

Augmented letter transducers as morphological analysers in a real machine translation system

Alicia Garrido-Alenda, Mikel L. Forcada and Rafael C. Carrasco

www.interNOSTRUM.com

Departament de Llenguatges i Sistemes Informàtics

Universitat d'Alacant, E-03071 Alacant, Spain

Introduction: interNOSTRUM is an online Spanish–Catalan machine translation system (<http://www.internostrum.com>) that attains great speed through the use of finite-state transducers (FST) in its lexical modules: in particular, the source-language morphological analyser. The FST in interNOSTRUM are a special class of nondeterministic letter transducers (Roche and Schabes 1997) that are obtained by compiling linguistic specifications given in a linguist-readable language.

Augmented letter transducers: Any FST may always be turned into an equivalent letter transducer (Roche and Schabes 1997). The *augmented letter transducers* (ALT) used in our analysers are $T = (Q, L, \delta, q_I, F, G, \xi)$, where Q is a finite set of states; L a set of transition labels $L = (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})$ where Σ is the input alphabet, Γ the output alphabet, and ϵ the empty string; $\delta : Q \times L \rightarrow 2^Q$ the transition function (where 2^Q represents the set of all finite sets of states); $q_I \in Q$ the initial state; $F \subset Q$ the set of *strong final* states; $G \subset Q$ the set of *weak final* states such that $F \cap G = \emptyset$, and $\xi : F \cup G \rightarrow 2^{\Sigma \cup \{\$\}}$ a function that assigns a *validation* (or *lookahead*) set $\xi(q)$ to each final state, with $\$$ the end-of-input marker. According to this definition, state transition labels may therefore be of four kinds: $(\sigma : \gamma)$, meaning that symbol $\sigma \in \Sigma$ is read and symbol $\gamma \in \Gamma$ is written; $(\sigma : \epsilon)$, meaning that a symbol is read but nothing is written; $(\epsilon : \gamma)$, meaning that nothing is read but a symbol is written; and $(\epsilon : \epsilon)$ means that a state transition occurs without reading or writing (used only during FST construction). An ALT is *deterministic* with respect to the alphabet L when $\delta : Q \times L \rightarrow Q$ (although it may still be non-deterministic with respect to Σ ; this is the case in our transducers).

Valid transductions: A string $w' \in \Gamma^*$ is considered to be a *strong* (resp. *weak*) *transduction* of an input string $w \in \Sigma^*$ if (i) there is at least one path from the initial state q_I to a final state $q_F \in F$ (resp. $q_F \in G$) whose

transition labels form the pair $w : w'$ when concatenated and (ii) the input symbol following w is in the validation set $\xi(q_F)$. There may in principle be more than one of such paths for a given transduction; this should be avoided, and is partially eliminated by determinization (see below). On the other hand, there may be more than a valid transduction for a string w (in analysis, this would correspond to the *lexical ambiguity* shown by homographs, surface forms having two or more morphological analyses). In morphological analysis, the symbols in Σ are those found in texts, and the symbols in Γ are those necessary to form the lemmas and those representing morphological information, such as `<noun>`, `<fem>`, etc.

Tokenize-as-you-analyse: Validation sets in ALT enable morphological analysers to both tokenize the input (segment it into surface forms suitable for analysis) and analyse it at the same time. Input is buffered for convenience. The ALT analyses the input by maintaining three analogous sets: a *set of partial transductions* (SPT), a set of *unvalidated transductions* (SUT) and a set of *validated transductions* (SVT). The SPT (updated after each input symbol) contains the last states reached, their associated partial transductions, and the current input position. After reading each symbol a , (i) a new SVT is built from the last SUT by removing all q such that $a \notin \xi(q)$ and their transductions; and (ii) a new SUT is built from the current SPT by removing non-final states and their transductions. Input is read until no more transitions are possible, transductions corresponding to the last SVT are output, and the ALT restarts at the character immediately after the position of the last SVT.

Unconditional strong final states ($q \in F : \xi(q) = \Sigma \cup \{\$\}$) are used to identify tokens such as punctuation marks (`:`, `-`) or apostrophated forms (`d'`, `l'`). Conditional strong final states ($q \in F : \xi(q) \subset \Sigma \cup \{\$\}$) are used, for example, to identify regular words using a validation set with all non-word symbols (so that the analyser does not stop at `bar` when input is `barber`). Finally, weak final states $q \in G$ are used to clip “unknown words” (such as `barnwazz`) and produce some kind of *guessed* (weak) transductions. If a strong transduction is present, weak transductions are ignored.

