The UAM Systemic Parser

Mick O'Donnell
Escuela Politecnica Superior
Universidad Autónoma de Madrid
Madrid, 28049
micko@wagsoft.com

Abstract

This paper describes a Systemic parser currently under development at the Universidad Autónoma de Madrid. The parser achieves faster processing time by restricting the formalism somewhat. An adaption of the standard chart mechanism for Systemic Grammars is described, including the use of a statistically prioritised agenda.

1 Background

This paper describes a parser for a restricted Systemic formalism which is currently under development at the Universidad Autónoma de Madrid. The parser represents an evolution of the author's previous work on parsing (O'Donnell 1993; 1994; Matthiessen *et al* 1991). These earlier works attempted to parse with mid-coverage Systemic-Functional grammars (SFGs) in the full Hallidayan formalism. The first attempt was in the EDA project (Matthiessen *et al* 1991), funded by Fujitsu, where a large subset of the Nigel grammar (Matthiessen 1985) was used for parsing. However, the system parsed very slowly, and as the size of the subset was increased, processing became unworkably slow.

In later work, a specifically developed systemic grammar was used, utilising the same systemic formalism, but the grammar itself was reduced in complexity by avoiding the functional richness of Nigel, instead taking a more structural approach. However, as this grammar grew, it also fell victim of the complexity of systemic grammars, and the work was abandoned.

A further systemic parser was developed during 2000-2002 within Language & Computing (L&C), a Belgian informatics company. Together with Joeri Van der Vloet, Frederik Coppens and Maarten Van Mol, we developed a wide-coverage parser for English in the medical domain. A "systemic dependency" formalism was used, similar to Hudson's Word Grammar, but stated more in terms of the usual Systemic formalism of system network and realisations. Given the 30 months of development work, the resulting system worked very well, with workable accuracy on free text in the medical domain (and reasonable results in other domains given some lexicon extension). Parsing of 20 word sentences usually completed under 2 seconds, and 10 word sentences on average at 0.25 seconds. Keep in mind that the grammar, due to coverage, might produce 100s of alternative parses of the 20 word sentences.

Given that intellectual property restrictions restrict public use of this system, the author is currently developing a new parser within the public domain, one which will hopefully reach the same level of coverage and accuracy as the L&C parser. This paper describes the current state of this parser.

One lesson learnt from L&C was that it is more important to have a working parser than it is to preserve your formalism intact. The formalism used in this new parser is thus a step away from the full Hallidayan formalism. See details below.

2 Language Resources

2.1 Lexicon

The lexicon used in the parser (and for other tasks) is derived from the UMLS lexicon[1], a large lexicon for English distributed by the U.S. National library of Medicine, largely oriented towards medicine, but containing good coverage of the lexis of English.

This lexicon has been extended in various ways, including adapting the verbs to reflect something

[1] See: http://www.nlm.nih.gov/research/umls/.

approaching the Systemic process-type classification.

The lexicon has been extended to provide a single 'semantic' category for each of the nouns. These categories include features such as 'human-noun', 'thing-noun', 'substance-noun', 'event-noun', 'time-noun', 'location-noun', etc. In building the grammar, it becomes clear that certain grammatical distributions depend on such classes. For instance, nominal groups with time-nouns as head can act as adjuncts, as in "Last month I went to Spain."

In previous systems by the author, the features of lexical items were drawn from the grammar. In this new system, the lexical hierarchy is distinct from the grammar, and a separate resource maps lexical features onto the corresponding word-rank features of the grammar. This separation allows for modularity between lexicon and grammar, permitting the lexicon to be used for various tasks, and for the parser to use various lexicons.

2.2 The Grammar

This section describes the grammar used in the parser. The discussion assumes knowledge of the systemic formalism. Some reductions in formalism and description are then proposed to support more efficient parsing.

Reduced Formalism & Description: Parsing directly in terms of SFGs has proven difficult because of the functional complexity of these grammars in comparison to grammars of other formalisms, particularly in regard to analysing each sentence in terms of multiple layers of structure, and also due to the semantic richness of systemic descriptions. Earlier work (O'Donnell 1993), looked for algorithms to handle the functional complexity of SFGs. However, as the grammar grows to a coverage sufficient to allow free text parsing, the complexity leads to unworkably slow processing times.

