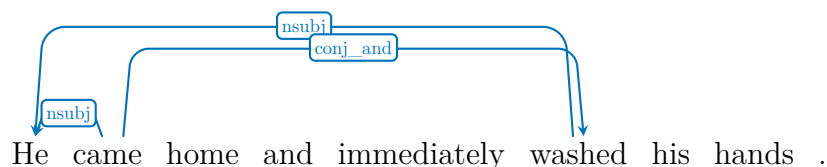
Fig. 6.5 Conjunction of verbs sharing the same subject

     The main reason these extra edges need to be removed is to avoid double traversal of the same node via different paths (which will lead to creation of two constituents). For example, if multiple subject relations occur in the DG then multiple subject are going to be instantiated in CG which is not intended in the grammar. Rather only one complex unit needs to be created with the subject role composed of two noun phrases (see discussion in the Section 2.4.6).

     The straight forward way to fix it this problem is removing functional edges to/from each conjunct except the first one. There are two generic patterns in figures 6.6 and

6.8 correspondingly with incoming and outgoing edges that are transformed into the forms depicted in 6.7 and 6.9.

I split the cases into two: patterns with incoming dependency edges and outgoing ones. First, see the pattern of conjuncts with *incoming dependency* relations represented in Figure 6.6 and exemplified in Figures 6.1 - 6.3. In SFG terms it corresponds to cases when the functional element of a parent constituent is filled by a complex unit below.
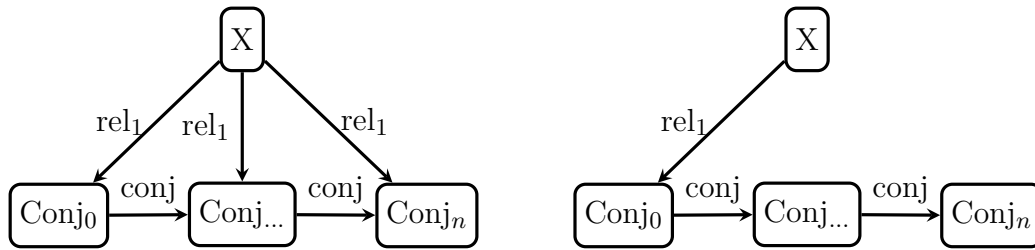
Fig. 6.7 Conjuncted elements with incom-
ing loosely connected dependen-
cies

The second is the pattern of conjuncts with *outgoing dependency relations* depicted in Figure 6.8. In SFG terms it correspond to cases when a unit is sharing an element with another conjunct unit. These are mainly the cases of conjuncted verbs or copulas and are further discussed in the Chapter 4 about null elements. In GBT terms, the second to last conjuncts may miss for example the subject constituent if the conjuncts are verbs or copulas as in Figures 6.4 - 6.5.
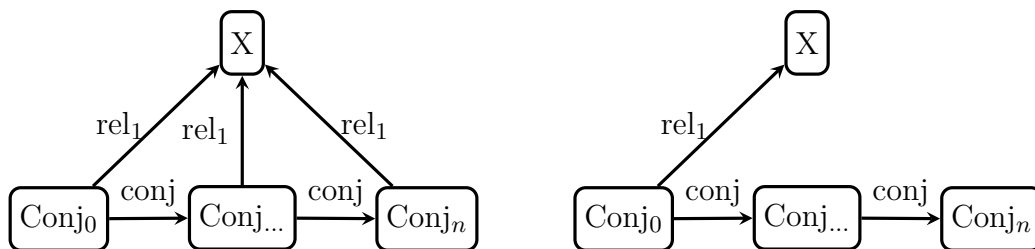
Fig. 6.9 Conjuncted elements with outgo-
ing loosely connected dependen-
cies

6.2.2   Transforming copulas into verb centred clauses

In Stanford dependency grammar *copular verbs* are treated as dependants of their complements (see Figures 6.10 and 6.11) because of the intention to maximize connections

between content words. This configuration breaks the rule of the main verb being the head of clause discussed in Sections 2.5.1 and 2.5.2.

Moreover, despite that a variety of verbs are recognised as copulative e.g. *act, keep, sound*, etc. Stanford parser provides copula configurations only for the verb *to be* leading to unequal treatment of copular verbs.

This case is sometimes accompanied by two relations that create cycles in th DG. They are the *xsubj*, the relation to a controlling subject and *ref*, the relation to a referent. The two relations are removed and their resolution is transferred to the semantic analysis stage of the algorithm.

Fig. 6.11 Conjunction of copulatives sharing the subject

To make the copulative verb the roots of its clause, following rules are implemented. First, some relations are transferred from the copula complement (adjective JJ or noun NN) to the copulative verb. The transferred relations are listed in Table 6.1 which distinguishes them based on the part of speech which of the *copula complement*.

part of speech	dominated relation
NN	dep, poss, possesive, amod, appos, conj, mwe, infmod, nn, num, number, partmod, preconj, predet, quantmod, rcmod,ref, det
JJ	advmod, amod, conj

Table 6.1 Relations dependent on the POS of the dominant node

Second, all the outgoing connections from the copula complement are transferred to the verb except those listed in second column of table 6.1, these relations must stay linked to the NN or JJ nodes. Third the *cop* relation is deleted. Fourth, all the incoming relations to the copula complement are transferred indistinguishably to the verb because these are all clause related and shall be linked to the clause dominant node. Finally the *dobj* link is created from the verb to the complement noun/adjective.

Figure 6.12 represents the generic pattern of copulas in Stanford DGs. The outgoing relations are distinguished between those in the filter as *rel_dep* and the rest simply as *rel* while the incoming relations are not discriminated. Figure 6.13 captures the final state of the transformation where the filtered outgoing relations stay attached to the

complement node while the rest incoming and outgoing relations are moved to the verb.
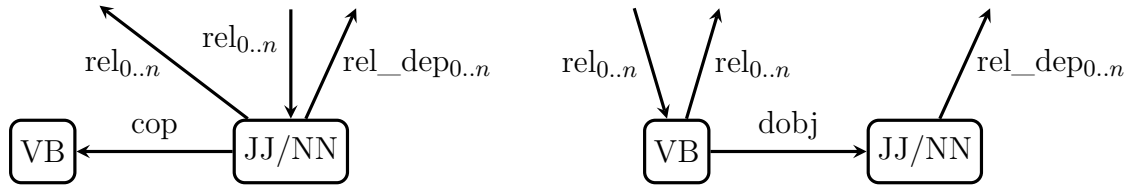
Fig. 6.13 Generic pattern for copulas after the transformation (the same as non-copular verbs).

In case of conjuncted copulas like in the example in Figure 6.11 the approach is slightly complicated by the fact that copula resolution algorithm shall be executed for each copula conjunct, however because of the previous step which is loosening the conjunction and removing graph cycles then only the first copula conjunct is concerned.

6.2.3 Non-finite clausal complements with adjectival predicates (a pseudo-copula pattern)

The Figure 6.14 represents a dependency parse exemplifying a clausal complement with an adjectival predicate. In this analysis there is a main clause governed by the verb *to paint* and the second one by the adjective *white*. In SFL Figure 6.14 receives a different analysis as it is represented in Table 6.2.
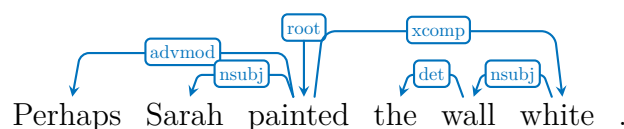
Fig. 6.14 Dependency parse for clausal complement with adjectival predicate

Perhaps	*Sarah*	*painted*	*the*	*wall*	*white.*
Adjunct	Subject	Finite/Main Verb	Complement		Complement
	Agent	Material Action	Affected		Attribute

Table 6.2 SFG analysis with attributive adjectival complement

xcomp relation defined in (Marneffe & Manning 2008a) to introduce non-finite clausal complement without a subject. Dependency grammar allows adjectives (JJ)

and nouns (NN) to be heads of clauses but only when they are a part of a copulative construction. In figure 6.14 it is not the case, there is no copulative verb *to be* and also *the wall* receives the subject role in the complement clause which should be absent.

So I treat it as a misuse of *xcomp* relation and the adjective should not be treated as governing a new clause but rather non-clausally complementing the verb *to paint.* Moreover that in SFG adjectival predicates are not allowed.

Certainly, depending on the linguistic school, opinions may diverge on the syntactic analysis comprising one or two clause. But when analysed from a semantic perspective it is hard to deny that there is a Material Process with an Agent and Affected thing which is specifying also the resultant (or goal) Attribute of the Affected thing.

To accommodate such cases the dependency graph is changed from pattern in Figure 6.15 to form in Figure 6.16. The *xcomp* relation is transformed into *dobj* and the subject of the embedded clause (if any) becomes the direct object (*dobj*) in the main clause.
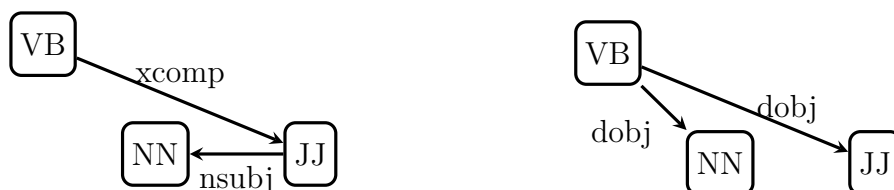
Fig. 6.16 Adjectival clausal complement
         as secondary direct object

6.3   Preprocessing – error correction

As noted by Cer et al. (2010) the most frequent errors are related to structures that are hard to attach i.e. prepositional phrases and relative clauses. During the implementation of current parser there had been discovered a set of errors, most frequent of which are described in this section and how are they treated. These errors are specific to Stanford Parser versions v2.0.3 – 3.2.0. This section may constitute a valuable feedback for SDP error analysis.

6.3.1   *prep* relations from verb to free preposition

As noted before only the collapsed version of the DGs are taken as input. This means that no pure *prep* relations shall occur but their expended version with the specific preposition appended to the relation name i.e. *prep_xxx.*

This is not always the case especially with phrasal verbs, the *prt* relations are mislabelled as *prep*. The correction consist in changing the *prep* (figure 6.17) into *prt* (figure 6.18) if the preposition node has no children e.g. *pobj.*

Fig. 6.18 Corrected relation to free preposition as verbal particle

6.3.2   Non-finite clausal complements with internal subjects

The *xcomp* relation stands for open clausal complements of either a verb(VB) or adjective (JJ/ADJP). The latter is actually transformed as discussed in Section 6.2.3. The open clausal complement defined in Lexical Functional Grammar (Bresnan 2001: p270–275) is always non-finite and does not have its own subject. However sometimes *xcomp* relation appears either (a) with finite verbs or (b) with own local subjects and both cases correspond to definition of *ccomp* relation.

To fix this I transform all the instances of *xcomp* relation to *ccomp* if the dependent verb has a local subject (nsubj) or a finite verb as depicted in Figures 6.19 - 6.20.

Fig. 6.20 Corrected clausal complement

6.3.3   The first auxiliary with non-finite POS

Sometimes the first auxiliary in a clause is mistakenly labelled as a non-finite verb. For some words the exact POS is less important as it has not big impact on the CG graph and features but in the case of first auxiliary verb of a clause it makes a big difference. It has an impact on determining the finiteness of the clause in a latter stage of the algorithm.

The algorithm is thus checking that the POS of the first auxiliary is according to the mapping defined in the Table 6.3.

word	POS	notes
shall, should, must, may, might, can, could, will, would	MD	modals
do, have, am, are	VBP	present
has, is	VBZ	present 3rd person
did, had, was, were	VBD	past

Table 6.3 Mapping lexical forms of auxiliaries to their POS

6.3.4 Prepositional phrases as false prepositional clauses

prepc is a relation that introduces, via a preposition, a clausal modifier for a verb, adjective or noun. Assuming that the copulas had been changed as described in subsection 6.2.2 then the head and the tail of the relation can only be a verb. However when the relation head is not a verb (only nouns encountered so far) then the relation needs to be corrected from *prepc* to *prep* introducing a prepositional phrase rather than a subordinate clause.

Fig. 6.22 Corrected prepositional phrase

6.3.5 Mislabelled infinitives

In English base form of the verb often coincides with present simple form (non $3\mathsf{sp}(rd)$ person). Therefore the POS tagger sometimes mislabels infinitive (VB) as present simple (VBP) the verb is and vice versa.

The algorithm checks the presence of the preposition *to* (linked via *aux* dependency relation) in front of the verb. If the preposition is present then the verb POS is changed to VB and reverse, if the auxiliary preposition is not present the verb POS is changed into VBP.

Fig. 6.24 Correct infinitive

Fig. 6.26 Correct present simple

6.3.6 Attributive verbs mislabelled as adjectives

In English, *attributive verbs* often have the same lexical form as their corresponding adjectives. This is a reason for POS being mislabelled adjective(JJ) instead of verb (VB) leading to situations when an adjective (JJ) has an outgoing subject relation which means that its POS should actually be VB. The algorithm checks for such cases and corrects the JJ POS into VBP (non $3\mathsf{sp}(rd)$ person present simple).

Fig. 6.28 Corrected attributive verb

6.3.7 Non-finite verbal modifiers with clausal complements

The early version of Stanford Dependencies (Marneffe & Manning 2008a) proposes two relations for non-finite vernal modifiers *partmod* for participial and *infmod* for infinitival forms exemplified in 79.Latter in (Marneffe et al. 2014) both relations have been merged into the *vmod*.

(79)   Tell the boy playing the piano that he is good.

Clauses such as "(that) he is good" following immediately after the qualifier clause ("playing the piano" in the example 79) are problematic with respect to where shall they be attached: to the main clause or to the modifying one. This problem is similar to the prepositional phrase attachment problem.

