Light Parsing as Finite State Filtering

Gregory Grefenstette
Rank Xerox Research Centre

Abstract

For a number of language processing tasks, such as information retrieval and information extraction tasks, pertinent information can be extracted from text without doing a full parse of the individual sentences. The most common restriction of the parser is to adopt a non-recursive model of the language treated, which allows an implementation of the parser using efficient finite state tools at the cost of missing some coverage. These light parsers allow the successive introduction of symbols into the input string wherever specified regular expressions of words and/or part-of-speech tags match. Recent advances in finite state expression compilation make writing mark up transducers simpler, leading to quicker implementations of layered finite state parsers. The resulting parsers are easier to create and maintain. In this article, we describe a light parsing method using recently created finite state operators. Two applications of this parser are described: grouping adjacent syntactically related units, and extracting non-adjacent n-ary grammatical relations. A system for evaluating the parser over a large corpus is described.

1 Introduction

For large scale text applications, a parser should be robust, rapid, and relatively accurate. In such applications as terminology extraction, lexicography, or information retrieval, o For large scale text applications, a parser should be robust, rapid, and relatively accurate. In such applications as terminology extraction, lexicography, or information retrieval, one of the purposes of parsing is to recognize recurring lexical syntactic patterns and their variants. The parser used need be no more powerful than is necessary to recognize these patterns. Many such parsers (Debili 1982) (Grefenstette 1983) (Grefenstette 1992) (Abney 1991) (Appelt *et al.* 1993) employ deterministic finite state recognizers over part of speech tagged text by first marking contiguous patterns as noun and verb groups, then marking heads within groups, and then extracting patterns between non-contiguous heads. Some of these parsers mix non-finite state procedures with finite state recognizers, but we show here that the entire parser can be built within a finite state framework.

Finite state transducers can introduce markings and labels within regular patterns of tagged text. Other finite state transducers can be composed with these marking transducers to be used as filters to extract and label a wide variety of n-ary dependency syntactic dependency relations between words. Recent advances in finite state calculus and finite state compilers (Karttunen 1996) allow for more powerful and more elegant descriptions of the patterns to be recognized, making parser writing and maintenance easier.

In this paper, we briefly review finite state transducers, and their application as text markers and filters. We then proceed in three stages: (1) using new finite state longest match and replacement operators, we create transducers identifying contiguous noun group and verbal group boundaries; (2) we use labeling transducers to mark the heads within each group; and (3) we create filtering transducers which label the syntactic relations between words within and across group boundaries, for example extracting verb-subject pairs. Finally, we evaluate the use of these filters over a large publically available text corpus.

2 Finite State Transducers

Informally, a finite state transducer[1] is a finite state machine (visualizable as a set of nodes corresponding to states and directed arcs corresponding to transition) that consumes input and produces output with each state transition. Each arc is labeled with a symbol "under the arc" and a symbol "above the arc." Depending upon which direction the transducer is applied in, the upper symbol can be considered as an input symbol and the lower as an output symbol, or vice versa. If the input symbol is an epsilon (written as a pair of empty brackets `[ ]`), no input symbol is consumed as the arc is followed, but the output symbol is inserted in the output. The output symbol may also be an epsilon meaning that the input symbol is consumed without producing an output.

A finite state transducer can be written as a regular expression. For example, the regular expression *x:x a:y* $*$ corresponds to the transducer that accepts an input of one x followed by any number of y's and produces an output of one x and as many a's as there were y's in the input. This expression may also be written *x y:a* $*$ since our finite state compiler considers a single symbol without a colon as an arc with the same symbol on the input and output sides.

An arc written as *[ ]:x* consumes an input x without producing an output, and an arc written as *y:[ ]* outputs one y without consuming any input.

A finite state transducer that introduces extra symbols into an input string can be considered as a **finite state marker**. We use finite state markers to introduce brackets around nominal groups and verbal groups, and to mark head words within these groups.

Finite state filters are transducers which output only certain parts of the input string, setting all the other parts of the string to epsilon and possibly introducing a filter indicating label. Once created, finite state filters, as with any finite state transducer, can be composed or unioned.

3 Noun and verb group demarcation

Before extracting non-contiguous syntactic dependency patterns with finite state transducers, we first use finite state markers to introduce noun[2] and verb group

[1] See (Mohri 1996) for a formal treatment of the realization of deterministic transducers such as those used in this article.