For this reason, in this current parser, it was decided instead to *reduce* functional complexity by restricting the formalism and description. These restrictions are discussed below:

1. *Reduced Functional Coverage*: a normal SFG assigns function structure to a sentence in terms of three layers: Transitivity (Actor/Goal, Sensor/Phenomena, etc.), Mood (Subject/Predicate/Object/Adjunct, etc.) and Theme (Theme/Rheme). However, much of the parsing complexity comes from producing these different layers of structure, and mapping them onto each other. One major problem is that to discover what can come next in a structure, one cannot rely on a single layer, as ordering information could be stated at each of the layers. One can thus only parse a normal SFG by exploring all layers simultaneously.

Several approaches avoid dealing with functional complexity by pre-parsing with another formalism, e.g., a PSG grammar in the case of Kasper (e.g., Kasper 1988), an HPSG grammar in the case of Bateman *et al* 1992. We prefer however is to parse directly in a single formalism, rather than maintain two separate grammars, and the mappings between them.

To this end, we restricted the grammar so that, functionally, only Mood structure (Subject/Predicate/DObj/IObj, etc.) is accounted for. The grammar does deal with process type variation (material, mental, verbal, etc.), but only in terms of systemic choices (features), not in terms of functional labels. The thematic placement of elements (e.g., fronting of Objects, Adjuncts, etc.) is also dealt with systemically, without reference to functions such as Theme or Rheme.

One reason for following this approach is because I believe these other layers are best treated as semantic analyses, a strata above the grammar. Routines are thus used to derive experiential structure (a dependency structure of participant roles such as Actor, Goal, etc.) directly from a Mood-based syntactic structure. Additionally, thematic and contrastive considerations are treated at a semantic-level, not as part of the grammar.

2. *Reduced Formalism (i) unifunctionality*: by dealing with only one layer of function structure, the need for conflation in the grammar is greatly reduced. Because conflation adds considerable complexity to a grammar, I decided to do away with it entirely, thus allowing only a single functional label per constituent.

The only problem that occurred was in the verbal auxiliaries, to ensure agreement between successive verbs in the verb chain, and with the Subject. Figure 1 shows a set of systems which account for verb sequences in a grammar using conflation (they generate "He will have been being eaten" and most variants). Figure 2 shows a network which generates exactly the same verb sequences, but does so without conflation. Note that this latter

network approximates a plate of spaghetti. However, this one small area of additional network complexity is a small cost for the reductions in complexity gained by being able to forget about conflation. See figure 3 for a sample analysis produced by the grammar, demonstrating unifunctionality.

3. *Reduced Formalism (ii) slot-based ordering*: Another prime area of complexity in a standard SFG is in calculating which elements can follow a given element. In earlier work by the author, this area of computation resulted in the largest complexity. Removing the possibility of conflation removes the problem to a degree, but not totally. The previous work used the Nigel formalism, which deals not only with *order* statements, which position two elements adjacent to each other, but also *partition*, which says one element occurs somewhere after another, *orderAtFront*, which says the named element should appear as first element, and