In this case, of course, attachment would depend on whether the verb accepts a clausal complement or not. In the example 79 the verb *to play* does not take clausal complements then the clause "that he is good" is complementing "tell the boy". Stanford parser does not take into consideration such constraints and sometimes provides an incorrect attachment.

This type of error can be captured as the graph pattern in Figure 6.29 which is transformed by the algorithm into the form represented in Figure 6.30

Syntactic structure is not enough to capture this error which originates in the semantic influences on the syntax. To grasp the them an extra constraint check is
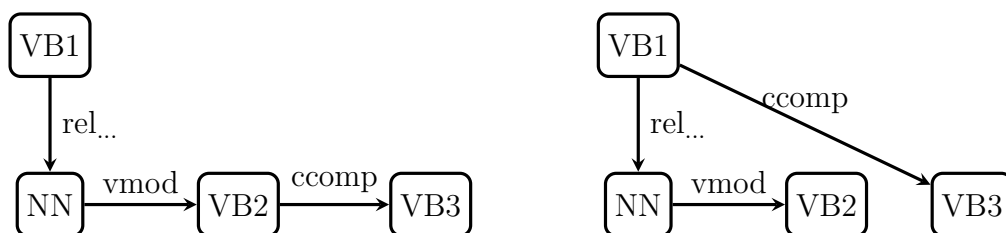
Fig. 6.30 Clausal complement attached to
the main clause

the possible lexico-semantic type of the verb. As only verbal and cognition process types can take clausal complements as phenomena then the verb in the higher clause VB_1 heeds to be capable of accepting clausal complement VB_2 before performing the reattachment. In other words, if VB_1 is capable of accepting two complements (i.e. di-transitive) then most likely the VB_2 is a complement, otherwise it is certainly not.

6.3.8   Demonstratives with a qualifier

Demonstratives (*this, that, these, those*) occurs as both determines and as pronouns. In English, when demonstratives are used as determiners, they function as Deictic element of a nominal group i.e. modifying the head of the nominal group. When used as pronouns, demonstratives never form phrases but occur as single words filling a clause element. Translated into dependency grammar demonstratives may have as parent either a noun (NN) or a verb (VB*).

   Examples below show uses of demonstratives in both cases. The word (thing/things)* enclosed between round brackets are not part of the sentence but are elipted.

   (80)   Bill moved those beyond the counter.

   (81)   Put that in our plan.

   (82)   Look at those (things)* beyond the counter.

   (83)   What is that (thing)* next to the screen?

   (84)   I thought those (things)* about him as well.

   (85)   He felt that (thing)* as a part of him.

When demonstratives are followed by a prepositional phrase the question arises whether it shall be attached to the verb and take a clause role or it should be attached to the demonstrative as post-modifier. I shall note that demonstratives cannot take by themselves a post-modifier in either case as determiner or pronoun.

However there are cases when apparently the post-modifier (prepositional phrase) pertains to the demonstrative like in the examples 82 and 83 and cases such as 84 and 85 when the post-modifier pertains to teh clause.

In fact the only acceptable analysis for apparently a demonstrative with a Qualifier (i.e. post-modifier) can be analysed as noun phrases with the Thing missing (elipted) and the Deictic taking the role of the Head. The implied missing head is the generic noun "thing(s)" or any noun anaphorically binding the demonstrative.

The verb argument structure and syntactic constraints on the arguments described in the Transitivity classification of process types enable precise distinctions of such cases. However at this stage the algorithm does not employ this type of information. Therefore as a rule of thumb, the prepositional phrase following the demonstrative shall be attached to the verb in the case of non-projective[1] di-transitive verbs which are *three role actions* and *directional* processes.

In examples 80 and 81, attaching the prepositional phrase to the demonstratives (depicted in Figure 6.31) is incorrect. It should be attached to the verb (like in the figure 6.32) because the prepositional phrase can function as Destination or Location in each case i.e take semantic roles.

The algorithm detects cases of demonstratives that have attached a prepositional phrase. If the parent verb is a three role action or a directional process then the prepositional phrase is reattached to the verb.
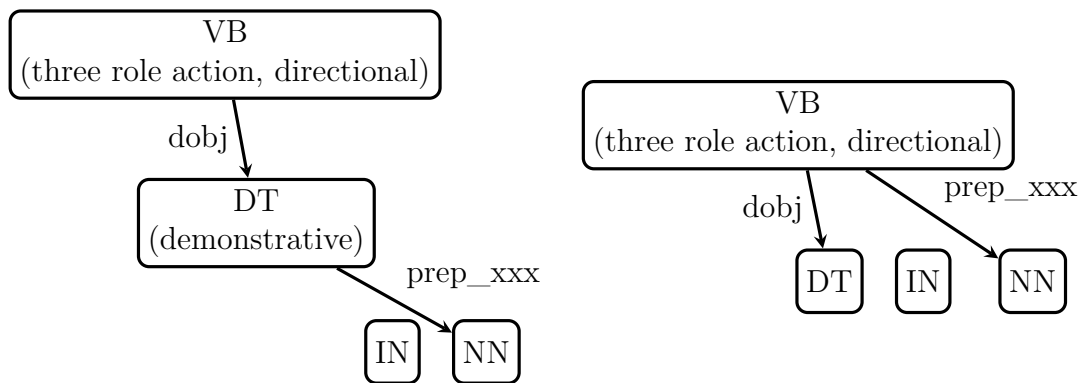
Fig. 6.32 Prepositional Phrase attached to the verb with a demonstrative pronoun in between

Ideally, the algorithm should also change the POS of the demonstrative into pronoun but unfortunately Penn tag-set only contains personal and possesive pronouns. The

---

[1]Projective verbs express cognitive and verbal processes like saying, thinking or imagining and often they verbs are di-transitive.

demonstratives are always labelled as determiners so no POS change is made to the dependency graph but it is properly represented when converted into the constituency graph.

6.3.9   Topicalized complements labelled as second subjects

In generative grammars the topicalization (or thematic fronting) of complements is described in *Trace Theory* as *WH/NP/PP-movement.* Examples 86–91 (from (Quirk et al. 1985: pp. 412-413)) present this phenomena. It is used in informal speech where it is quite common for an element to be fronted with a nuclear stress thus being informationally and thematically stressed. Alternatively this phenomena is used as a rhetorical style to point parallelism between two units and occurs in adjacent clauses like in examples 90–91.

(86)   Joe(,) his name is.

(87)   Relaxation(,) you call it.

(88)   Really good(,) cocktails they make at the hotel.

(89)   Any vitamins(,) I could be lacking?

(90)   His face(,) I'm not found of but his character I despise.

(91)   Rich(,) I may be (but that does not mean I'm happy).

These are difficult cases for Stanford parser (tested with versions up to 3.5.1). None of the above examples are parsed correctly. However, if the comma is present between topicalized complement and the subject, then it produces parses that are closest to the correct one where the topicalized complement is labelled as second subject but still not a complement. So having a comma present helps.
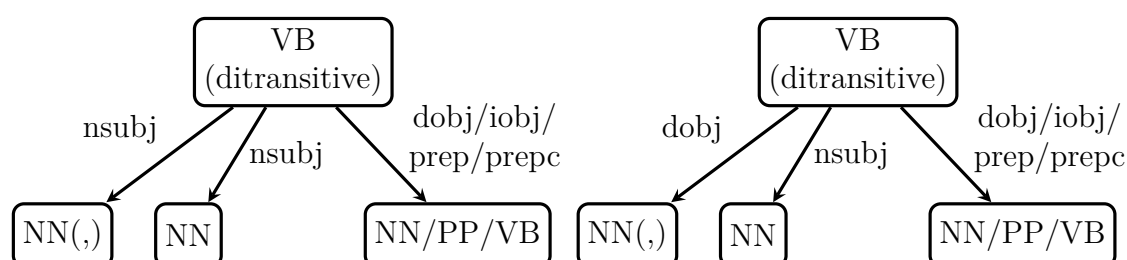
Fig. 6.34 Topicalized   Direct   Object   –
               moved to pre-subject position

The algorithm is looking for the cases of multiple subjects (represented in figure 6.33) and gives priority to the one that is closest to the verb. The other one is relabelled

as a complement (Figure 6.34). The rule is generalized in the algorithm for multiple subjects even if so far only cases of two subjects have been observed.

6.3.10 Misinterpreted clausal complement of the auxiliary verb in interrogative clauses

Sometimes the auxiliary verb in the interrogative clauses (examples 92 and 93) is mistakenly used as a clause main verb. Instead of *aux* relation from the main verb to the auxiliary there is a clausal complement relation from the auxiliary to the main verb.

(92)   Do you walk alone?.

(93)   Has Jane fed the cat?.

The algorithm searched for the pattern depicted in Figure 6.35 and transforms it into the form Figure 6.36.
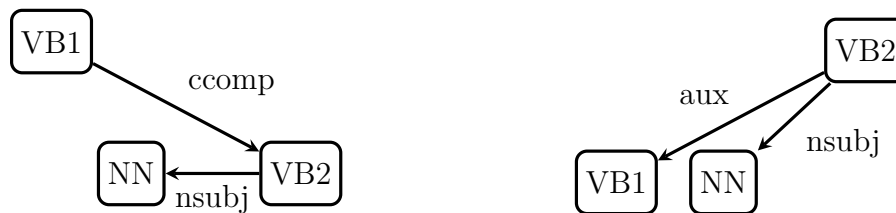
Fig. 6.36 Corrected clausal complement

6.4 Creation of systemic constituency graph from dependency graph

This section describes how the systemic constituency structure is generated from the dependency graph. Even if at first sight they appear isomorphic, DGs and CGs differ in their structure. The CG is created in two phases as presented in the Algorithm 4. The first phase generates an incomplete CG through a top-down breadth-first traversal

of a DG. The second phase complements the first one ensuring creation of all CG constituents through a bottom-up DG traversal.

Algorithm 4: Constituency graph creation

    input **:** dg (the dependency graph), rule table
    output : cg (the constituency graph)

1 begin
2   create the bootstrap cg by top-down traversal
3   complete the cg by bottom-up traversal
4 end

Before presenting the two stages of creation I will first reiterate over the difference in the dependency nature in the constituency and dependency graphs. Then I will also talk about the tight coupling of the two graphs and the rule tables used in traversal.

6.4.1   Dependency nature and implication on head creation

As explained in Section 3.5, the nature of dependency relations is different in dependency graphs and in systemic functional constituency graphs.

The DG uses a *parent-daughter dependency* while in Constituency Graphs there is a *sibling dependency*. This difference implies that, when mapped into CG, a DG node, stands for both a unit and that unit's head. In other words a DG node corresponds to two functions at different rank scales. For example the root verb in DG corresponds to the clause node and the lexical item which fills the Main Verb of the clause.

In current approach, the top-down perspective considers the DG node as representing the upper most rank. The bottom-up perspective considers the DG nodes as representing the lower most rank. Thus the generation algorithm first traverses the graph in a top-down order and generating the units as appropriate and then bottom up in order to create their heads.

The result of the top-down phase is a constituency graph without head nodes. Therefore the bottom-up phase is performed by traversing th constituency graph and not on the dependency graph. The traversal task is to locally resolve which dependency nodes form the syntactic head. The local resolution is possible because of the tight coupling between the dependecy and constituency graphs established in the top-down phase. It is explained in the section below.

6.4.2   Tight coupling of dependency and constituency graphs

At the creation stage, the CG is tightly coupled with the original DG. This allows navigating easily from one graph to the other one via references stored within the nodes of each of them. I say that a graph node is *aware* of it's ascription in another graph if it carries information to which nodes it is linked within the second graph.

Through a stack of CG nodes, the DG nodes are made aware of which CG nodes and in which order they subsume them (Figure 6.37). On the other hand, the CG nodes are also made aware through a list of DG nodes, over which DG nodes do they span (Figure 6.38). This way the positioning information is available bidirectionally about CG an DG structures.
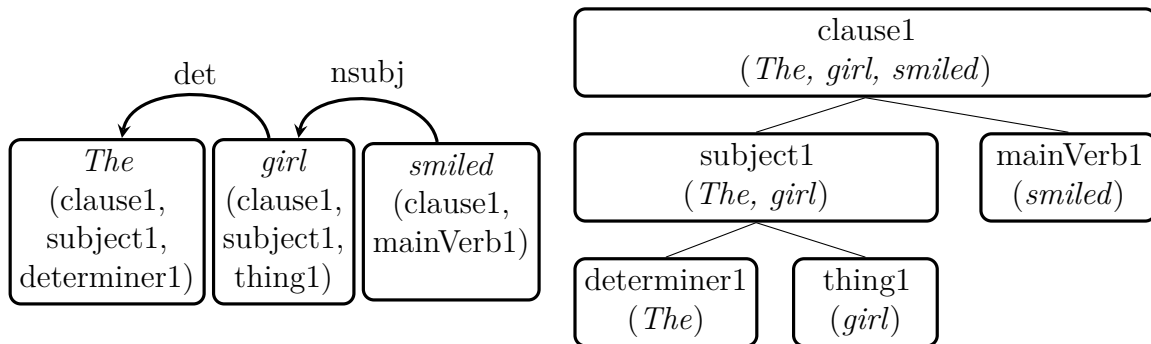
Fig. 6.38 Dependency aware CG nodes

The Figure 6.37 depicts a dependency graph. Each node has a stack of ids corresponding to constituents in the CG (Figure 6.38). Conversely, in Figure 6.38, depicts a constituency graph where the constituent nodes carry a set of tokens corresponding to DG nodes. This way the DG in Figure 6.37 and the CG in 6.38 are aware of each other.

The node awareness has two interesting properties worth exploring. First, the DG nodes receive a vertical constituency strip. Each strip is a direct path from the root to the bottom of the constituency graph where the word of the DG is found. These strips are the very same ones explored in the parsing method explored by Day (2007). Second, the CG nodes receive a horizontal span over DG nodes enabling exploration of elements linear order. These two properties could eventually be explored in future work to inform or verify the correctness of the constituency graph.