[2] We adopt a prescriptive, rule based, approach to such phrasal isolation, rather than a stochastic approach which recognizes small nominal chunks such as in (Church 1988). The reasons for this are many: one need not have a hand tagged text so that it is easier to apply the same techniques to languages other than English; we are interested in seeing how far the finite state approach to language processing can be pushed; and by using rules, it is easier to explain and correct errors.

```
NounGroup =
  [ [ Art    => _ [ Noun  ]] &
    [ Noun   => _ [ PAdj | Prep | .#. ]] &
    [ PAdj   => _ [ PAdj | Prep | .#. ]] &
    [ Prep   => _ [ Art  | Noun ]]        ]
   &
   [ [ Art | Noun ] [ Art | Noun | Padj | Prep ]* ] ;
```

Figure 1: Defining a regular expression, called `NounGroup`, which is a finite state compiler representation of a precedence matrix (Debili, 1982) describing French noun phrases. Brackets (`[` and `]`) are used for grouping, the symbol `&` means intersection, the vertical bars `|` stand for union, and the star `*` is the Kleene star. The double arrow operator `A => B __ C` means that whenever the regular expression `A` appears, it must be preceded by the something matching the regular expression `B` and followed by something matching the regular expression `C`. The _ shows where the left hand side expression `A` must appear. In the four uses of the double arrow operator above the regular expression corresponding to `B` is always absent since only the right context is used in a precedence matrix. The line beginning with `Noun` can be read as "Noun must be followed by either a Posterior Adjective or a preposition or else end the group." The symbol `.#.` stands for the end of the regular input string and is recognized by the finite state compiler. The line beginning `[[Art | Noun]]` intersects with preceding and states that this simplified noun group must begin with either an article or a noun, and must contain only nouns, articles, prepositions and posterior adjectives.

boundaries into tagged text. In his dissertation work, Debili (1982) used part of speech precedence matrices, coupled with vectors describing possible starting and ending parts of speech, to describe the contour of nominal and verbal groups. Such matrices can be easily created in our finite state calculus notation as shown in figure 1. The noun group recognizing machine described by this regular expression is given the alias `NounGroup`. Figure 2 shows how this machine is turned into a marking transducer using the deterministic replacement (Karttunen 1996) operator `@->` and the insertion point symbol three dots (`...`). This transducer unambiguously inserts noun group (<NG and NG>) markers around the longest instances of these groups in each sentence.

When a slightly more complete version of this group marking transducer, as well as one for verb groups, is run over the part of speech tagged sentence:

> Administration/NN of/IN 10/CD per/IN cent/NN oxygen/NN to/IN the/AT ewe/NN for/IN 1/CD hour/NN prior/NN to/IN delivery/NN did/DOD not/NOT alter/VB the/AT surfactant/JJ properties/NNS of/IN the/AT fetal/JJ tracheal/JJ fluid/NN

the noun and verb group markers <NG NG> <VG VG> are inserted as follows:

> <**NG** Administration/NN of/IN 10/CD per/IN cent/NN oxygen/NN to/IN the/AT

```
MarkNGroup = NounGroup @-> "<NG"  ...  "NG>"  ;
```

Figure 2: Definition of an unambiguous noun group marking transducer, called `MarkNGroup`. The operator `A @→ B ... C` inserts the expressions `B` and `C` around the longest substrings that match the regular expression `A` in a string, in a left to right manner. A string enclosed in quotes such as `"<NG"` is considered as one symbol by the finite state compiler.

> ewe/NN for/IN 1/CD hour/NN prior/NN to/IN delivery/NN **NG**> <**VG** did/DOD not/NOT alter/VB **VG**> <**NG** the/AT surfactant/JJ properties/NNS of/IN the/AT fetal/JJ tracheal/JJ fluid/NN **NG**>

3.1 Marking the Heads of Phrasal Units

Once the group markers are inserted into the tagged text by the transducers described above, another transducer places head labels before certain classes of words within the groups. These additional labels indicate that the words appear in specified contexts and make the task of writing syntactic filters (see Section 4) easier and shorter.

Nominal heads are marked as being modified by prepositions (`*PrepN`) or as noun phrase heads (`*HeadN`) by the transducers produced by the regular expressions in Figure 3. These expressions state that the empty string `[ ]` is replaced with the symbol `*HeadN` in front of a noun that is not followed by another noun, or by one that ends a noun group. This transducer is composed with another which replaces the `*HeadN` label with a `*PrepN` when the label is preceded by a preposition with no intervening prepositions, commas, etc.