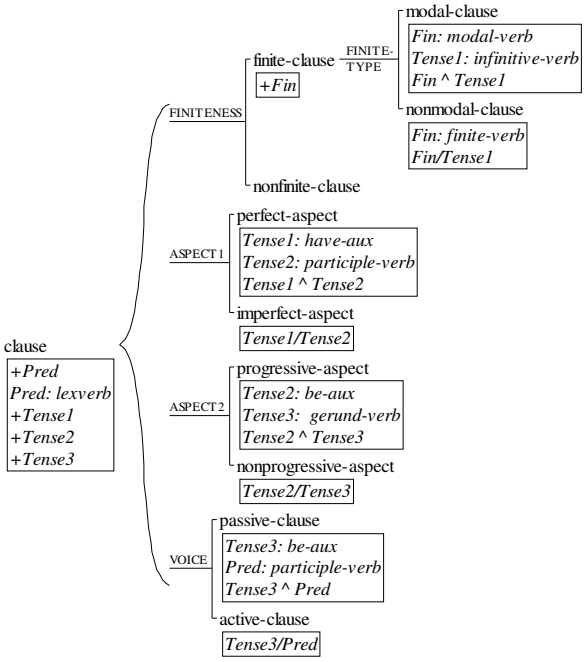

Figure 1: Auxiliary Verb Systems with Conflation

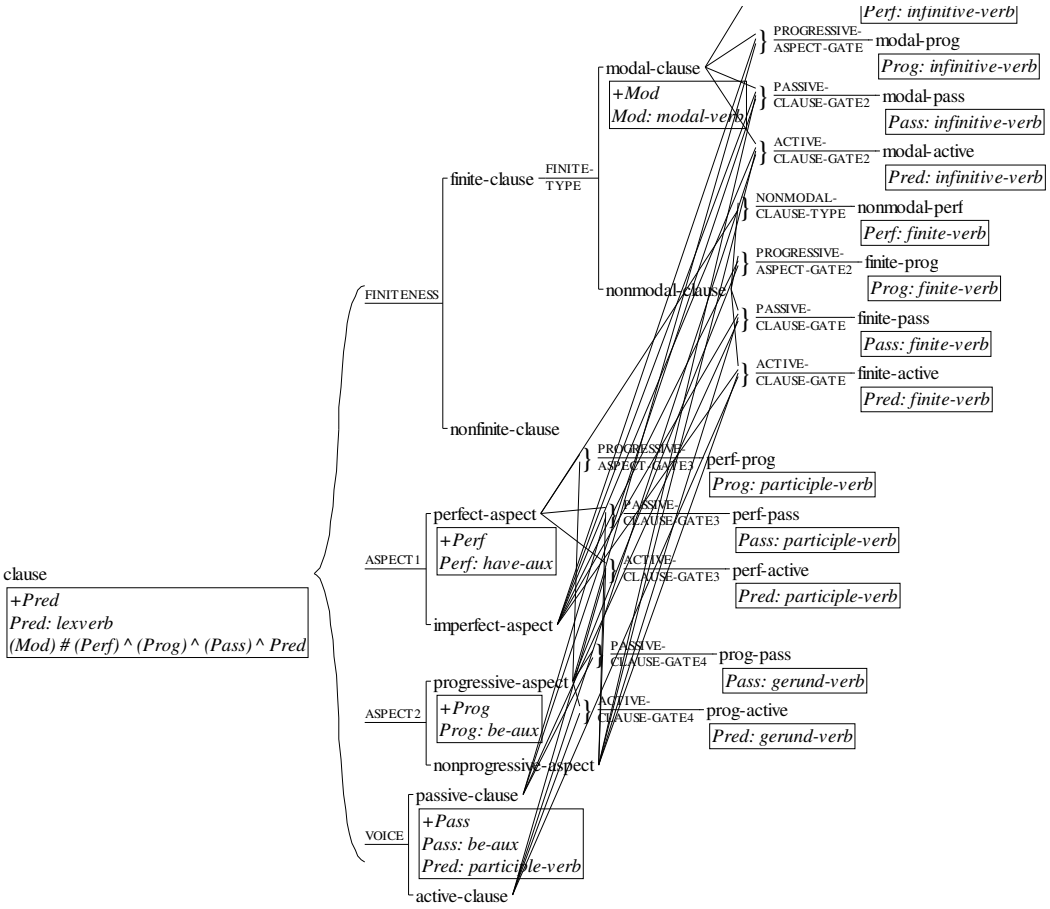

Figure 2: Auxiliary Verb Systems without Conflation

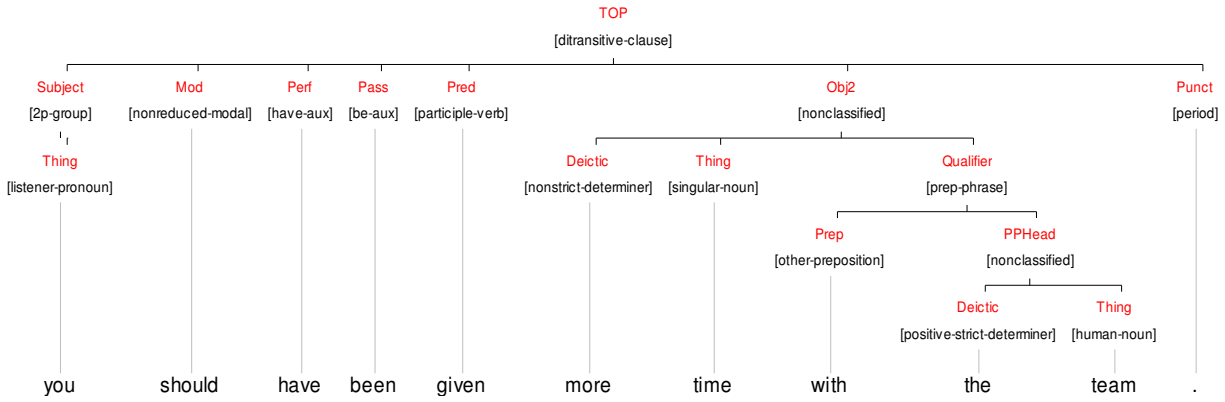

Figure 3: A sample sentence analysis

orderAtEnd, which says the named element should appear as the last element. Nigel was developed as a generation grammar, and when applied to parsing, it was found that it vastly under-constrained ordering. Nigel has an additional resource, a default ordering list of all functions, which is used in generation to determine final order. However, for parsing, the grammar did not define all possible sequences of elements, and thus was not directly utilisable for parsing.

During the EDA project, where we used a large fragment of the Nigel grammar for parsing, it was necessary to add explicit sequencing rules. While Nigel could get away with two-element *order* and *partition* rules, to fully represent potential element sequence in the grammar, this syntax was expanded to allow sequencing of any number of elements, and also to allow reference to optional elements (whose presence will be determined by an *insert* statement elsewhere in the grammar), e.g.,

```
order Deict (Num) (Epith) (Classif) Thing
```

...which places Deict before the Thing, and allows for other elements to intervene.

The real problem of ordering was due to the *partition* statement. It makes working out what can follow a given function a complex process. For instance, imagine we have a grammar with the following three realisation statements somewhere in the grammar:

```
[a] ↘  order Pred DObj
[b] ↘  partition Pred Manner
[c] ↘  partition Pred Cause
```