6.4.3 Mapping Rule Tables

Constituency Graphs are created through a top-down breadth-first walk of dependency graph. During the traversal each visited edge triggers execution of a *creational operation* on the growing CG on the side. To know what operation to execute a rule table is used where the edge type, head and tail nodes are mapped to a creational operation and eventually a parameter (specifying the element type if a constituent is to be created).

A simplified example of the rule table is presented in Table 6.4. It can be regarded as an attribute value matrix or a Python dictionary where the left column titled: key, contains a unique *dependency graph context* serving as a rule trigger; while the left side column named: value, contains the operation to be executed within the given context.

Current implementation uses three operation types: (a) *creating a new constituent* under a given one (b) *creating a new sibling* to the given one (c) *extend* a constituent with more dependency nodes.

The parameter is used only for the operations (a) and (b) and specifies which element the new constituent is filling as described in Section 2.2 and 2.3. Most of the time there is only one element provided but in the case of prepositional phrases and clauses it is impossible to specify purely on syntactic basis the exact functional role and thus multiple options are provided (Adjunct or Complement) and then in lather parsing phases, which account for the verb semantics, these options ideally are reduced to one.

	Key	Value	
		Operation	*Parameter*
1	nsubj	new_constituent	Subject
2	csubj	new_clause_constituent	Subject
3	prepc	new_clause_constituent	Complement, Adjunct
4	VB-prep-NN	new_constituent	Complement, Adjunct
5	NN-prep-NN	new_constituent	Qualifier
6	VB-advmod-WR	new_constituent	Complement
7	VB-advmod-RB	new_constituent	Adjunct
8	mwe	extend_current	
9	nn	extend_current	

Table 6.4 Rule table example mapping (specific or generic) dependency context to constructive operation

There are two types of keys in the rule table: the *generic* ones where the key consist of the (non-extended) dependency relation and the *specific* ones surrounded by the POSes of head and tail edge nodes taking the form *Tail–relation–Head*. For example

nsubj relation (on row 1) always leads to creation of a Subject nominal constituent regardless if it is headed by a noun, pronoun or adjective. Since all individual cases lead to the same outcome it suffices to map the dependency relation to the creation of a new constituent with Subject role ignoring the POS context of head and tail nodes. The same holds for *prepc* relation (on the table row 3) as it always leads to creation of subordinate clause constituent with Complement or Adjunct roles. So the generic relations can be viewed as equal to the form *Any–relation–Any* only that the nodes are omitted due to redundancy.

In the case of *prep* relation (on the rows 4 and 5) the story is different. Its interpretation is highly dependent on its context given by the parent/tail and child/head nodes. If it is connecting a verb and a noun then the constituent prepositional phrase takes the role of either Complement or Adjunct in the clause. But if the prep relation is from a noun to another noun, then it is a prepositional phrase with Qualifier function in the nominal group.

Some dependency relation are not mapped to operation of creating a new but rather extend the existing constituent with all nodes succeeding current one in the DG. These operation is used for two reasons: either (a) the constituent truly consists of more than one word, for example the cases of multi word expressions (e.g. ice-cream) marked via *mwe* relation (table-row 8) or (b) the relation (with or without it's POS context) is insufficiently informative for instantiating a constituent node and is postponed for the second phase of the CG creation.

Note that the contextualised relations are "slightly" generalised by reducing POS to first two letters which can be up to four letters long. For example nouns generically are marked as NN but they may be further specified as NNP, NNPS and NNS or verbs (VB) may be marked as VBD, VBG, VBN, VBP, and VBZ depending on their form.

Next I explain the top-down traversal phase which is the core essence of the constituency graph creation.

6.4.4   Top down traversal phase

The goal of this first phase is to bootstrap a partial constituency graph starting from a given dependency graph and a table with mapping rules. The CG is created as

ap parallel structure through the process of breadth-first traversal on DG edges as described in Algorithm 5.

Algorithm 5: Top-down CG creation

    input **:** dg (the dependency graph), rule table
    output: cg (the constituency graph)

1 begin

2   create the cg with a root node

3   make the cg root node aware of the dg root node

4   **for** edge **in** *list of* dg *edges in BFS order***:**

5   rule ← find the suitable rule for the current edge in the rule table

6   operation ← get operation from the rule

7   element type ← if any get the parameter from the rule

            `// as cg and dg nodes are aware of each other we can`
                `navigate between them`

8   constituency stack ← the constituency stack from the tail node of the current dg edge

9   cg pointer ← the top node from the constituency stack

10   children ← all child nodes for the current dg edge

            `// constructing or extending the cg with a new node`

11   execute the operation on cg given element type, cg pointer and children

12   **return** cg

13 end

First the CG is instantiated and an empty root node is created within the CG. Also, the root node is made aware of the root node in DG via the mechanism described in Section 6.4.2.

Then the DG is traversed on its the edges in BFS order starting from the root node. As each DG edge is visited an operation is chosen based on the edge type and nodes POS along with computation of an additional set of parameters Lines 7 - 10; after which the creative operation is executed with the established parameters: dependency successors (children), a constituent node parenting the newly created one (cg pointer), element type which is chosen from the rule-table together with the operation. Note that the head nodes are not created in current but the next phase.

The Line 5 of the algorithm is responsible for looking up and selecting the operation from the rule table as described in Algorithm 6. It is based on two lookups based on the rule indexes: one contextualised to the edge relation and its nodes and another one generic based on the edge relation alone.

The rule table is conceived as a Python dictionary with string keys and and two-tuple containing the operation and the element type parameter. If the key is found (either specific or generic) in the rule table then the operation and parameter are returned otherwise None is returned.

Algorithm 6: Operation selection in the mapping rule table based on the edge type

---

input : rule table, edge
output : rule

1 begin
2   generic key ← the simplified dependency relation of the edge
3   head POS ← the POS of the edge head node
4   tail POS ← the POS of the edge tail node
5   specific key ← tail POS + generic key + head POS
6   **if** specific key *index* **in** rule table:
7   rule ← rule indexed with specific key from rule table
8   **elif** generic key *index* **in** rule table:
9   rule ← rule indexed with generic key from rule table
10   **else:**
11   rule ← None
12   **return** rule
13 end

In Python function are first class objects allowing objects to be called (executed) if they are of callable typed. This duality allows storing the functions directly in the rule table and then, upon lookup they are returned as objects but because they are also callable these objects are executed with the expected set of parameters based on their function aspect. The possible operation have been already explained in Section 6.4.3: *extend current* and *new (sibling) constituent*. Next I present the pseudo-code for each of them. Note that these operations do not return anything because their effect is on the input cg and dg.
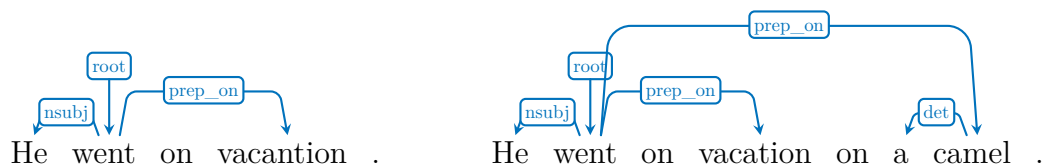
Fig. 6.39 Challenging free nodes

Extend constituent. Algorithm 7 outlines the functionality for extending current cg pointer. It does two main things. It increases the span of an already existing CG node over more DG nodes concomitantly making them aware of each other.

Algorithm 7: Extend a constituent with DG nodes

    input : cg pointer, children, element type, edge, dg, cg

1 begin

    // handling special relations *prep* and *conj*

2   **if** $prep \lor conj$ **in** edge *relation***:**

3   free nodes ← find in dg the free nodes refereed in the edge relation

4   create new node with *marker* function under the cg pointer with free nodes as children

5   children ← children & free nodes

    // making the children and cg pointer aware of each other

6   **for** node **in** children**:**

7   constituency stack ← the constituency stack of the node

8   push the cg pointer to the constituency stack

9   span ← constituent span of the cg pointer

10   extend the span with current node

11 end

If, however, the edge relation is a conjunction or a preposition then the children list is extended with the free nodes that stand for the preposition or conjunction in place and eventually neighbouring punctuation marks (line 3).

This exceptional treatment is due to the fact that *prep* and *conj* relations are always specialised by the preposition or conjunction in place. For details on this aspect of Stanford Dependency Grammar please refer to Section 3.3.

Figure 6.39 exemplifies easy (on the left) and more difficult cases (on the right) of free node occurrence. The challenge in the second figure comes from the fact that there may be two suitable free nodes for the edge *went-prep_on-camel*. Another challenge type in resolving free nodes is when the preposition is a multi word construction.

Once all the free DG nodes are found, a new **cg** node is created with *Marker* **element type** spanning over them (line 4). Then the free nodes are included into the list of **children** and all together are made aware of the current **cg pointer** and vice versa.

Create new constituent. The Algorithm 8 is a function that extends CG with a newly created constituent object (line 4). The new constituent is created as a child of a pointed CG node with the **element type** extracted from the **rule table** together with the **operation**. Once the **cg** node is created, it is extended with the **children** nodes as described in Algorithm 7 above.

Note that only the functional element is assigned to the freshly created constituent. It's class is added in the second phase of the creation algorithm. This is due to the fact that a function can be filled by units of several classes. The bottom up traversal provides a holistic view on the constituency of each unit giving the possibility to assign a class accordingly. For details see the Chapter 2.

Algorithm 8: Creating new child constituent

    input : cg pointer, children, element type, edge, dg, cg

1 begin

2   node ← new Constituent

3   node type ← element type

4   add to **cg** the edge (**cg pointer**, node)
       `// invoking the Algorithm 7`

5   extend **cg pointer** with **children** of **edge**

6 end

A variation of the *create new* is *create sibling* outlined in the Algorithm 9. It sets the newly created constituent as a sibling of the current one and not as a child. This will make the new constituent a child of current **cg pointer**'s parent.

Algorithm 9: Creating new sibling constituent

    input : cg pointer, children, element type, edge, dg, cg

1 begin

2   cg pointer ← get the parent of **cg pointer**
       `// invoking the Algorithm 8`

3   create new constituent to an updated **cg pointer**

4 end

6.4.5   Bottom up traversal phase

Chapter 2 explains that each constituent must specify the unit class and the element it is filling within parent unit. The first phase of the algorithm achieves creating most of the constituents and assigns each unit functional elements derived from the dependency graph.

The constituency graph misses, however, the unit classes and the syntactic head nodes. The second phase complements the first one by fulfilling two goals: (a) creation of constituents skipped in the first phase and (b) class assignment to the constituent units.
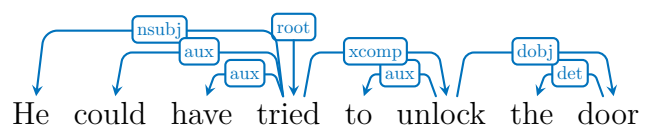
Fig. 6.40 The dependency graph before the first phase

The Figure 6.41 depicts and example CG generated in the $1sp(st)$ phase with dotted lines representing places of the missing constituents.
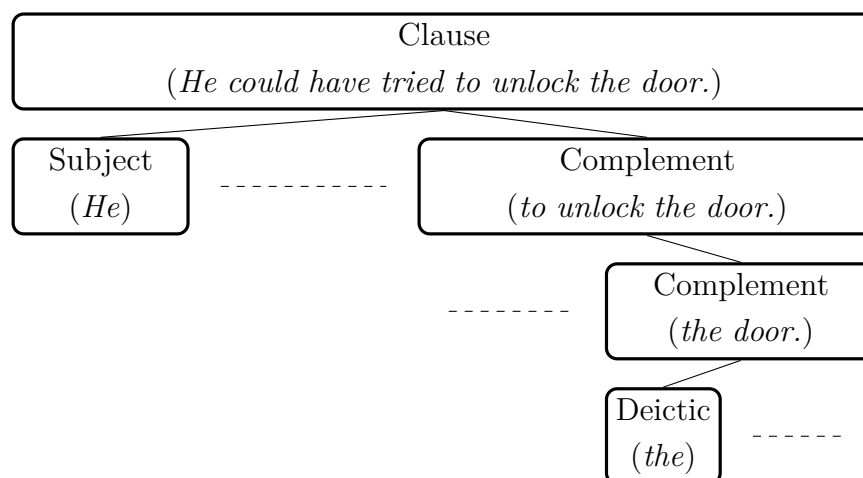
Fig. 6.41 Constituency graph after the top down traversal missing the head nodes

The missing constituents are the syntactic heads for all units. The clause, besides the Main Verb, also misses the Finite, Auxiliary elements. Determining these functions strongly depends on the place within a unit and syntagmatic order in which units occur. As the first phase is performed as graph traversal, the order dimension is not available so they have to be created in the second phase.

The class membership of constituent units is decided based on three informations available within each constituent: (a) part o speech of the head dependency node (b) element type of the constituent and (c) presence or absence of child constituents and their element types. The corresponding constraints are listed in Table 6.5. The first column carries the unit class (to be assigned), the second and third columns enumerate part of speech and element types that constituents might fill. The Second and third column enumerations are exclusive disjunction sets (S_{XOR}) because only one may be selected at a time while the list in last column is an open disjunction (S_{OR}) because any of child elements may be present. The last column is an enumeration of what child constituents might the current one have. The *n/a* means that information is unavailable.