Verbal heads are similarly marked with aspect labels inside verbal groups. Inside a verbal group, verbs will be preceded by either an active verb label `*ActV`, a passive verb label `*PasV` or as an attributive verb `*BeV`. Inside noun groups, verbs are marked as past participles `*EdV`, present participles `*IngV` or as infinitive verbs `*InfV`.

For example, the sentence:

> Significant correlations were obtained between the maternal and fetal glucose levels and the maternal and fetal ffa levels.

receives, after being tagged, the following inserted labels:

```
HeadNouns =
 "*HeadN" -> [] || [ "<NG" | TAG ]  _ [ NOUN  [ ~$ NOUN ]
                                              [INGVERB | PPART |
                                               PREP | COMMA |
                                               CC | "NG>" ] ;
 # Add in a *HeadN label after the beginning of a noun group
 # or after any other tagged word,  and before a noun
 # which is not followed by another noun until you
 # reach an  ingverb, a past participle, a preposition
 # or a comma or a conjunction, or the end of the noun group.


PrepNouns =
"*PrepN" ->  "*HeadN"
             ||  [ [$ PREP] &  ~$[ PREP ?* [PREP | COMMA | "NG>" ]] _ ;

 # Replace  *HeadN by a *PrepN  when there is some preceding
 # preposition with no intervening preposi-
tion, comma or other noun group

LabelNounFST = [ PrepNouns .o. HeadNouns ]

 # Create a finite state marker which first marks heads
 # of noun groups
 # and then alters some of the *HeadN labels into *PrepN labels
```

Figure 3: Creating a noun group head labeling finite state marker. In the regular expressions following the equal signs, the symbol `?` stands for any character; the dollar sign, followed by a regular expression, means anything containing the following regular expression; the tilde $\sim$ means “not,” so the sequence $\sim$`$` can be read as “does not contain;” the finite state grammar construction `A` $\rightarrow$ `B` $\|$ `C _ D` means that `A` replaces `B` when `B` is preceded by `C` and followed by `D`. The first expression in the above figure can be read: “Insert a head noun label before any noun that begins a noun group, or follows some other tagged word, but which is not followed by a noun.” The second expression changes this tag into a prepositional label when it is preceded by a preposition without any intervening prepositions, commas, or noun phrases. These two expressions are composed using the composition (`.o.`) operator. The other symbols `TAG, NOUN, INGVERB`, etc. correspond to previously defined regular expressions matching different parts of part-of-speech tagged text.

<NG Significant/JJ *HeadN correlations/NNS NG> <VG were/BED *PasV obtained/VBN VG> <NG between/IN the/AT maternal/JJ and/CC fetal/JJ glucose/NN *PrepN levels/NNS and/CC the/AT maternal/JJ and/CC fetal/JJ ffa/JJ *PrepN levels/NNS NG> ./SENT

4 Syntactic Function Filters

The preceding section showed how finite state marking transducers are used to mark contiguous group boundaries and to label head words within these groups. In this section, we show how filtering transducers using these marks and labels can be created to extract non-contiguous syntactic n-ary dependencies. These filters take advantage of the finite state possibility of regularly specifying what must not appear between two items.

The basic principle in describing a syntactic dependency filter between two words is to describe the structure of the syntactic context to the left of the first word (with a regular expression `LEFTCONTEXT`), between the words (with a regular expression `MIDDLE`), and to the right of the last word (with a regular expression `RIGHTCONTEXT`). None of this context is output (it is set to the empty string, epsilon `[ ]`) by the transducing filter as the context is recognized. The only itens that are output by the filtering transducer are the tokens (the tagged words) that are found in surrounded by the contexts, and a relation label inserted after the last word. Schematically this gives the following structure:

```
[ ]:LEFTCONTEXT
            token
                [ ]:MIDDLE
                        token
                           relation:[ ]
                                [ ]:RIGHTCONTEXT
```

Similar syntactic filters were described and implemented in 1982 by Debili (1982, p. 99). Having introduced nominal and verbal group delimiters and labeled heads within these groups makes it easier to write expressions that filter out syntactic functions between words. For example, Figure 4 shows a simple "subject of an active verb" filter can be described as the head noun(s) in the preceding noun group. This definition will not always match the true subject, but we can estimate its accuracy as will be shown in the next section. Figure 5 shows a filter for extracting syntactic subjects of passive verb phrases.