To find out what can follow a DObj, we not only need to check for *order* statements which specify a following function, we also need to check for any *partition* statement naming a function to follow any previous element in the current structure. For instance, if we have just recognised a Pred element and a DObj element, then the next element could be a Manner or a Cause. In actual practice, this process gets very complex, particularly due to potential conflation of elements.

To avoid this complexity, a simpler approach to constituent ordering was adopted. In dependency grammar, it is commonplace to specify element ordering by assigning them to a slot position relative to the head. The same approach is followed here: realisation rules assign each element to a numbered slot, and those assigned to lower numbered slots appear before those in higher numbered slots. If two elements are assigned to the same slot, then these elements can appear in any order in respect to each other (useful for the free order of adjuncts of different types). Note that Fawcett takes this approach in his grammar. I was critical of his approach in earlier days, but as I am now focused on ends rather than means, I better understand his rationale.

Formalism Extension: handling multiple fillers: SFL has long had problems allowing multiple occurrences of the same element.[2] In some grammatical contexts, a given syntactic function can appear multiple times. For instance, we can have multiple Epithets, as in "the big brown cuddly teddybear". Qualifiers and Adjuncts can also be repeated, and 'and/or' complexes are indefinitely extendable.

The standard approach by non-computational systemicists is to draw a *recursive network*, that is, a system which can loop, and on each loop, the option is available to insert another element of structure or to stop (see figure 4).

[2] For good discussion on this issue, see Patten (1988), p41-3.