Class	POS of the Head DG Node (XOR)	Element Type (XOR)	Child Constituents (OR)
Clause	VB*	Subject, Complement, Qualifier, n/a	Subject, Complement, Adjunct, n/a
Prepositional Group	CD, NN*, PR*,WP*, DT, WD*	Complement, Qualifier, Adjunct	Marker, n/a
Nominal Group	CD, NN*, PR*,WP*, DT, WD*, JJ, JJS	Subject, Complement	Deictic, Numeral, Epithet, Classifier, Qualifier, n/a
Adjectival Group	JJ*	Complement, Epithet, Classifier	Modifier, n/a
Adverbial Group	RB*, WRB	Adjunct	Modifier, n/a

Table 6.5 Constraints for unit class assignment

The Algorithm 10 traverses the CG bottom-up (not the DG) with *post-order depth-first* order (line 2) during which every visited constituent node is assigned a unit class and missing child constituents are created.

Because the CG and DG are tight coupled which means that each CG node spans over a set of DG nodes then traversing CG is equal to traversing groups of DG nodes at each step. This creates a focused mini context suitable for resolving the unit class and missing elements.

When assigning unit class the following informations are considered: (a) part of speech of the head DG node that triggered node creation in the first phase (head POS), (b) the assigned element type (element type) and (c) element type of each direct child (children of node). Lines 6 to 15 assign classes according to conditions stated in Table 6.5.

Clause classes are assigned to units started by a non-modal verb. This rule corresponds to the one main verb per clause principle discussed in Section 2.5.1. This approach however does not take into consideration elliptic clauses and they need additional resolution. This short coming should be considered in the future by an additional *ellipsis resolution mechanism*, similar to the one for *null elements* described in Chapter 4.

The second part of algorithm creates head nodes for every non leaf constituent and in case the node is a clause then it also creates the clause elements such as: Finite, Auxiliary, Main Verb, Negator and Extension.

After the second stage, all the sentence token must be covered by CG nodes. Moreover the CG nodes build up to a constituency graph that at this stage is always a tree. Provided the class and element type the nodes are ready to be enriched with choices from systemic networks described in the next section.

6.5 Feature enrichment process

In this stage the CG nodes are assigned features from system networks. This is achieved by visiting each CG node in a bottom-up order and based on the node class and/or element function (also refereed to as triggers) the relevant system networks are being activated and executed. The relevancy criteria is established by system's precondition set. Each network has one or a set of selector functions associated to it and when the network is activated then the selector function is executed returning the systemic choices based on the visited node context in the CG graph and features (if any already assigned).

Table 6.6 associates a list of triggers to a list of systemic networks. Table 6.7 associates systems with the choice making functions. Most of the selection criteria have been first implemented as hard-coded Python functions and latter transformed into the Graph patterns with update operations (see Section 5.7 describing pattern based operations).

Next I describe the enrichment algorithm and then treat some of the selection functions for some of the system networks.

Algorithm 10: Creating the head units and assigning classes

   input : cg, dg

1 begin

2   **for** node **in** *list of* cg *nodes in DFS post-order***:**

3   head POS ← POS of the dg entry node of node

4   element type ← currently assigned element of node

5   children ← get assigned elements to the children of node

      // assigning classes

6   **if** head POS **in** *main verb POSs***:**

7   assign node *Clause* class

8   **elif** head POS **in** *nominal POSs* ∧ element type **in** *prepositional element types* ∧ *Marker* **in** children**:**

9   assign node *Prepositional Group* class

10   **elif** head POS **in** *nominal POSs* ∧ element type **in** *nominal elemement types***:**

11   assign node *Nominal Group* class

12   **elif** head POS **in** *adverbial POSs* ∧ element type **in** *adverbial element type***:**

13   assign node *Adverbial Group* class

14   **elif** head POS **in** *adjectival POSs* ∧ element type **in** *adjectival element type***:**

15   assign node *Adjectival Group* class

      // creating the rest of the units

16   **if** node *is not a leaf***:**

17   **if** node *is a Clause***:**

18   create the Clause elements for current node

19   **else:**

20   create the head for the current node

21 end

The Algorithm 11 shows how to enrich CG nodes with systemic choices using hard-coded selector functions (in the future, these hard coded selector functions will be replaced by verifying realization statements). The outer loop is a bottom-up iteration over the CG nodes in depth first (DFS) post-order. The inner loop iterates over the networks activated by the focus node. Then a lookup in the Table 6.7 returns the function for choosing features from the system network and eventually a parameter (if the function requires it). This technique, involving a mapping table from SN to functions, is similar to the one in CG creation phase (Algorithm 5). The line 5 executes the selection function returning a set of choices. Then the last line assigns the choices to the node in focus.

Algorithm 11: Enriching CG with systemic features implemented as custom
methods

   input   **:** cg, dg

1 begin

2   **for** node **in** *list of* cg *nodes in DFS postorder***:**

3   **for** network **in** *activated networks for the* node**:**

4   selector function ← the selector function associated to the **network**

5   element type ← if any get the **selector function** parameter

6   systemic choices ← execution result of **selector function** for node, cg
              and **element type**

7   add the **systemic choices** to the **node**

8 end

Key	*System Activation Order*
clause	POLARITY,VOICE, AGENCY, MOOD TYPE, INDICATIVE TYPE, DEICTICITY, TENSE, MODALITY
nominal	PERSON, ANIMACY, GENDER, NUMBER
deictic	DETERMINATION
predeictic	DETERMINATION
thing	PERSON, ANIMACY, GENDER, NUMBER
possessor	PERSON, ANIMACY, GENDER, NUMBER

Table 6.6 System activation table depending on unit class or element type

Currently implemented networks are outlined in Table 6.6 as associations to the
node class or function. The left column represents activation triggers and the right
column represents an ordered list of systems.

System Name	Python Function	Additional Parameter
POLARITY	check_polarity	
VOICE	check_voice	
AGENCY	check_agency	
MOOD TYPE	mood_type	
DEICTICITY	check_deicticity	
TENSE	check_tense	
MODALITY	check_modality	
DETERMINATION	dictionnary_lookup	determination_dict
PERSON	dictionnary_lookup	person_dict
ANIMACY	dictionnary_lookup	animacy_dict
GENDER	dictionnary_lookup	gender_dict
NUMBER	check_number	
MOOD ADJUNCT TYPE	dictionary_lookup	mood_adjunct_dict

Table 6.7 Mapping systemic networks to selection functions

Algorithm 12: Dictionary lookup selector function

input : cg, dg

1 begin

2   **for** node **in** *list of* cg *nodes in DFS postorder*:

3   **for** network **in** *activated networks for the* node:

4   selector function ← the selector function associated to the network

5   element type ← if any get the selector function parameter

6   systemic choices ← execution result of selector function for node, cg
                and element type

7   add the systemic choices to the node

8 end

The *dictionary_lookup* function is a type of *naive backwards induction* (Algorithm 2). It checks whether the word(s) of the *mcg_node* are in a dictionary with preselected features. It is the only function that requires a parameter which is the lookup dictionary. An example of dictionary is presented in the Table 6.8

6.6 Discussion

> * replace hard coded selector functions with constraint checking

Lexical item	Feature
a	partial-non-selective-singular
all	positive-plural
either	partial-singular
that	non-plural

Table 6.8 Dictionary example for the DEICTICITY system network

> assign feature with a certain probability and allow assignments of mutually exclusive features; this would require the next step of more general constraint checking employing many more other sources such as semantic, phonetics, situation etc.

> use of logical notations (OR, XOR, AND sets)

Chapter 7

Transitivity parsing: the semantics

Semantic Role Labeling (SRL) is a well established task in the mainstream computational linguistics. Transitivity analysis is the SFL counterpart for SRL and constitutes the focus of this chapter.

The general approach is to first account for covert syntactic constituents and after assign process types and participant roles according to a given resource. In current case the account for empty constituents is implemented as described in Government and Binding Theory (GBT) (Haegeman 1991) and the employed verb database is called Process Type Database (PTDB) (Neale 2002).

Compared to SRL task, where the majority of implementations use probabilistic models trained on an annotated corpus, I employ a static data base based on which I assign a set of configurations that may be the case, or what are the possibilities, rather than the best single guess. For this reason I use the preparatory step of identifying covert constituents (i.e. Null Elements) which reduces the number of possible assignments taking the analysis close to the goal of a single "correct" configuration.

7.1 Creation of Empty Elements

Sometimes a semantic participant is not mentioned (is elided) in a clause. It happens for one of two reasons: the participant mentions are contextually implicit and usually located outside the clause borders. So when they are implicit the resolution is contextual and required discourse structure awareness. This task is out of the current scope and constitutes a research field on its own.

However, when the participants are missing because they are syntactically recoverable in the sentence then they are inferred from the structure and reference nodes are created for the missing elements. This phenomena are described in GB theory

specifically the *Control and Binding* of empty elements (Haegeman 1991) which has been introduced in Section 4.2. The *reference constituents* are important for increasing the completeness of semantic analysis by making the participants and their afferent label explicit.

This stage is particularly important for semantic role labeling because usually the missing elements are participant roles (theta roles) shaping the semantic configuration. The most frequent are the cases of *control* where the understood subject of a clause is in the parent clause like in examples 94 96 where *Subj* is a pronominal subject placeholder.

(94)   Piotr is considering whether [*Subj* to abandon the investigation].

(95)   Susan promised us [*Subj* to help].

(96)   They told you [*Subj* to support the effort].

There are also movement cases when a clause constituent receives no thematic role in higher clause but one in lower clause. The other case is of the non-overt constituents that are subjects in relative clauses and refer to head of the nominal group. This part of the algorithm is set up to detect cases of *null elements* as described in the Chapter ch:gbt and create placeholder constituents for them which are in the next step enriched with semantic roles.

In Section 4.3 I discuss how to relate GBT to Stanford Dependency Grammar. This section discusses how to relate GBT to Systemic Functional Grammar. Currently the NP traces and PRO subjects are created with a set of graph patterns while the Wh traces are created with an algorithm.

7.1.1   The PRO and NP-Trance Subjects

The *xcomp* relation in DC can be encoded as an CG pattern graph (Figure 7.1) targeting the constituents that are non-finite clauses functioning as complement that have no subject constituent of their own and no "if" and "for" markers (according to generalization 4.2.4). They shall receive a the PRO subject constituent (governed or not) by the parent clause subject.

The generalization 4.3.2 reflects criteria for selecting the controller of PRO based on its proximity in the higher clause. The schematic representation of the pattern for obligatory and subject object control is depicted in Figure 7.2 and respectively 7.3. Please note that in case of Figure 7.3 the prepositional complements do not affect subject control in any way since it specifies only the nominal complements this making it complementary to 7.2 with respect to prepositional complements.
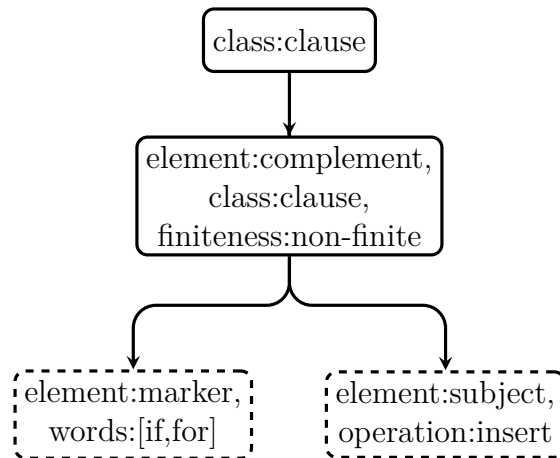
Fig. 7.1 CG pattern for detecting PRO subjects
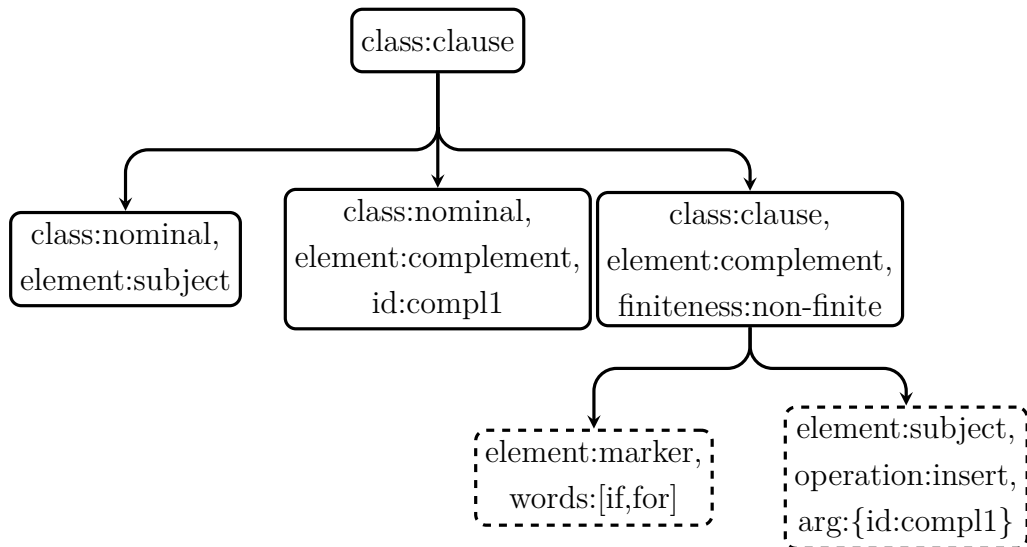
Fig. 7.2 CG pattern for obligatory object control in complement clauses
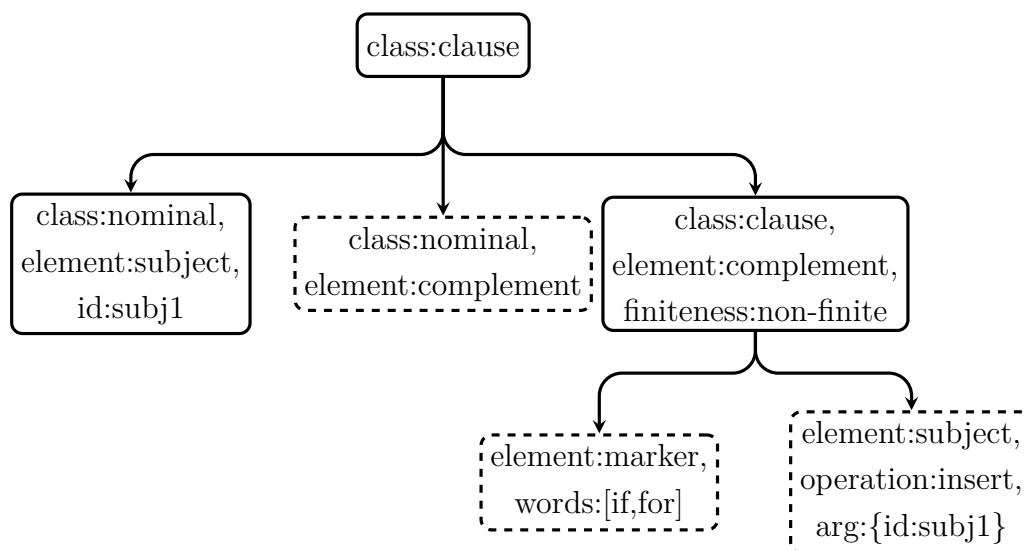
Fig. 7.3 CG pattern for obligatory subject control in complement clauses

In dependency grammar the adjunct clauses are also introduced via *xcomp* and *prepc* relations, so syntactically there is not distinction between the two and patterns from Figures 7.2 and 7.3 are applicable.