5 Evaluation

Syntactic filters can be evaluated in the following way: (i) take any corpus, (ii) tag the corpus, (iii) mark verb and noun group boundaries, (iv) label noun and verb heads, (v) apply filters, (vi) sample results, (vii) count errors in sample. We ran these two filters over the first megabyte of AP news[3] from 1988, and randomly

[3] Available from the Linguistic Data Consortium. http://www.ldc.upenn.edu.

chose one hundred instances of each filter output. On a SPARC 20, tagging the 164,000 words took about 15 seconds of real time, inserting noun and verb phrase boundaries via non-optimized marking transducers took about 2 minutes, marking heads inside boundaries took about 3 minutes 12 seconds, and applying the union of 17 different filters took about 11 minutes. In all, this is about 10,000 words per minute. Manual evaluation showed that 76(Figure 4) relations from these 8000 sentences correct. 80% of the relations extracted by passive direct object filter shown in Figure 5 were correct. These numbers can be improved by introducing more complicated filters, but, noisy as they are, they already can provide useful indications of subcategorizations, given enough text. As an anecdotal example, the typical passive direct objects extracted for the word *killed* in the AP corpus were: people, seaman, villagers, vendors, teen-agers, soldiers, rebels, pilot, patient, ...

5.1 Error Analysis

Here is a detailed description of the 12 errors in subject recognition that this simple SUBJ filtered produced:

- Preposition scope unrecognized — (2 of 12 errors)
  Iowa [along with *Michigan, assessment,* and *contest*] chosen as subject of *make* in *It's evident to me that the contest in Hawaii, Michigan, and Iowa, and my own assessment of New Hampshire, make it clear that Al Haig will not be the Republican candidate in 1988*
  The filter that marks prepositional dependency could not handle the paranthesization of the following sentence and *nozzle* was chosen as a subject of *came* in *Tests have resulted in "fractures ... so numerous, so large, so closely spaced and so extensive that the integrity of the entire (nozzle) came into question," said the scientists' study, obtained by the Post.*
- Subjectless imperatives unaccounted for, adverbially time expressions not implemented — (1 of 12)
  day and *minute* are chosen of subjects of *listen* in *How can you conduct negotiations together when your partner runs every day and every minute to the other side and says, 'Don't listen to what Shamir says, I'll sell it to you cheaper?*
- Tagging errors — (4 of 12)
  In the version of the tagger used in these tests, any tagging errors remained unchanged and led to errors, as in the following sentence in which *charges* was tagged as a verb, with *killings* as its subject. *The killings of five Irish nationalist guerrillas and a teen-ager prompted charges that police had a shoot-to-kill policy.*
  Network was tagged as a verb with *Games* as its subject below. Currently, we tokenize structurally evident proper names such as Interactive Games Network Inc. as one token. *GAMES PEOPLE PLAY _ Interactive Games Network Inc. will develop new technology that will allow viewers*

at home to compete against one another in live game shows and sports programs.
The tagger was led astray by the following garden path sentence in which *struck* was considered as a finite verb with subject *supertanker*: *Shipping executives said today that a Danish supertanker struck late Thursday in an attack initially blamed on Iran now appears to have been hit by Iraqi warplanes.*
Similarly *held* is considered as a finite verb by the older version of the tagger, and *bodies* was extracted by the filter as its subject in *Jerome Woods, director the Department of Human Services, ordered the two bodies held this week, however, citing a city law that allows the government to withhold the remains if they are to be used in a "tasteless" way.*

- Non-recognition of subclauses — (4 of 12)
  In the simple version of the mark up and filters shown here, this is no recognition of subclauses, which leads to most of the incorrect subject errors as in the following sentences:
  The filters extracted *Poland* as well as *Hitler* as subjects of *allow* in *Those demonstrations marked the anniversary of the 1939 Molotov-Ribbentrop pact under which Stalin agreed to allow Hitler to invade western Poland and Hitler allowed Stalin to invade eastern Poland, Lithuania, Latvia and Estonia.*
  Both *efforts* and *accomplices* were extracted as subjects of *fabricate* in *Fanatics of one calling or another have in recent months tried to disrupt the relief distribution efforts and their accomplices continue to fabricate such unfounded allegations*
  In the following sentence *bales* survives as subject of *incriminating* for two reasons: (i) it is not recognized as being consumed as a direct object, (ii) the conjunction of verbs to the same subject is not recognized, and (iii) *incriminating* is incorrectly seen as a continuation of the preceding verbal chain. *He said U.S. authorities had inspected 300 bales at the company's Elmhurst, Queens, warehouse but did not find incriminating evidence.*
  Another error suffers from many of the same causes, as *lives* is recognized as the subject of *have* in *We just live our own lives and don't have great conflicts.*
- One other error was due to a minor coding bug in the filter.