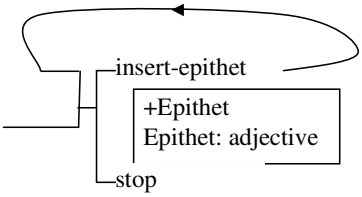

Figure 4: A recursive system network

Bateman, in his work on Japanese, implemented a prototype system allowing sentence generation with such networks (see Matthiessen & Bateman 1991, pp148-165). However, the implementation of recursive networks is questionable, since it requires a unit's selection expression (its feature set) to contain multiple selections from a system, which goes against the norms of implementing systemic grammars.

Other implementations have 'simulated' recursion: under the feature that inserts a function, a further system offers a choice to insert another, e.g., as in figure 5a. Sometimes a third layer is added.

Since any number of Epithets (or Qualifiers, Adjuncts, etc.) might be needed, this approach is not adequate (although statistically more than 3 occurrences of a function is quite rare. However, it does happen).

PSG handles this type of recursion quite easily, using a "*" operator, indicating that the tagged element can appear one or more times., e.g.,

```
NP -> adj* ^ noun
```

I prefer this approach to the above approaches, and thus opted to implement it. The system of figure 5a

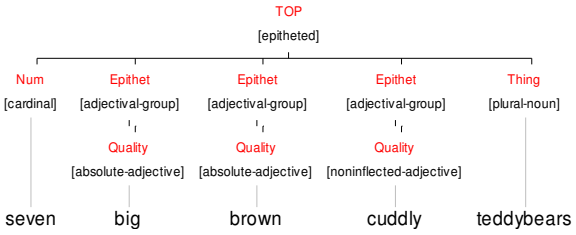

Figure 6: Parse tree showing function repetition

is reduced to that of figure 5b, which handles any number of Epithets. Figure 6 shows an analysis of a nominal group with multiple Epithets.

There is one problem with this approach. Often systemic grammars provide systems which enumerate the different fillers possible for a given function. For instance, if we allow Epithets to be filled additionally by gerund and participle verbs, we might produce a network such as figure 5c. Now, if we were to place a '*' on the insertion of Epithet, these systems would seem to say that we can have any number of Epithets, but they must all be of the same type (since a given system can only have one selection from it). To avoid this situation, I instead represent the choice of filler in the preselection operation itself, as in figure 5d.

3 The Parser

The current parser has the following features:

- **Bottom up**: generally, bottom-up parsing is more efficient than top-down parsing, as it starts with the tokens of the input string, while the top-down approach needs to build hypothetical structure before reaching down to

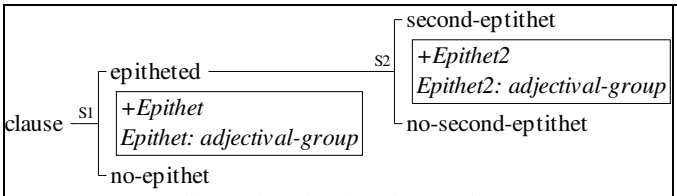

Figure 5a: Simulated recursion

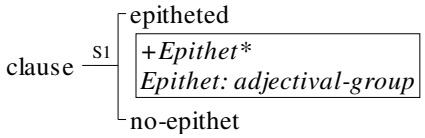

Figure 5b: Allowing multiple fillers

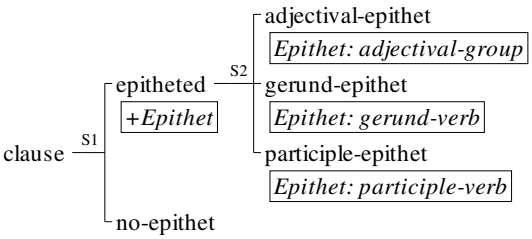

Figure 5c: Specifying Fillers without '*' operator

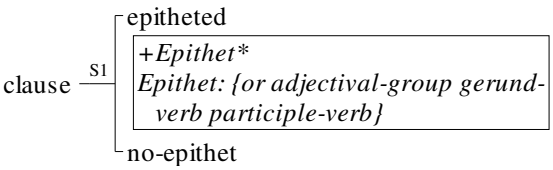

Figure 5d: Specifying Fillers with '*' operator

word rank.

- **Left-to-right**: a left-to-right processing strategy is used, because I have not found any advantage to either right-to-left processing or head-driven processing.