According to generalization 4.2.9 the PRO is optionally controlled in subject non-finite clauses. Since it is not possible to bind PRO solely on syntactic grounds in the generalization 4.2.9 is proposed the arbitrary interpretation.
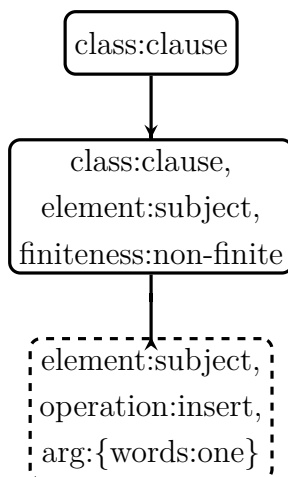
Fig. 7.4 CG pattern for arbitrary control in subject clauses

The pattern for subject control in subject clause is represented in Figure 7.4. This of course is an oversimplification and more rigorous binding rules can be be developed in a future work to cover binding scenarios exemplified in 36-39.

7.1.2 Wh-trances

Creating constituents corresponding to Wh-traces involved a slightly larger number of scenarios and for the pragmatic reasons I have implemented the following algorithm rather than create the set of corresponding graph patterns. Nevertheless in the future this shall be expressed as graph patterns to be consistent with the general approach. The algorithm pseudo-code below shows how it is done.

Algorithm 13: Creating the Wh-traces

input : dg, cg

1 begin

2   **for** element **in** *list of Wh-elements in entire* cg**:**

3   identify the Wh-groups containing the element

4   identify the syntactic function of the Wh-element within the group

5   identify the function of Wh-group in the clause

6   check the number of clauses and which of clauses contains the Wh-group

7   **if** *more than one clause in* cg**:**

8   **for** group **in** cg**:** low to high

9   **if** *Wh-group is* **not** *Subject* ***AND***

10   *Wh-group is* **not** *in lowest embedded clause***:**

11   **if** *Wh-group is Adjunct function***:**

12   create Adjunct Wh-trace using dg and cg

13   **else:**

14   create Theta Wh-trance using dg and cg

15 end

Algorithm 14: Creating the Adjunct (circumstantial) Wh-traces

input : wh-group, dg, cg

1 begin

2   check the tense and modality for all the clauses

3   **for** *clause* **in** cg**:** from the clause of wh-group to lowest

      /* create the adjunct trace in the first clause that has

        non present simple tense   */

4   **if** *clause tense is* **not** *present simple***:**

5   create Adjunct Wh-trace for Wh-group

6   **return**

7 end

Algorithm 15: Creating the Theta (participant) Wh-traces

    input : wh-group, dg, cg

1 **begin**

2   get possible configurations for each clause from the PTDB

       /* check if the higher clause is a projection and and has an

         extra argument   */

3   **foreach** *config in higher clause configurations* **do**

4   **if** *(config is two role cognition **and** config takes expletive subject) **or***

           *(config is three role cognition **and** clause is passive voice)***:**

5   higher is eligible ← True

6   **break**

7   **end**

       /* check if the lower clauses might miss an argument   */

8   **foreach** *clause in lower clauses* **do**

9   **foreach** *config in clause configurations* **do**

10   **if** *number of clause theta constituents < number of config arguments***:**

11   lower is eligible ← True

12   break

13   **end**

14   **end**

15   **if** *higher is eligible **and** lower is eligible***:**

16   **if** *higher clause has "that" complementizer***:**

17   create Object Wh-trace in the lowest clause

18   **else:**

19   **if** *Wh-group has case***:**

20   **if** *Wh-group case is nominative(subjective)***:**

21   create Subject Wh-trace in lowest clause

22   **else:**

23   create Object Wh-trace in lowest clause

24   **else:**

25   create Wh-trace with Subject function and attempt to assign

           theta roles

26   **if** *theta roles **not** successfully assigned in lower clause***:**

27   change the Wh-trace to Object function and assign theta roles

28 **end**

---

cuss the wh trace
eation algorithm

After the null elements are created the constituency graph is ready for the semantic enrichment stage semantic configuration are assigned to each clause. It is described in the next section.

7.2 Semantic enrichment stage

In this section is explained how the parser assigns Transitivity configurations to the constituency graph. Transitivity roughly corresponds to what is known in computational linguistics as semantic analysis of text or *semantic role labeling*. In this task the clause is assigned a semantic frame called configuration in which predicate functions as the process and the participants take frame dependent roles (or functions). The nodes that do not receive any participant role are the adjuncts which act as circumstances.

Because the proposed task goes beyond the syntactic structure it needs to rely on additional external semantic resources. One such resource is the Process Type Database(PTDB) created by Neale (2002). It is a table which listing possible configurations of semantic roles for each verb sense for over five thousands most popular verbs in English. This resource is then integrated into the current parser pipeline to automatically assignment semantic configurations and participant soles. How this is done is covered in the

In the following I will explain the the practical steps of generating Transitivity but not before introducing the PTDB. Then I explain how the automation of the semantic role labeling is performed and why it is important to identify and create references to empty elements.

7.2.1 The Process Type Database

The Process Type Database (PTDB) (Neale 2002) is the key resource in the automatization of Transitivity Analysis. It is the source for creating a set of graph patterns which are then used to enrich the constituency graph as described in the Section 6.5.

PTDB provides information on what possible process types and participants can correspond to a particular verb meaning. The PTDB is a dictionary-like dataset of verbs bound to an exhaustive list of verb senses and the corresponding Process Configuration for each of them.

In her work on PTDB Neale (2002) improved the TRANSITIVITY system of the Cardiff Grammar by systematizing over 5400 senses (and process configurations) for

2750 most popular English verbs. Table 7.1 presents a simplified sample of PTDB content.

verb form	informal meaning	process type	configuration
calcu-late	work out by mathematics (commission will then,calculate the number of casted votes)	cognition	Ag-Cog + Ph
	plan (newspaper articles were calculated to sway reader's opinions)	two role action	Ag + Cre
catch	run after and seize (a leopard unable to catch its normal prey)	possessive	Ag-Ca + Af-Pos
	fall ill (did you catch a cold?)	possessive	Ag-Ca + Af-Pos
catch (up with)	reach (Simon tried to catch up with others)	two role action	Ag + Ra

Table 7.1 An example of records ins PTDB

7.2.2 Cleaning up the PTDB

The original version of the PTDB available on Neale's personal page is not usable for computational purposes as such. It contains records applying a couple of different notation notations and sometimes informal, human friendly comments which represent noise for the computer programs and cannot be processed as such. In this section I explain now how the original PTDB was transformed in order to be used as a parsing resource.

I focus on three columns which are of interest for the parsing purpose: the *verb form* ($1\mathsf{sp}(st)$), the Cardiff grammar *process type* ($6\mathsf{sp}(th)$) and the participant role *configuration* ($8\mathsf{sp}(th)$) columns. Note that column numbers correspond to the original PTDB structure.

After the transformation the PTDB column descriptions are as described in the Table 7.2.

Next I describe the transformed PTDB and how it be interpreted. For a start, the verb form column contains either base form of the verb (e.g. draw, take), base form plus a preposition (e.g. draw into, draw away, take apart, take away from) or base form plus a phraseologic expression (e.g draw to an end, take on board, take the view that, take a shower). The prepositions are either the verbal particles or the preposition introducing the prepositional phrase complement. Prepositions often influence the

Column	Original	Modified
1/A	Form	Form
2/B	*Occurences of form*	*Occurences of form*
3/C	COB class (& figure where possible)	COB class (& figure where possible)
4/D	meaning descrition	meaning descrition
5/E	Occurences in 5 million words	Occurences in 5 million words
6/F	*Cardiff Grammar feature*	*Cardiff Grammar process type (reindexed/renamed)*
7/G	Levin Feature	*Cardiff participant feature*
8/H	*Participant Role Configuration*	*Cardiff participant feature (extra)*
9/I	Notes	Levin feature
10/J		*Participant Role Configuration*
11/K		Notes

Table 7.2 The table structure of PTDB before and after the transformation

process type and the participant configuration. So they are good cues to be considered during the semantic role assignment. The verb forms that have the same process type and configuration but different prepositions often are grouped together delimited by a slash "/" (e.g. draw into/around, take off/on) or if optional (i.e. coincide with the meaning of the verb base form without any preposition) they are placed in round brackets "()" (e.g. flow (into/out/down)).

The process type column registers one feature in the PROCESS-TYPE systemic network depicted in Figure 7.5.

The participant configuration column contains the sequence of participant type abbreviations joined by plus sign "+" (e.g. Ag + Af, Em + Ph, Ag-Cog + Ph). The order of participants corresponds to the Active voice in Declarative mood also called the *canonical form of a configuration*. Originally the configurations contained the "Pro" abbreviation signifying the place of the main verb/process. As all configurations are in canonical form, the Pro was redundant occurring always in the second position thus had been removed. So the first participant corresponds to the Subject, second to the first complement and third to the second complement. Some participants are optional for the meaning and are marked round brackets "()" (e.g. Ag + Af-Ca (+ Des)) meaning that the participant may or may not be realized.

Not all the records in the original resource fulfill the description above and needed corrections. For example when Neale had doubts during the making of PTDB, she

marked uncertainties with question mark "?". Commas "," and and "&" sign are use with various meaning and inconsistently in all columns. Comments such as "not in Cob" were encountered across several columns.

Some records contain only prepositions listed in the verb form column which actually represent omissions of the main verb which is to be found in the immediately preceding records(s), this have been fixed by pre-pending the verb form to the preposition.

Among the identified verb meanings in PTDB, there are some that do not contain configurations. These records missing a process type and configuration had not been suppressed.

The process type feature column contained originally a second feature which had been removed, representing a compressed version of the participant configuration however it was redundant as the full configuration is registered in the next column.

In PTDB Neale uses a slightly different process type names than the ones adopted in this work. The process type features have been re-indexed and adapted to match exactly the feature labels in the PROCESS-TYPE systemic network (e.g. "one role action" became "one-role-action"). Annexe .2.5 provides the mapping across the process type versions.

Configuration column is one of the most important ones in PTDB. Checking it's consistency conform to Fawcett's Transitivity system revealed the need for some corrections. For example "Af + Af", "Af-Ca + Pos + Ag", "Af-Cog + Ph + Ag" are grammatically impossible configurations and were manually corrected to the closest likely configuration "Ag + Af", "Ag + Af-Ca + Pos", "Ag + Af-Cog + Ph".

Other records, judged by the process type, were incomplete. For example instances of two role action registered only one of the role e.g. Af or Ca omitting the Ag participant. These records have been also manually corrected by perpending the Ag, Ag-Af or Cog roles depending on the case.

The "Dir" participant is interpreted as direction is not registered/defined in the Cardiff Grammar. Nevertheless there is a "Des" participant which I believe is the closest match. Therefore all "Dir" occurrences had been changed to "Des". One may argue that the two have different meanings however grammatically they seem to behave the same (at least in the accounted configurations).

By contrast some process types had been changed from Action into either Locational or Directional because they contained either Loc (location), Des (destination) or So (source) participants which are not found in Action processes unless they function as Adjuncts which are out of the context of the current description.

