Pre-processing Closed Captions for Machine Translation

Davide Turcato Fred Popowich Paul McFetridge
Devlan Nicholson Janine Toole
Natural Language Laboratory, School of Computing Science, Simon Fraser University
8888 University Drive, Burnaby, British Columbia, V5A 1S6, Canada
and
gavagai Technology Inc.
P.O. 374, 3495 Cambie Street, Vancouver, British Columbia, V5Z 4R3, Canada
{turk,popowich,mcfet,devlan,toole}@cs.sfu.ca

Abstract

We describe an approach to Machine Translation of transcribed speech, as found in closed captions. We discuss how the colloquial nature and input format peculiarities of closed captions are dealt with in a pre-processing pipeline that prepares the input for effective processing by a core MT system. In particular, we describe components for proper name recognition and input segmentation. We evaluate the contribution of such modules to the system performance. The described methods have been implemented on an MT system for translating English closed captions to Spanish and Portuguese.

1 Introduction

Machine Translation (MT) technology can be embedded in a device to perform real time translation of closed captions included in TV signals. While speed is one factor associated with the construction of such a device, another factor is the language type and format. The challenges posed by closed captions to MT can be attributed to three distinct characteristics:

Firstly, closed captions are transcribed speech. Although closed captions are not a completely faithful transcription of TV programs, they render spoken language and therefore the language used is typically colloquial (Nyberg and Mitamura, 1997). They contain many of the phenomena which characterize spoken language: interjections, repetitions, stuttering, ellipsis, interruptions, hesitations. Linguistically and stylistically they differ from written language: sentences are shorter and poorly structured, and contain idiomatic expressions, ungrammaticality, etc. The associated difficulties stem from the inherently colloquial nature of closed captions, and, to different degrees, of all forms of transcribed speech (Hindle, 1983). Such difficulties require a different approach than is taken for written documents.

Secondly, closed captions come in a specific format, which poses problems for their optimal processing. Closed-captioners may often split a single utterance between two screens, if the character limit for a screen has been exceeded. The split is based on consideration about string length, rather than linguistic considerations, hence it can happen at non-constituent boundaries (see Table 1), thus making the real time processing of the separate segments problematic. Another problem is that captions have no upper/lower case distinction. This poses challenges for proper name recognition since names cannot be identified by an initial capital. Additionally, we cannot rely on the initial uppercase letter to identify a sentence initial word. This problematic aspect sets the domain of closed captions apart from most text-to-text MT domains, making it more akin, in this respect, to speech translation systems. Although, from a technical point of view, such input format characteristics could be amended, most likely they are not under a developer's control, hence they have to be presumed.

Thirdly, closed captions are used under operational constraints. Users have no control over the speed of the image or caption flow so (s)he must comprehend the caption in the limited time that the caption appears on the screen. Accordingly, the translation of closed captions is a "time-constrained" application, where the user has limited time to comprehend the system output. Hence, an MT system should produce translations comprehensible within the limited time available to the viewer.

In this paper we focus on the first two factors, as the third has been discussed in (Toole et al., 1998). We discuss how such domain-

```
good evening. i'm jim lehrer.
on the "newshour" tonight, four members of congress debate the
u.n. deal with iraq; paul solman tells the troubled story of
indonesia's currency; mark
shields and paul gigot analyze the political week;
and elizabeth farnsworth explains how the universe is getting
larger.
```

Table 1: Closed caption script fragment.

dependent, problematic factors are dealt with in a pre-processing pipeline that prepares the input for processing by a core MT system. The described methods have been implemented for an MT system that translates English closed captions to Spanish and Portuguese. All the examples here refer to the Spanish module.

2 Pre-processing design

Input pre-processing is essential in an embedded real time system, in order to simplify the core processing and make it both time- and memory-effective. In addition to this, we followed the guideline of separating domain-dependent processes and resources from general purpose ones. On the one hand, grammars and lexicons are costly resources. It would be desirable for them to be domain-independent and portable across different domains, as well as declarative and bidirectional. On the other hand, a domain with distinctive characteristics requires some specific treatment, if a system aims at robustness. We decided to have a domain independent core MT system, locating the domain dependent processing in a pipeline of low-level components, easy to implement, aiming at fast and robust processing and using limited linguistic knowledge.