- **Chart parser**: most parsers these days use a chart in some form, as it avoids needlessly repeating structural analysis of sub-strings. A normal chart parser explores all structural possibilities in parallel, and avoids the need to reanalyse any sub-sequence.

- **Agenderised Chart**: an extension of the chart parser (sometimes attributed to Kay 1973) involves the use of an *agenda* with the chart. Normally in chart parsing, when faced with a choice of rules to apply to a chart edge, all are applied. With an agenderised chart parser on the other hand, as structural operations become possible, they are placed on the agenda. The main control program operates by continually popping the first operation off the agenda and applying it to the chart. Done properly, the parser can operate like a backtracking parser, producing a first complete analysis without exploring all possibilities. The use of a chart however means that no analysis ever needs to be redone, as is necessary with backtracking parsers.

- **Prioritised Agenda**: Before being placed on the agenda, each operation is assigned a value, an estimate of its likelihood of being part of the 'correct' parse. This value is usually derived from analysing a treebank, e.g., to see how often a particular rule is involved in the correct parses. Operations are ordered on the agenda in terms of this probability. The result is that the parser applies operations to the chart in order of their probability of being part of the correct parse. This leads to higher likelihood that the correct parse is produced earlier rather than later in the chart-filling process.

2.1 Segmentation & Tokenisation

The UAM CorpusTools includes a sentence segmenter and tokeniser, which divides a plain text file into sentences, and within each sentence, recognises token boundaries. Multi-word tokens are recognised when contained in the lexicon.

2.2 Precompilation

O'Donnell (1994) argued for precompiling the systemic grammar into a form more suitable for parsing. The current system takes the same approach. Precompilation means that calculations which tend to be repeated for each sentence analysis are performed only once. The grammar is precompiled into two resources:

1. **Sequence table**: a table detailing what functions can follow a given function in a given slot. Each entry also details the feature constraints on the edge, and also any feature constraints on the filler of the function. This table is constructed by logically combining both insert statements and preselect statements.
2. **Starter table**: a table detailing what functions can legally start an edge, and the feature constraints on the edge, as well as feature constraints on the starting constituent.

The sequence table is similar to the 'ordering partial structures' of O'Donnell (1994), although the compilation is vastly simplified because conflation is not involved. Additionally, the preselection information is now combined into the structures as this allows the applicability of each continuation to be conditioned by the features of the element we are trying to attach into the structure.

2.3 Chart Operations

Before giving details of how the UAM parser operates, some basic concepts of the chart will be outlined. Firstly, the chart maintains details of the alternative analyses of the input tokens. Each analysis can either be a *complete edge* (meaning the system has decided this unit can stand alone as a unit without further elements), or a *working edge* (meaning the system is still trying to add extra elements at the end of the unit).

The UAM parser allows 4 kinds of chart operations:

- **Enter(SENSE)**: creates a completed edge for the lexical sense, and enters it on the chart.

- **Complete(WU)**: test to see if current working edge, WU, can be considered a complete syntactic unit. If so, a new completed edge is produced, copying the structure of the working edge. The feature constraints on the edge are modified to restrict the insertion of other elements in the structure (e.g., a group with last

element Thing will have feature "not-qualified" added to its feature constraint).

- **Extend(WU CU)**: attempt to add the complete edge CU as a new constituent of working edge WU, at the right of the structure.
- **Raise(CU)**: find possible new working edges which can have the current completed edge as its leftmost constituent. The raising of a completed edge produces a new working edge with the old edge as first constituent.

2.4 Parsing Process

The parsing process proceeds as follows:

1. **Seeding the agenda** at the start of processing, we identify each sense of each token in the input string. For each sense, we add an entry to the agenda 'enter <sense>'. When later executed, this operation causes the lexical sense to be added into the chart. Note that each lexical sense can be awarded a probability of correctness (using frequency in the current register, possibly modified by trigram frequencies). As the agenda is statistically prioritised, lower rated senses might not enter the chart until higher rated senses have failed to produce a parse. Bobrow (1990) comments that this allows the system to deal with rare uses without compromising parsing time for normal uses.
2. **Loop through agenda**: we take each operation in turn from the top of the agenda, and process it. Each operation will build the chart in some way. The operation will in turn place some new operations onto the agenda. Parsing is finished when the agenda is empty. Alternatively, the system can be set to terminate when a preset number of parses covering the entire input string has been produced.