A note worth making here is that the Locational, Movement specifically but also other spatial verbs are very difficult to assign roles correctly because their participants are Locations, Directions, Destinations etc. which can also serve as spatial adjuncts. This aspect has not been directly addressed in present work and is reflected by a lower accuracy for these classes of verbs.
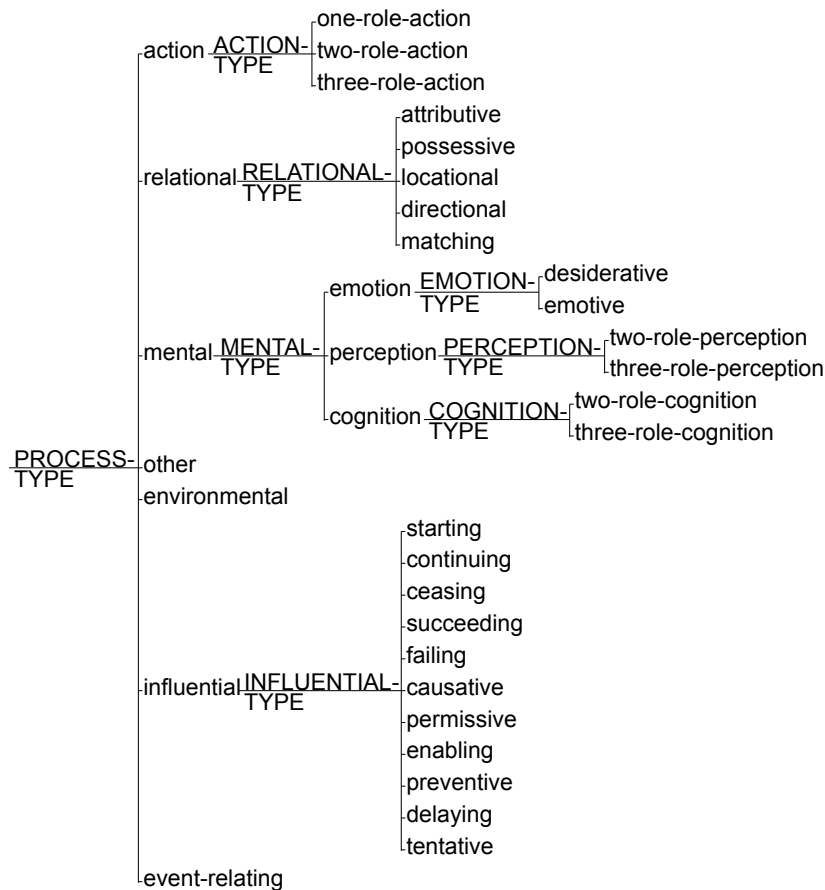
Fig. 7.5 Transitivity System in Cardiff Grammar

The Cardiff features column indicates the process type selected in the TRANSITIVITY system corresponding to one of the top levels depicted in Figure 7.5

7.2.3   Generation of the Configuration Graph Patterns

The configuration pattern graphs are used for CG enrichment executed through graph matching operation described in Section 5.6. The PTDB has been cleaned up and normalized to support automatic generation of the *Configuration Graph Patterns* (CGP). These graph patterns represent a constrained syntactic structure that carry

semantic features. The latter are to be applied if the graph pattern is identified the sentence CG.

CGPs are generated from the process type and participant configuration columns of the PTDB. Figure 7.6 depicts the prototypical template for generating three role CGP for canonic participant order (indicative mood active voice). This part assigns the Clause constituent Process Type (i.e. configuration type) while Subject and Complement constituents receive participant roles.
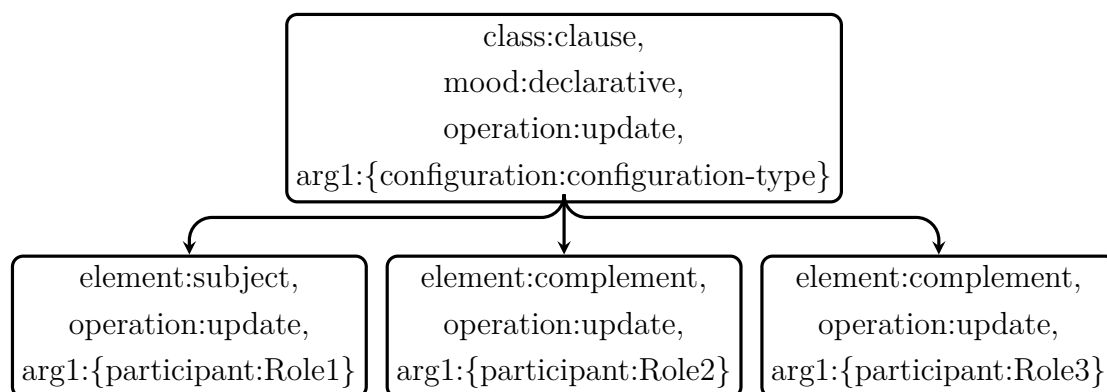
Fig. 7.6 Indicative mood and active voice configuration pattern with three participant roles

Besides the canonic CGP a set of variations are generated for each configurations by transforming the canonic configuration. The variations are function of the process type, participant roles, mood and voice. When each of the variants are supported by the process type and participant configuration the flowing CGP are generated: (1) the declarative active (2) the passive (3) the imperative and (4) Wh-interrogative (Wh-Subj/Wh-obj/Wh-adj especially important for locational and directions processes).

If the configuration accepts passive voice i.e. the first configuration role is not expletive "there" or pleonastic "it" and the last role is not Agent role then both active and passive voice CPG are generated.

The imperative form CGP is generated if the first role of the configuration implies an active animate entity. Roles that accept imperative form are: Agent, Emoter, Cognizant, Perceiver and their compositional derivations (e.g. Agent-Carrier, Agent-Cognizant, Affected-Emoter etc.)

The passive differs from active voice pattern by switching places of the first two roles resulting in the second role matched to the subject function and the first role one the first complement. In the case of imperative the first role as well as the subject constituent are simply omitted.

Algorithm 16 outlines how the CPGs are generated from the PTDB. The CPGs are represented as Python structures and are stored in a Python module. This way the graph patterns are accessible as native structures making it handy to instantiate the graph patterns.

Algorithm 16: Generating the CPGs from the PTDB

    input : PTDB

1 **begin**

2   generate unique set of process type + participant roles

3   generate unique set os process type

4   **for** *process type* **in** *possible process type***:**

5   generate configuration set for given process type

6   **for** *configuration* **in** *configuration set***:**

7   generate indicative active pattern graph

8   **if** *no expletive in configuration***:**

9   **if** *configuration accepts passive***:**

10   generate indicative passive pattern graph

11   **if** *configuration accepts imperative***:**

12   generate imperative pattern graph

13   **if** *Locational process***:**

14   generate expletive there pattern graph

15   **if** *configuration participants may function as Adjuncts***:**

                     `/* the Directional processes varying optional`
                             `Source, Path and Destination        */`

16   generate variate role indicative active pattern graphs

17   generate variate role indicative passive pattern graphs

18   generate variate role imperative pattern graphs

19   generate variate role Wh-interrogative pattern graphs

20 **end**

The first two lines of the algorithm synthesize the PTDB by grouping unique configurations for each process type. Then for each configuration of each process type is generated one to three pattern graphs depending on the configuration and process type specifics.
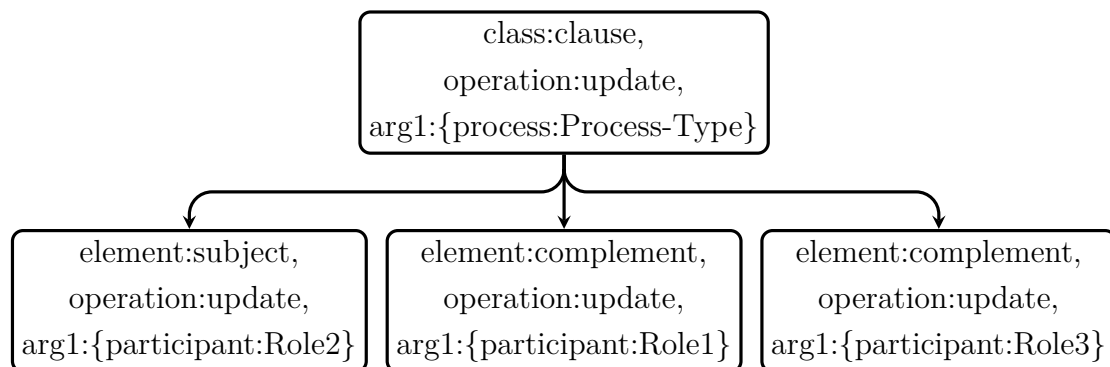
Fig. 7.7 Indicative mood and passive voice configuration pattern with three participant roles
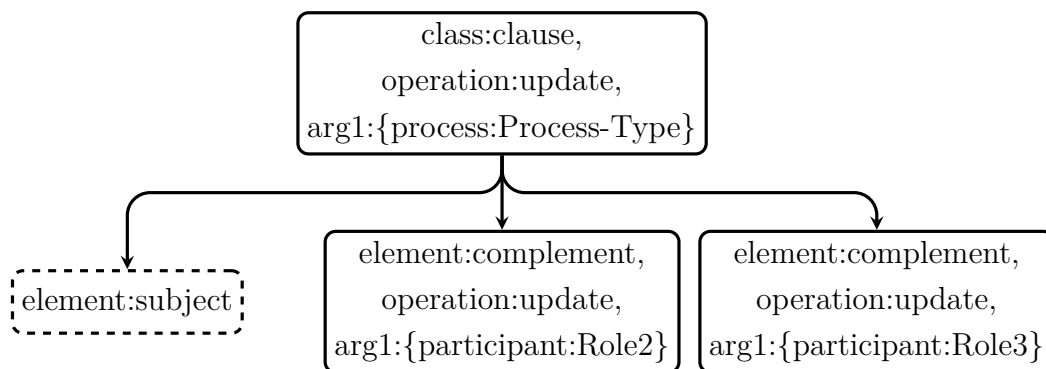
Fig. 7.8 Imperative mood configuration pattern with three participant roles

The indicative mood active voice pattern (depicted in Figure 7.6) corresponding to the canonic form in PTDB is always generated. If the configuration does not contain an expletive and accepts passive voice then the corresponding pattern is generated with first and second roles switched like in Figure 7.7. If the configuration accepts imperatives then also a subjectless pattern graph is generated with first role omitted as depicted in Figure 7.8.

Directional processes are rather a special case. Examples 97 to 99 are equally valid configurations. Example 100 is a generic representation highlighting the optionality of these participants.

(97)   The parcel traveled from London[So].

(98)   The parcel traveled via Poland[Pa].

(99)   The parcel traveled to Moscow[Des].

(100)   The parcel traveled (from London[So]) (via Poland[Pa]) (to Moscow[Des]).

So in Directional configurations second, third and fourth participants are optional and may occur in any order but at least one of them should be present. Therefore So, Pa and Des participants patterns should be generated for all combinations as presented in the Table 7.3 below.

So	Pa	Des
+	-	-
-	+	-
-	-	+
+	+	-
+	-	+
-	+	+
+	+	+

Table 7.3 Participant arrangements for Directional processes (order independent)

Finally, the CPGs are grouped by process type. This alleviates the burden of selecting the number of patterns to test for a certain clause. The python structure looks as represented in Listing 7.1

Listing 7.1 Configuration Pattern Set as a Python dictionary structure

```python
configuration_pattern_set = {
        process_type1 : {
                config1 : [pattern1, panttern2,...],
                config2 : [pattern3, panttern4,...],
        },
        process_type2 : {
                config3 : [pattern5, panttern6,...],
                config4 : [pattern7, panttern8,...],
        }
        ...
}
```

Role	Prepositions
Des	to,towards,at,on,in,
Ben	to,for,
Attr	as,
Ra	on,in,
So	from,
Pa	through,via,
Loc	in,at,into,behind,in front of, on
Mtch	with,to,
Ag	by,
Ph	about,
Cog	to

Table 7.4 Prepositional constraints on participant roles

7.2.4 Transitivity parsing algorithm

Transitivity enrichment is performed on the constituency graph after it has been enriched with Mood features and the null elements had been identified and appended.

Algorithm 17: Transitivity parsing

    input : cg, dg

1 begin

2   **for** *clause* **in** *clauses in mcg***:**

3   select from PTDB candidate process types and configurations

4   filter configurations fitting the clause

5   **for** *config* **in** *valid possible configurations***:**

6   filter pre-generated pattern graphs of the config fitting the clause

7   **for** *pattern* **in** *filtered pattern graph set***:**

8   enrich clause using pattern and mcg filtered by role constraints

9 end

Algorithm 17 outlines the semantic parsing process implemented in the current parser which is a cascade of three loops.

The first loop iterates through clauses in the mood constituency graph and for each the candidate process types are identified by considering: (a) the main verb, (b) the prepositions connected to it (either prepositional particles, or prepositions introducing a complement or adjunct prepositional phrase listed in Table 7.4) or (c) phrasal expressions such as "take a shower" which are explained in Section 7.2.1.

The second loop iterates through the candidate configurations for each candidate process type and selects the graph patterns that shall be matched to the current clause.

Then iteratively, each of the retrieved graph patterns (in the third loop) are applied to the clause graph. Line 7 enriches CG nodes with new features of the pattern graph each time they are successfully matched. Before enrichment the CG nodes are checked against an additional set of condition (captured by Algorithm 18) which were omitted in the pattern graph. These conditions may prevent the enrichment even if the pattern has been identified.

Algorithm 18: Participant Role constraint check if a role is not illegal for constituent

---

    input : node, role, dg

1 begin

    /* Cog, Em and Perc must be animates   */

2   **if** *role is Cog* **or** *Em* **or** *Perc***:**

3   check that the node has animate feature

    /* clauses can only be phenomenas   */

4   **if** node *is a clause***:**

5   check that role is Ph only

    /* prepositional phrases can only start with certain prepositions for a role   */

6   **if** node *is a Prepositional Phrase***:**

7   get the list of allowed prepositions for the role

8   check if the prepositional phrase starts with any of the allowed prepositions

9 end

The Transitivity parsing results in multiple process configurations being assigned to clauses introducing uncertainty. This uncertainty, laying within the reasonable limits, is not dealt with by the current parser but shall be further resolved through deeper semantic and pragmatic means.

7.3 Discussion

This chapter describes how the semantic role labeling from Transitivity system network is performed on top of the constituency graph.

First null elements are identified and filled with proxy constituents. This process ensures higher accuracy of semantic role assignments from the configuration patterns in the second step.

The configuration patterns have been generated from the PTDB - a verb database accounting for Transitivity patterns in Cardiff grammar. In order to generate these

patterns the PTDB had to be first cleaned up, unified and aligned to the present Transitivity system. Then for each configuration were generated a set of possible patterns varying based on mood, voice, process type and participants.