We use declarative and bidirectional grammars and lexicons. The lexicalist approach is indeed suitable to the closed caption domain, e.g. in terms of its capability of handling loosely structured or incomplete sentences. Also, the linguistic resources are geared towards this domain in terms of grammatical and lexical coverage. However, our system architecture and formalism make them equally usable in any other domain and translation direction, as the linguistic knowledge therein contained is valid in any domain. For the architecture we refer the reader to (Popowich et al., 1997). In the rest of this paper we focus on the pre-processing module and how it deals with the issues discussed in the introduction.

The task of the pre-processing pipeline is to make the input amenable to a linguistically-principled, domain independent treatment. This task is accomplished in two ways:

1. By *normalizing* the input, i.e. removing noise, reducing the input to standard typographical conventions, and also restructuring and simplifying it, whenever this can be done in a reliable, meaning-preserving way.
2. By *annotating* the input with linguistic information, whenever this can be reliably done with a shallow linguistic analysis, to reduce input ambiguity and make a full linguistic analysis more manageable.

Figure (1) shows the system architecture, with a particular emphasis on the pre-processing pipeline. The next section describes the pipeline up to tagging. Proper name recognition and segmentation, which deal more specifically with the problems described in the introduction, are discussed in further sections.

3 Normalization and tagging

The label *normalization* groups three components, which clean up and tokenize the input.

The *text-level normalization* module performs operations at the string level, such as removing extraneous text and punctuation (e.g. curly brackets, used to mark off sound effects), or removing periods from abbreviations. E.g.:

(1) `"I went to high school in the u.s."`
⇒
`"I went to high school in the usa."`

The *tokenizer* breaks a line into words. The *token-level normalization* recognizes and annotates tokens belonging to special categories

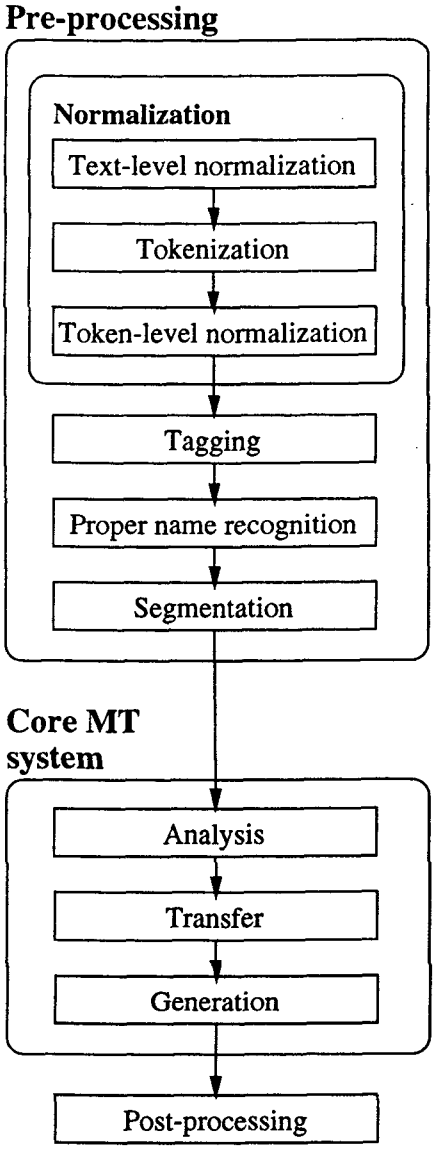

Figure 1: System architecture.