6 Conclusion

We have presented a sequence of transducers, that composed together, provide a light parser capable of rapidly and robustly extracting common syntactic patterns from part of speech tagged text. By dividing the parsing task into a sequence

```
# Input:   <NG *HeadN corticos-
teroids/NNS NG> <VG did/DOD not/NOT
#     appear/VB to/TO *ActV affect/VB VG> <NG the/AT
#     *HeadN progress/NN of/IN the/AT *PrepN disease/NN NG>
# yields:  corticosteroids/NNS affect/VB <SUBJ

FilterSubj =
 []:? * []:"*HeadN"
      Token
          []:[ ~$ ["<NG"|"VG>"]] []:"*ActV"
           Token
             " <SUBJ":[]
                  []:? * ;
```

Figure 4: A filter for extracting subjects of active verbs. The left context is anything appearing before a token (a word plus part of speech tag) that is labeled with `*HeadN`. The middle context is any sequence not containing any noun or verb group markings that ends in and `*ActV` (active verb) label. The right context is any sequence. The contexts are all transduced to epsilon `[]` by the colon operator. For example, `a:b` transduces `b` into `a`.

```
# Input: <NG *HeadN amyloidosis/NN NG> <VG was/BEDZ
#        *PasV found/VBN VG>
# yields:
#        amyloidosis/NN found/VBN <PDOBJ

FilterPassDobj =
 []:? * []:"*HeadN"
      Token
          []:[ ~$ ["<NG"|"VG>"]] []:"*PasV"
           Token
             " <PDOBJ":[]
                  []:? * ;
```

Figure 5: A filter for extracting passive direct objects in the subject position. The left, right and middle contexts are the same as for the active subject, except for the presence of the `*PasV` label that was inserted by a marking transducer.

of contiguous group marking, head labeling, and non-contiguous extracting filter transducers, we are able to write small compact rules that are easy to create and maintain. Recent finite state calculus advances, such as the longest match operator, allow us to perform all the marking and extraction efficiently within the finite state paradigm.

We presented an evaluation procedure for measuring the precision of relation extraction, which shows that even simple filters produce reliable results over a large corpus. Meanwhile, current research (Chanod and Tapanainen 1996) in finite state grammars is proving that an extremely high level of accuracy is possible using finite state rules.

References

Steven Abney. 1991. Parsing by chunks, in *Principle-Based Parsing*, eds., Steven Abney Robert Berwick and Carol Tenny, Kluwer Academic Publishers, Dordrecht.

Douglas E. Appelt, Jerry R. Hobbs, John Bear, David Israel, and Mabry Tyson. 1993. FASTUS: A finite-state processor for information extraction from real-word text, in *Proceedings IJCAI 93*, Chambery, France.

Jean-Pierre Chanod and Pasi Tapanainen. 1997. Finite-State Based Reductionist Parsing for French. In András Kornai (ed): Extended Finite State Models of Language. Cambridge University Press.

Kenneth Church. 1988. A stochastic parts program and noun phrase parser for unrestricted text, *Proceedings of the 2nd Conference on Applied Natural Language Processing*, 136–143.

Fathi Debili. 1982. *Analyse Syntaxico-Semantique Fondee sur une Acquisition Automatique de Relations Lexicales-Semantiques*, Ph.D. dissertation, University of Paris XI, France.

Gregory Grefenstette. 1992. Use of syntactic context to produce term association lists for text retrieval, in *Proceedings of SIGIR92*, Copenhagen, Denmark, ACM.

Gregory Grefenstette. 1983. *Linguistic treatments applied to Information Retrieval (Traitements Linguistiques Appliquees a la Documentation Automatique)*, University of Paris XI, Orsay, France.

Lauri Karttunen. 1996. Directed replacement, in *Proceedings of the 34rd Annual Meeting of the ACL*, Santa Cruz, CA.

Mehryar Mohri. 1994. Finite-state transducers in language and speech processing, *Computational Linguistics*, **20**(1).