Finally the semantic role labeling step employs the same pattern based enrichment mechanism as the one used in Section 6.5

The Algorithms can be improved a great deal by externalization of constraints and conditions. Some of them however are too complex to check to be completely removed but in the next iterations the tendency should definitely be towards higher abstraction and separating the grammatic constraint as realization rules from the code.

Chapter 8

The Empirical Evaluation

8.1 Evaluation settings

There are several well established annotated corpora such as Penn or Prague Tree-Banks for phrase structure, dependency and other grammars that serve as training and evaluation data sets. To date I am not aware of any open corpus annotated with Cardiff or Sydney grammars that would play the role of the gold standard for SFL parsers.

To overcome this problem I decided to create my own evaluation corpus. A part of it was created together with Ela Oren covering basic Sydney Mood constituency structure. Another part is the PhD dataset of Anke Shultz covering Cardiff Transitivity analysis. The current evaluation is based on these two datasets.

The parser is evaluated by comparing manually annotated text to automatically generated analyses. The datasets have not been developed for the purpose of evaluating current parser however they are compatible and can be adapted to evaluate identification of constituent boundaries and how is being assigned a limited set of features. The first dataset represents Transitivity analysis of 31 files spanning over 1466 clauses and 20864 words created by Anke Schultz as part of her PhD project and is mainly used to evaluate semantic parsing. The second dataset represents Constituency and Mood analysis of blog text spanning over 988 clauses and 8605 words. This dataset was created by Ela Oren as a part of her project on analysing texts of people with obsessive compulsive disorder (OCD) and is mainly used to evaluate the syntactic parsing.

Both datasets were created with UAM Corpus Tool authored by O'Donnell (2008a,b). It stores annotations as segments with their corresponding start and end positions in the text together with set of features(selected from a systemic network) attributed to

that segment without any proper constituency relations among segments. Below is a XML representation of an annotation segment for Transitivity.

```
<segment id="4" start="20" end="27" features="configuration;
 relational; attributive" state="active"/>
```

To make manual and automatic analyses comparable constituency graphs (CG) are transformed into a set of segments the evaluation task being reduced to find matches or near-matches between two segment bundles. This task is almost the same as know problem in computer science called Stable Marriage Problem. Because the problem formulation slightly differs from the standard one I implement Gale-Shapley algorithm (Gale & Shapley 1962) one of the most efficient algorithms with the corresponding extra requirements.

The standard stable marriage problem is formulated as 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 every individual expressed preferences as an ordering of individuals from the opposite group from the most preferred one to the least preferred one. Thus the preferences must be *complete* and *fully ordered*. None of the last two constraints could be fulfilled in the context of the present evaluation and I explain why by exploring and explaining the identity criteria between two segments.

Leaving aside the features and focusing only on the segment span we can say that two segments are identical when their start and end indexes are identical. However it is not always the case and there are situations when we would like to consider two segments identical even if their spans differ.

Firstly, there is an encoding problem. Some text files start with a BOM or contain non printable characters resulting in a technical problem of mismatch between how UAM Coprus Tool reads files and how the parser evaluation module reads them introducing two types of text-segment mismatch: (a) shifted text and (b) shrieked/enlarged text span. This problem has been overcome by an extra module which indexes MCG words and sentences (as word collections) in the raw text. This helps re-indexing the parse segments in such a way that their text "coincides" with the text in the CT annotations and readjusting their indexes.

Secondly, there is a feature comparison problem. Provided that two segments have identical spans and textual content but two different sets of features. To overcome this problem I force segments to carry only one feature. The consequence are multiple segments with the same span (Figure 8.2) for each feature instead of single segment with multiple features (Figure 8.1).
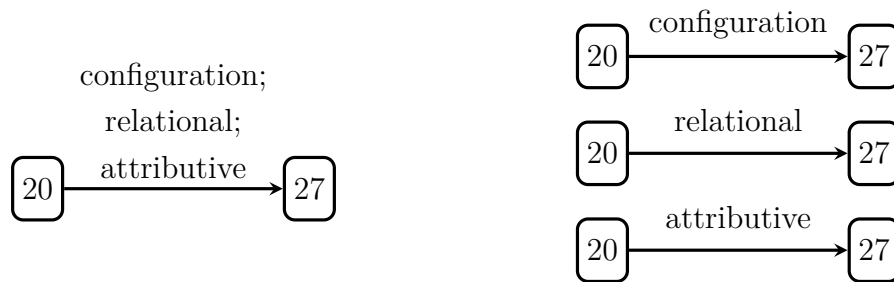
Fig. 8.2 A set of segments with single features

When each segment contains exactly one feature the evaluation can be focused on one or a set of features of interest by selecting segments that contain exactly those.

8.2 Syntactic Evaluation

The syntactic evaluation is based on the corpus produced together with Ela Oren focused on Constituency and clause Mood analysis. The text represents a blog article of a person diagnosed with OCD writing about the challenge of overcoming OCG.

I first present few differences in how corpus has been annotated as compared to parser output, then present segmentation statistics followed by featured statistics.

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 usually at the end of each segment. Neither the conjunctions (and, but, so , etc.) are included into the corpus segments because they are considered markers in the clause/group complexes. The parser, on the other hand, includes them into the segments. Moreover the conjunct spans differ as well. Instead of being annotated in parallel segments as represented in Figure 8.3 they are subsumed in a cascade from the former to the latter as depicted in Figure 8.4.
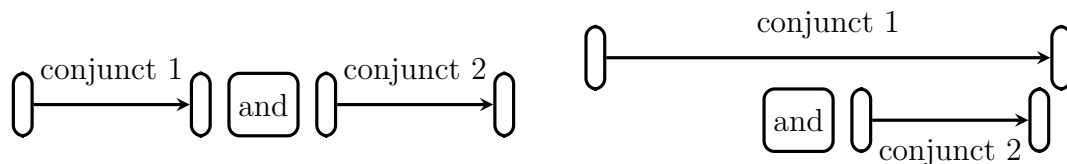
Fig. 8.4 Conjuncts annotated as subsumed segments

Another difference is the way verbs and verb groups are handled. In the parser is implemented the Sydney Grammar's Finite and Predicate constituents while the corpus is annotated according to Cardiff Grammar with Main Verbs, Finite, Auxiliary, Negation Particles etc. as explained in Section 2.5 and Appendix 9.6.

The above described differences in annotation rules are inevitably reflected in segmentation differences. Let's compare now the corpus and parser segmentation. I use geometric distance ($d = \sqrt{\Delta start \mathsf{sp}(2) + \Delta end \mathsf{sp}(2)}$) to measure the difference between the segments matched with Gale-Shapley algorithm. Figure 8.5 represents distribution of matches according to the distance. 71.11% of the segments have identical spans while the rest of 28.8% have minor to major differences of which 19% are distanced 1-5 characters due to differences in (a) punctuation, (b) conjunction and (c)verbal group treatment described above. The rest 9.9% of long and very long distances (6-182 characters) are due to 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).
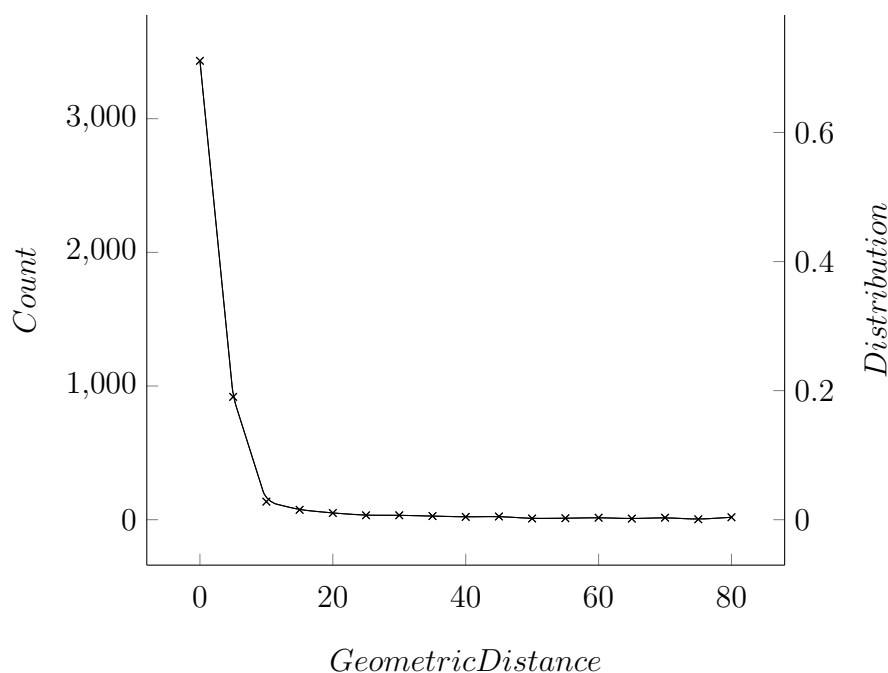
Fig. 8.5 Segment distance distribution

Next I present evaluation data for systemic features and systems overall annotated in the corpus and parser. In the tables below are presented the following statistics for features and systems (in upper caps): number of segments in corpus containing the feature (M/N) percentage of the in corpus (M/%), number of segments generated

by the parser (A/N) and corresponding percentage (A/%), precision, recall and F_1 measures.

Table 8.1 contains evaluation statistics for the rank system. In this evaluation only group and clause level features have been taken into account excluding those at the word rank which are missing in the corpus. Clause and group constituents are identified with 0.88 and 0.9 F_1 scores.

The marker feature is used for functional words such as prepositions, verbal prepositional particles, coordinating and subordinations conjunctions. However the prepositional phrases in the corpus do not contain the marker segment as the parser generates and only conjunctions and verbal particles are marked leading to low precision of 0.38. Overall the rank constituents are identified with 0.82 F_1 score.

Name	M/N	M/%	A/N	A/%	Prec	Rec	F_1
RANK	2455	26.922	4052	36.156	0.716	0.902	0.762
clause	529	21.548	545	13.450	0.875	0.902	0.888
group	1699	69.206	1760	43.435	0.886	0.918	0.902
marker	224	9.124	396	9.773	0.389	0.688	0.497
word	3	0.122	1351	33.342	-	-	-

Table 8.1 Rank system evaluation statistics

The clause elements are systematized in Interpersonal function system whose evaluation statistics are contained in the Table 8.2. Subjects and Predicators have the highest F_1 scores of 0.889 and 0.888. Indirect objects have been perfectly identified but they are very few and do not have a statistical significance. Adjuncts and Complements (direct objects) have a lower precision and higher recall scoring 0.628 and 0.661 F_1. This is in part due to the fact that the parser annotates prepositional phrases that follow a predicate as both Complement and Adjunct being unable to syntactically distinguish between the two and this task being overtaken during the semantic analysis. The Finite elements are distinguished in the corpus only when they stand alone thus are not conflated with the predicator as in Example 101. But when the finite is conflated with the Predicator as in Example 102 it has not been annotated in the corpus but it should be and further corrections of the corpus shall be implemented. For this reason there is a big difference between precision and recall 0.172 and 0.980 because most Finites in corpus have been identified by the parser and most of them are missing from the corpus.

(101)   He may (Finite) go (Predicator) home latter.

(102)   He already went (Finite/Predicator) home.

Name	M/N	M/%	A/N	A/%	Prec	Rec	F_1
INTERPERSONAL-FUNCTION	1681	18.434	2793	24.922	0.659	0.910	0.726
finite	150	8.923	855	30.612	0.172	0.980	0.293
adjunct	266	15.824	358	12.818	0.547	0.737	0.628
subject	389	23.141	439	15.718	0.838	0.946	0.889
complement-direct	347	20.642	594	21.267	0.524	0.896	0.661
complement-indirect	2	0.119	2	0.072	1	1	1
predicator	527	31.350	545	19.513	0.873	0.903	0.888

Table 8.2 Mood clause elements evaluation statistics

In Table 8.3 are presented evaluation statistics for Group Type system meaning how well are identified grammatical classes at the group level. Nominal and Verbal groups are parsed with 0.9 precision followed by prepositional, adverbial and adjectival groups with 0.72, 0.6 and 0.55 scores.

Name	M/N	M/%	A/N	A/%	Prec	Rec	F_1
GROUP-TYPE	1698	18.620	1728	15.419	0.838	0.853	0.845
nominal-group	648	38.163	622	35.995	0.905	0.869	0.887
verbal-group	678	39.929	705	40.799	0.894	0.929	0.911
prepositional-group	120	7.067	124	7.176	0.726	0.750	0.738
adverbial-group	187	11.013	218	12.616	0.606	0.706	0.652
adjectival-group	65	3.828	59	3.414	0.559	0.508	0.532

Table 8.3 Evaluation statistics for group types

8.3   Semantic Evaluation

For the semantic evaluation has been used a corpus created by Anke Schultz for her PhD project annotated with Transitivity features 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 8.4 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 8.5 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.

8.4   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 9

Conclusions

9.1 The thesis summary

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 2 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 3. The practical aspects and concrete algorithms are described in Chapter 6 together with the mapping rules used in the process.

To make clear what are the basic ingredients and how the algorithms are cooked, Chapter 5 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 5. Then in Chapters 6 and 7 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 4 and then a concrete implementation described in Chapter 7.

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 8.4 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.

9.2   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.

9.3   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.

9.4   Further work

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.

9.4.1   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 2.5.1) 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 9.1 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 9.2 Cardiff sample analysis of a clause *embedded* into another

Check the sample analyses in Table 9.1 and 9.2. 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 9.1.

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 9.4.1 (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 9.4.1. 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 9.1 and 9.2 – but also of the modal assessment that includes modality, as demonstrated by the Examples 103 and 104.

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

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

Examples 103 and 104 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*.

9.4.2   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 2.3 and discussed in 2.5.

For instance, Nominal class structure proposed in Cardiff grammar (discussed in Section 2.5.3), 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 2.5.4) 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.

9.4.3 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.

As presented in Appendix **??** I have created a database of 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.