2.5 Processing Agenda Items

When an Agenda item is taken for processing, it is processed as follows:

a) **raise CU**: we check the precompiled "starter table" to see what types of elements can start with an element of the same class as CU (each element in the grammar is assigned a basic class, such as noun, nominal-group, clause, etc. These classes are used for quick filtering in feature constraint matching). For each of these,

1. The more detailed feature specification is checked to see if the operation is in fact valid.
2. If so, a new working edge WU is created, with CU as its first constituent, with the nominated relation.
3. WU is assigned the necessary feature constraints to allow this constituent to come first.

b) **complete WU**: the feature condition of WU is checked for compatabity with the lack of any more constituents. If so,

1. A copy of the edge, CU, is made;
2. CU's type is changed to COMPLETE;
3. CU's feature specification is extended to restrict addition of any other constituents;

c) **extend WU CU**: this operation makes use of the 'sequence table' derived from the grammar in the precompilation stage, which details which elements can continue a partially analysed structure.

1. Find the last function label (and its slot number) in WU,
2. Look up the set of possible candidate extenders.

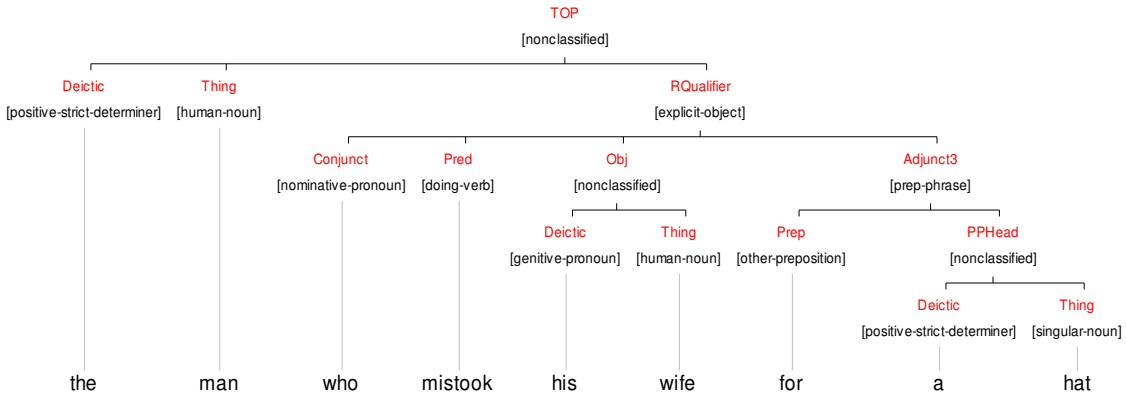

Figure 7: Parse involving relative clause

3. Of these, eliminate those which do not match the class of the child, CU.
4. Test that the rule's feature constraint on the parent unit can unify with that of WU. If not, proceed to next rule.
5. Test that the rule's feature constraint on the child unit can unify with that of CU. If not, proceed to next rule.
6. Create a new edge, CU1 which is a copy of WU, with added constituent CU. To the feature constraints of CU add the features which require the insertion of the new constituent.

2.6 Post-operation calculation

When a new edge (created by performing a chart operation) is entered on the chart, a routine is triggered which looks for consequences of this change. The routines differ between working and completed edges. For a new working edge WU, the routine is:

1. For each completed edge CU that begins immediately after this edge, agenderise "extend WU CU".
2. Agenderise "complete WU"
3. Add WU to the list of working edges that end at its rightmost token position.

For a new completed edge CU, the routine is:

1. Test if CU covers all words in the sentence. If so, it is a complete parse, and is added to the VALID_PARSE list. If the PARSE_LIMIT is reached, processing stops.
2. Add CU to the list of completed edges which start at its leftmost token position.
3. Place "raise CU" on the agenda (for later processing)
4. For each working edge WU that so far ends just before CU, agenderise "extend WU CU" (i.e., test if CU can extend WU).