(times, numbers, etc.), expands contractions, recognizes, normalizes and annotates stutters (e.g. *b-b-b-bright*), identifies compound words and converts number words into digits. E.g.:

(2) `"I" "went" "to" "high" "school" "in" "the" "usa" "." ⇒ "I" "went" "to" "high school" "in" "the" "usa" "."`

(3) `"W-wh-what's" "that" "?" ⇒ "what"/stutter "is" "that" "?"`

Note that annotations associated with tokens are carried along the entire translation process, so as to be used in producing the output (e.g. stutters are re-inserted in the output).

The *tagger* assigns parts of speech to tokens. Part of speech information is used by the subsequent pre-processing modules, and also in parsing, to prioritize the most likely lexical assignments of ambiguous items.

4 Proper name recognition

Proper names are ubiquitous in closed captions (see Table 1). Their recognition is important for effective comprehension of closed captions, particularly in consideration of two facts: (i) users have little time to mentally rectify a mistranslation; (ii) a name can occur repeatedly in a program (e.g. a movie), with an annoying effect if it is systematically mistranslated (e.g. a golf tournament where the golfer named Tiger Woods is systematically referred to as *los bosques del tigre*, lit. '*the woods of the tiger*'). Name recognition is made harder in the closed caption domain by the fact that no capitalization information is given, thus making unusable all methods that rely on capitalization as the main way to identify candidates (Wolinski et al., 1995) (Wacholder et al., 1997). For instance, an expression like '`mark shields`', as occurs in Table (1), is problematic in the absence of capitalization, as both '`mark`' and '`shields`' are three-way ambiguous (proper name, common noun and verb). Note that this identical problem may be encountered if an MT system is embedded in a speech-to-speech translation as well. This situation forced us to explore different ways of identifying proper names.

The goal of our recognizer is to identify proper names in a tagged line and annotate them accordingly, in order to override any other possible lexical assignment in the following modules. The recognizer also overrides previous tokenization, by possibly compounding two or more tokens into a single one, which will be treated as such thereafter. Besides part of speech, the only other information used by the recognizer is the lexical status of words, i.e. their ambiguity class (i.e. the range of possible syntactic categories it can be assigned) or their status as an unknown word (i.e. a word that is not in the lexicon). The recognizer scans an input line from left to right, and tries to match

each item against a sequences of patterns. Each pattern expresses constraints (in terms of word, part of speech tag and lexical status) on the item under inspection and its left and right contexts. Any number of items can be inspected to the left and right of the current item. Such patterns also make use of regular expression operators (conjunction, disjunction, negation, Kleene star). For instance (a simplified version of) a pattern might look like the following:

(4) `[the/DET (NOUN|ADJ)*] X' [~NOUN]`

where we adopt the convention of representing words by lowercase strings, part of speech tags by uppercase strings and variables by primed Xs. The left and right context are enclosed in square brackets, respectively to the left and right of the current item. They can also contain special markers for the beginning and end of a line, and for the left or right boundary of the proper name being identified. This way tokenization can be overridden and separate tokens joined into a single name. Constraints on the lexical status of items are expressed as predicates associated with pattern elements, e.g.:

(5) `proper_and_common(X')`

A pattern like the one above (4-5) would match a lexically ambiguous proper/common noun preceded by a determiner (with any number of nouns or adjectives in between), and not followed by a noun (e.g. '`the bill is...`'). Besides identifying proper names, some patterns may establish that a given item is not a name (as in the case above). A return value is associated with each pattern, specifying whether the current match is or is not a proper name. Once a successful match occurs, no further patterns are tried. Patterns are ordered from more to less specific. At the bottom of the pattern sequence are the simplest patterns, e.g.:

(6) `( [] X' [] ), proper_and_common(X')`
`⇒ yes`

which is the default assignment for words like '`bill`' if no other pattern matched. However (6) is overridden by more specific patterns like:

(7) `( [X''] X' [] ),`
`proper_and_common(X'), common(X'')`
`⇒ no`

(8) `( [X''] X' [] ),`
`proper_and_common(X'), proper(X'')`
`⇒ yes`

The former pattern covers cases like '`telecommunications bill`', preventing '`bill`' from being interpreted as a proper name, the latter covers cases like '`damian bill`', where '`bill`' is more likely to be a name.