9.4.4 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.

9.4.5 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.

9.4.6 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.

9.4.7 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.

9.4.8 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.

propose a natural
guage task classific
based on the numb
features needed as
and provided as ou

9.5 Overall evaluations

A deduction ma
on the basis of t
main body (i.e. 0
cluding statemen

The writer's perso
opinion on wha
has been discuss

9.6 A final word

half to one page

References

Abney, S. 1987. *The English noun phrase in its sentential aspects*. MIT Press.

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

Bach, Emmon. 1966. *An introduction to transformational grammars*. Holt, Rinehart and Winston. Inc.

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. 2008. Systemic-Functional Linguistics and the Notion of Linguistic Structure: Unanswered Questions, New Possibilities. In Jonathan J. Webster (ed.), *Meaning in context: Implementing intelligent applications of language studies*, 24–58. Continuum.

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>.

Brensan, J. 1982. Control and complementation. *Linguistic Inquiry* 13(3). 343–434.

Bresnan, Joan. 2001. *Lexical-Functional Syntax*. Blackwell. <http://books.google.lu/books/about/Lexical{_}Functional{_}Syntax.html?id=vMxgevXoq{_}gC{&}redir{_}esc=y>.

Bresnan, Joan. 2010. *Lexical-functional syntax*. Wiley.

Butler, Christopher. 1985. *Systemic linguistics: Theory and applications*. Batsford Academic and Educational.

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

Carreras, Xavier & Lluís Màrquez. 2005. Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of the ninth conference on computational natural language learning* CONLL '05, 152–164. Stroudsburg, PA, USA: Association for Computational Linguistics. <http://dl.acm.org/citation.cfm?id=1706543.1706571>.

Carroll, John, Guido Minnen & Ted Briscoe. 1999. Corpus Annotation for Parser Evaluation. In *In proceedings of the eacl workshop on linguistically interpreted corpora (linc)* June, 7. Association for Computational Linguistics. <http://arxiv.org/abs/cs/9907013>.

Cer, Daniel D.M., Marie-Catherine M.C. De Marneffe, Daniel Jurafsky & C.D. Manning. 2010. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *Lrec 2010*, vol. 0, European Language Resources Association. <http://171.64.67.140/pubs/lrecstanforddeps{_}final{_}final.pdfhttp://www.lrec-conf.org/proceedings/lrec2010/pdf/730{_}Paper.pdf>.

Chomsky, Noam. 1957. *Syntactic Structures.* Mouton & Co.

Chomsky, Noam. 1981. *Lectures on Government and Binding.* Foris.

Cordella, L P, P Foggia, C Sansone & M Vento. 2001. An improved algorithm for matching large graphs. *3rd IAPRTC15 workshop on graphbased representations in pattern recognition* 219(2). 149–159. doi:10.1.1.101.5342. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.5342{&}rep=rep1{&}type=pdf>.

Cordella, L P, P Foggia, C Sansone & M Vento. 2004. A (sub)graph isomorphism algorithm for matching large graphs. doi:10.1109/TPAMI.2004.75. <http://www.ncbi.nlm.nih.gov/pubmed/15641723>.

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. 2008. *Invitation to Systemic Functional Linguistics through the Cardiff Grammar.* Equinox Publishing Ltd.

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

Fillmore, Charles J. 1985. Frames and the semantics of understanding. *Quaderni di Semantica* 6(2). 222–254. <http://scholar.google.it/scholar?q=fillmore{&}hl=it{&}btnG=Cerca{&}lr={#}5>.

Firth, J.R. 1957. A synopsis of linguistic theory 1930-1955. *Studies in linguistic analysis* 1–32. <http://www.bibsonomy.org/bibtex/25b0a766713221356e0a5b4cc2023b86a/glanebridge>.

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>.

Haegeman, Liliane. 1991. *Introduction to Government and Binding Theory*, vol. 2. Blackwell.

Halliday, Michael A. K. 1966. Some notes on 'deep' grammar. *Journal of Linguistics* 2(1). 57–67.

Halliday, Michael A. K. 1996. On grammar and grammatics. In Ruqaiya Hasan, Carmel Cloran & David Butt (eds.), *Functional descriptions – theory in practice* Current Issues in Linguistic Theory, 1–38. Amsterdam: Benjamins.

Halliday, Michael A. K. 2003a. 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. 2002. Categories of the theory of grammar. In Jonathan Webster (ed.), *On grammar (volume 1)*, 442. Continuum.

Halliday, Michael A.K. 2003b. On the "architecture" of human language. In Jonathan Webster (ed.), *On language and linguistics*, vol. 3 Collected Works of M. A. K. Halliday, 1–32. Continuum.

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

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

Hasan, Ruqaiya. 2014. The grammarian's dream: lexis as most delicate grammar. In Jonathan Webster (ed.), *Describing language form and function*, vol. 5 Collected Works of Ruqaiya Hasan, chap. 6. Equinox Publishing Ltd.

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>.

Hudson, Richard. 2010. *An Introduction to Word Grammar*. Cambridge University Press.

Hutchins, W John. 1999. Retrospect and prospect in computer-based translation. In *Proceedings of mt summit vii "mt in the great translation era"* September, 30–44. AAMT.

Kasper, Robert. 1988. An Experimental Parser for Systemic Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, .

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

King, Tracy Holloway & Richard Crouch. 2003. The PARC 700 Dependency Bank. In *4th international workshop on linguistically interpreted corpora (linc03)*, <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.3277>.

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.

Koeva, Svetla, Borislav Rizov, Ekaterina Dimitrova, Tarpomanova Tsvetana, Rositsa Dekova, Ivelina Stoyanova, Svetlozara Leseva, Hristina Kukova & Angel Genov. 2012. Bulgarian-english sentence-and clause-aligned corpus. In *Proceedings of the second workshop on annotation of corpora for research in the humanities (acrh-2)*, 51–62.

Kucera, Henry & W. Nelson Francis. 1968. Computational Analysis of Present-Day American English. *American Documentation* 19(4). 419. doi:10.2307/302397. <http://search.ebscohost.com/login.aspx?direct=true{&}db=bth{&}AN=16865479{&}login.asp{&}site=ehost-live>.

Lee, Jinsoo, Wook-Shin Han, Romans Kasperovics & Jeong-Hoon Lee. 2013. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proceedings of the 39th international conference on Very Large Data Bases* 133–144. doi:10.14778/2535568.2448946. <http://dl.acm.org/citation.cfm?id=2448936.2448946>.

Lemke, Jay L. 1993. Discourse, dynamics, and social change. *Cultural Dynamics* 6(1-2). 243–276.

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.

Marcus, Mitchell P, Beatrice Santorini & Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2). 313–330. doi:10.1162/coli.2010.36.1.36100. <http://portal.acm.org/citation.cfm?id=972470.972475>.

Marneffe, Marie-Catherine, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre & Christopher D. Manning. 2014. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the ninth international conference*

on language resources and evaluation (lrec-2014)(vol. 14), European Language Resources Association (ELRA). <http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062{_}Paper.pdf>.

Marneffe, Marie-Catherine, Bill MacCartney & Christopher D Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *Lrec 2006*, vol. 6 3, 449–454. Stanford University. <http://nlp.stanford.edu/manning/papers/LREC{_}2.pdf>.

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

Marneffe, Marie-Catherine & Christopher D. Manning. 2008b. The Stanford typed dependencies representation. *Coling 2008 Proceedings of the workshop on CrossFramework and CrossDomain Parser Evaluation CrossParser 08* 1(ii). 1–8. doi:10.3115/1608858.1608859. <http://portal.acm.org/citation.cfm?doid=1608858.1608859>.

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.

McCarthy, John, Marvin L. Minsky, Nathaniel Rochester & Claude E. Shannon. 2006. A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955. *AI Magazine* 27(4). 12. doi:10.1609/aimag.v27i4.1904. <http://www.aaai.org/ojs/index.php/aimagazine/article/view/1904{%}5Cnhttp://www.mendeley.com/catalog/proposal-dartmouth-summer-research-project-artificial-intelligence-august-31-1955/{%}5Cnhttp://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.htmlhttp://>.

McDonald, Edward. 2008. *Meaningful Arrangements: Exploring the Syntactic Description of Texts.* Equinox Publishing Ltd.

McDonald, Ryan, Kevin Lerman & Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the tenth conference on computational natural language learning* CoNLL-X '06, 216–220. Stroudsburg, PA, USA: Association for Computational Linguistics. <http://dl.acm.org/citation.cfm?id=1596276.1596317>.

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

Miyao, Yusuke & Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage hpsg parsing. In *Proceedings of the 43rd annual meeting on association for computational linguistics* ACL '05, 83–90. Stroudsburg, PA, USA: Association for Computational Linguistics. doi:10.3115/1219840.1219851. <https://doi.org/10.3115/1219840.1219851>.

Moravcsik, Edith A. 2006. *An Introduction to Syntactic Theory.* Continuum paperback edn.

Natalia Silveira, Timothy Dozat, Marie-Catherine De Marneffe, Samuel Bowman, Miriam Connor, John Bauer & Chris Manning. 2014. A Gold Standard Dependency Corpus for English. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk & Stelios Piperidis (eds.), *Proceedings of the ninth international conference on language resources and evaluation (lrec'14)*, European Language Resources Association (ELRA). <http://nlp.stanford.edu/pubs/USD{_}LREC14{_}paper{_}camera{_}ready.pdf>.

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

Nivre, Joakim. 2006. *Inductive dependency parsing (text, speech and language technology)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

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

O'Donnell, Michael. 1994. Sentence Analysis and Generation: a systemic perspective. Tech. rep. Department of Linguistics, University of Sydney.

O'Donnell, Michael. 2005. The UAM Systemic Parser. *Proceedings of the 1st Computational Systemic Functional Grammar Conference* <http://www.wagsoft.com/Papers/ODonnellUamParser.pdf>.

O'Donnell, Michael J. & John A. Bateman. 2005. SFL in computational contexts: a contemporary history. In Ruqiaya Hasan, M.I.M. Matthiessen, Christian & Jonathan Webster (eds.), *Continuing discourse on language: A functional perspective*, vol. 1 Booth 1956, 343–382. Equinox Publishing Ltd.

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.

Postal, P. M. 1974. *On Raising*. MIT Press.

Quirk, Randolph, Sidney Greenbaum, Geoffrey Leech, Jan Svartvik & David Crystal. 1985. *A comprehensive grammar of the English language*, vol. 1 2. Longman. <http://www.amazon.com/dp/0582517346http://journals.cambridge.org/production/action/cjoGetFulltext?fulltextid=2545152>.

Radford, Andrew. 1997. *Syntax: A Minimalist Introduction*. Cambridge University Press.

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

Santorini, Beatrice. 1990. Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision). *University of Pennsylvania 3rd Revision 2nd Printing* 53(MS-CIS-90-47). 33. doi:10.1017/CBO9781107415324.004. <http://www.personal.psu.edu/faculty/x/x/xxl13/teaching/sp07/apling597e/resources/Tagset.pdf>.

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>.

Shang, Haichuan, Ying Zhang, Xuemin Lin & Jeffrey Xu Yu. 2008. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *VLDB Endow* 1(1). 364–375.

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

Stevan Harnad. 1992. The Turing Test Is Not A Trick: Turing Indistinguishability Is A Scientific Criterion. *SIGART Bulletin* 3(4). 9–10. <http://users.ecs.soton.ac.uk/harnad/Papers/Harnad/harnad92.turing.html>.

Stockwell, Robert P., Barbara Hall Partee & Paul Schacter. 1973. *The major syntactic structures of english*. New York: Holt, Rinehart and Winston. <http://hdl.handle.net/2027/mdp.39015002216417>. Bibliography: p. 811-840.

Tensiere, Lucien. 2015. *Elements of Structural Syntax*. John Benjamins Publishing Company translation by timothy osborne and sylvain kahane edn.

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

Tucker, Gordon H. 1998. *The Lexicogrammar of Adjectives: A Systemic Functional Approach to Lexis*. Bloomsbury.

Turing, Allan. 1950. Computing machinery and intelligence. *Mind* 59. 433–460.

Weerasinghe, Ruvan. 1994. *Probabilistic Parsing in Systemic Functional Grammar*: University of Wales College of Cardiff dissertation.

Winograd, Terry. 1972. *Understanding natural language*. Orlando, FL, USA: Academic Press, Inc. <http://linkinghub.elsevier.com/retrieve/pii/0010028572900023>.

Zhang, Niina Ning. 2010. *Coordination in syntax*. Cambridge University Press.

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

Normalization of PTDB and Cardiff TRANSITIVITY system

major	minor	new index	min # roles	max # roles
action	one role	one-role-action	1	1
	two role	two-role-action	2	2
	three role	three-role-action	3	3
relational	attributive	attributive	2	3
	locational	locational	2	3
	directional	directional	2	5
	possessive	possessive	2	3
	matching	matching	2	3
emotion	desiderative	desiderative	2	2
	plux xxx	emotive	2	3
perception	xxx	two-role-perception	2	2
	3 p Ag	three-role-perception	3	3
cognition	xxx	two-role-cognition	2	2
	3 p Ag/ matchee	three-role-cognition	3	3
environmental		environmental	x	x
influential	starting	starting	1	2
	continuing	continuing	1	2
	ceasing	ceasing	1	2
	succeeding	succeeding	1	2
	failing	failing	1	2
	causative	causative	1	2
	permissive	permissive	1	2
	enabling	enabling	1	2
	preventing	preventing	1	2
	delaying	delaying	1	2
	tentative	tentative	1	2

The process type column in PTDB contains two words separated by comma. The first one I call *major* as it represents a high level selection in the TRANSITIVITY system and the second one I call *minor* hints at selecting a particular participant configuration. The re-indexing consists of replacing the two features with a more delicate selection in the TRANSITIVITY system.