Multiword units and formatting: This left-to-right, longest-match way of functioning makes it very easy to treat (variable or invariable) multiword units (MWU), for input: if a MWU is not complete, the final state reached will correspond to a smaller unit, which will be clipped and whose transduction will be output (for example, if the dictionary contains “George” and the MWU “George Washington”, when reading “George W. Bush” the MWU “George Washington” will abort at the “.”, “George” will be clipped and the analyser will be ready to process the remaining text, “W. Bush”). Text formatting information (i.e., RTF or HTML tags) is encapsulated prior

to analysis as *superblanks* in “[[...]]”, which the analyser takes as a single blank.

Minimization: ALT may be determinized with respect to the alphabet L and minimized using adapted versions of existing algorithms for finite automata (Hopcroft and Ullman 1979; van de Snepscheut 1993). Transitions labeled $(\epsilon : \epsilon)$ are eliminated through ϵ -closure during determinization. The minimal ALT may still be nondeterministic with respect to input. Since cycles are not very common in our ALTs, we have found the algorithm in van de Snepscheut (1993) to be particularly efficient in our case: in each of its two identical steps the arcs in the LT are reversed, so that the final state(s) are initial and the initial state(s) are final, and the resulting ALT is determinized with respect to L : the LT resulting from the double reversal–determinization process is minimal.

The specification language: The morphological dictionary is a text file where spaces, tabulators and newlines may be freely inserted for legibility, and where comments starting with “#” finish at the end of line. Basically, the dictionary has three main sections:

1. The *symbol declaration* section, where special symbols representing morphological features (such as <fem>) are declared.
2. The *paradigm* section, where inflection *paradigms* are declared and given a name which will be used in square brackets in the dictionary (such as [verbs_in_ducir] for Spanish verbs such as *traducir*, *producir*, *inducir*, etc.). Paradigms may be nested, that is, used to define other paradigms (they are compiled into subtransducers that are then integrated to build the complete transducer). A paradigm is basically a name for a set of alternate *transductions*; valid *transductions* are: *couples* of strings (input and output), such as “(com:comer<verb>)”, paradigm names, such as “[pi1]”, and the concatenation of one or more transductions, such as “(am:amar<verb>) [V1C]”, which represents the transduction resulting from the separate concatenation of inputs and outputs. Couples containing strings of different length are left aligned and padded with ϵ (Karttunen 1993; Garrido et al. 1999). Paradigms may describe endings (suffixes), but also infixes and suffixes.
3. Each *dictionary* section (one per validation set) is simply a large paradigm containing transductions representing the lexical units in the dictionary.

The compiler: The compiler (developed under Linux using `yacc` and `lex`) reads in the morphological dictionary file, combines the partial transducers

corresponding to the declared paradigms into a single transducer using ($\epsilon : \epsilon$) transitions as “glue”, minimizes the resulting transducer and generates a binary file which is read by a generic analyser which may be used on its own or included in a larger application such as a machine translation system.

Experiments: We do not determinize ALTs with respect to input into p -subsequential letter transducers (Mohri 1997; Roche and Schabes 1997) because they are already very fast and because determinizing small test ALTs lead to very large transducers. To better assess the effect of nondeterminism on time complexity we analysed a 1,357,000-word (11,652,000-letter) corpus of Spanish newspaper text; the *average number of live states* in a 50,014-state ALT-based Spanish analyser containing 49,446 entries was 2.8 and the standard deviation, 4.2. The average speed was 5,500 words (47,000 characters) per second on a standard 500-MHz Pentium.

Acknowledgements: Supported by the Caja de Ahorros del Mediterráneo, by the Vice-rectorate for Information Technologies of the Universitat d’Alacant and by the Spanish *Comisión Interministerial de Ciencia y Tecnología* through grant TIC2000-1599-C02-02.

References

- Garrido, A., Iturraspe, A., Montserrat, S., Pastor, H., and Forcada, M. (1999). A compiler for morphological analysers and generators based on finite-state transducers. *Procesamiento del Lenguaje Natural*, (25):93–98.
- Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation*. Addison–Wesley, Reading, MA.
- Karttunen, L. (1993). Finite-state lexicon compiler. Technical Report Technical Report ISTL-NLTT-1993-04-02, Xerox Palo Alto Research Center, Palo Alto, California.
- Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.
- Roche, E. and Schabes, Y. (1997). Introduction. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, pages 1–65. MIT Press, Cambridge, Mass.
- van de Snepscheut, J. (1993). *What computing is all about*. Springer-Verlag, New York.