2.7 Statistical Ordering of Agenda

At the time of writing, the use of statistics to prioritise the agenda has not yet been introduced. For this, a treebank is needed from which to derive statistics as to the syntactic likelihood of each chart operation (e.g., likelihood of an NP to fill Subject rather than Object slots, likelihood of a Classifier to follow a Deict compared to a Thing element, etc.).

However, the treebank is currently under construction, using a treebank annotation tool developed at the UAM.

When the statistics are at hand, each chart operation will be assessed as to its likelihood for being part of the final "best" parse, and inserted on the agenda behind those of higher likelihood. In this way, those operations with highest likelihood of leading to a correct parse are executed first, hopefully leading the parser to produce the best parse amongst the first produced. Experiences with the L&C parser suggest that this is indeed the case.

The parser can then be set to stop after a small number of parse trees have been produced, without needing to populate the chart completely, vastly reducing parse time. At that point, the available parses are each assessed for likelihood (using the product of probabilities for each operation in the structure), and the highest scored tree is selected as the preferred analysis.

Alternatively, the parser can be set to stop producing alternatives after some time limit is reached, and select the best parse of those produced.

4 Conclusions

The UAM parser is a compromise between the desire to work with a systemic formalism and description, contrasted with the pragmatic need to get a wide-overage system up and running today.

The formal and descriptive limitations taken in the system do reduce its value to a degree, as the extra information contained in a full systemic analysis provides a powerful basis for deep semantic work, such as interpreting propositional content, exploring the author's method of development, etc.

However, no wide-coverage parser using the full Hallidayan formalism has yet eventuated, while the demands for systemic-oriented sentence analysers is on the increase.

Because of this, we have opted for the compromise solution of producing a system at least partially systemic, while capable of fast sentence parsing with a large grammar and lexicon.

The UAM parser will be freely distributed to the community, and along with some basic linguists tool based on its functionalities (e.g., clause segmenter, process-type classification of clauses, tools for discourse critiquing of texts, etc.).

References

A. Ruvan Weerasinghe & Robin Fawcett. 1993. Probabilistic Incremental Parsing in Systemic Functional Grammar. *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, the Netherlands, August 10-13, 1993.

Christian M. I. M. Matthiessen. 1985. "The systemic framework in text generation: Nigel." In James Benson & William Greaves (eds.) *Systemic Perspectives on Discourse*, Volume 1, 96-118 Ablex, Norwood, New Jersey.

Christian Matthiessen, Michael O'Donnell & Licheng Zeng. 1991. "Discourse Analysis and the Need for Functionally Complex Grammars in Parsing." *Proceedings of the Second Japan-Australia Joint Symposium on Natural Language Processing*, October 2-5, 1991, Kyushu Institute of Technology, Iizuka City, Japan.

Christian M. I. M. Matthiessen & John A. Bateman. 1991. *Text Generation and Systemic-Functional Linguistics: Experiences from English and Japanese*. Pinter: London.

John A. Bateman, Martin Emele & Stefan Momma. 1992. "The nondirectional representation of Systemic Functional Grammars and Semantics as Typed Feature Structures." *Proceedings of COLING-92*. Nantes, France, Volume III, 916-920.

Michael O'Donnell. 1993. "Reducing Complexity in a Systemic Parser", in *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, the Netherlands, August 10-13, 1993.

Michael O'Donnell. 1994. *Sentence analysis and generation: a systemic perspective*. PhD thesis. Department of Linguistics, University of Sydney, Australia.

Robert J. Bobrow. 1990. Statistical agenda parsing. *DARPA Speech and. Language Workshop*, pages 222-224.

Robert Kasper. 1988. An Experimental Parser for Systemic Grammars. *Proceedings of the 12th Int. Conf. on Computational Linguistics*, Budapest, Association for Computational Linguistics.

Tim F. O'Donoghue. 1991. The Vertical Strip Parser: A lazy approach to parsing, *Research Report* 91.15, School of Computer Studies, University of Leeds, Leeds, UK.