In general, the recognizer tries to disambiguate lexically ambiguous nouns or to assign a category to unknown words on the basis of the available context. However, in principle any word could be turned into a proper name. For instance, verbs or adjectives can be turned into proper names, when the context contains strong cues, like a title. Increasingly larger contexts provide evidence for more informed guesses, which override guesses based on narrower contexts. Consider the following examples that show how a word or expression is treated differently depending on the available context. Recognized names are in italics.

(9) *`bill`* `!`

(10) `the bill is ...`

(11) `the` *`bill`* `clinton is ...`

(12) `the` *`bill clinton`* `administration is ...`

The lexically ambiguous `bill`, interpreted as a proper name in isolation, becomes a common noun if preceded by a determiner. However, the interpretation reverts to proper name if another noun follows. Likewise the unknown word `clinton` is (incorrectly) interpreted as a common noun in (11), as it is the last item of a noun phrase introduced by a determiner, but it becomes a proper name if another noun follows.

We also use a *name memory*, which patterns have access to. As proper names are found in an input stream, they are added to the name memory. A previous occurrence of a proper name is used as evidence in making decisions about further occurrences. The idea is to cache names occurred in an 'easy' context (e.g. a name preceded by a title, which provides strong evidence for its status as a proper name), to use them later to make decisions in 'difficult' contexts, where the internal evidence would not be sufficient to support a proper name interpretation.

Hence, what typically happens is that the same name in the same context is interpreted differently at different times, if previously the name has occurred in an 'easy' context and has been memorized. E.g.:

(13) `the individual title went to tiger woods.`

...

`mr.` *`tiger woods`* `struggled today with a final round 80.`

⇒ name-memory

...

`the short well publicized professional life of` *`tiger woods`* `has been an open book.`

The name memory was designed to suit the peculiarity of closed captions. Typically, in this domain proper names have a low dispersion. They are concentrated in sections of an input stream (e.g. the name of the main characters in a movie), then disappear for long sections (e.g. after the movie is over). Therefore, a name memory needs to be reset to reflect such changes. However, it is problematic to decide when to reset the name memory. Even if it was possible to detect when a new program starts, one should take into account the possible scenario of an MT system embedded in a consumer product, in which case the user might unpredictably change channel at any time. In order to keep a name memory aligned with the current program, without any detection of program changes, we structured the name memory as a relatively short queue (first in, first out). Every time a new item is added to the end of the queue, the first item is removed and all the other items are shifted. Moreover, we do not check whether a name is already in the memory. Every time a suitable item is found, we add it to the memory, regardless of whether it is already there. Hence, the same item could be present twice or more in the memory at any given time. The result of this arrangement is that a name only remains in the memory for a relatively short time. It can only remain for a longer time if it keeps reappearing frequently in the input stream (as typically happens), otherwise it is removed shortly after it stopped appearing. In this way, the name memory is kept aligned with the current program, with only a short transition period, during which names no longer pertinent are still present in the memory, before getting replaced by pertinent ones.

	# of items
Proper names correctly identified	152
False positives	8
False negatives	57

Table 2: Name recognition evaluation results.

The recognizer currently contains 63 patterns. We tested the recognizer on a sample of 1000 lines (5 randomly chosen continuous fragments of 200 lines each). The results, shown in table (2), illustrate a recall of 72.7% and a precision of 95.0%. These results reflect our cautious approach to name recognition. Since the core MT system has its own means of identifying some proper names (either in the lexicon or via default assignments to unknown words) we aimed at recognizing names in pre-processing only when this could be done reliably. Note also that 6 out of the 8 false positives were isolated interjections that would be better left untranslated (e.g. `pffoo`, `el smacko`), or closed captioner's typos (e.g. `yo4swear`).

5 Segmentation

Segmentation breaks a line into one or more segments, which are passed separately to subsequent modules (Ejerhed, 1996) (Beeferman et al., 1997). In translation, segmentation is applied to split a line into a sequence of translationally self-contained units (Lavie et al., 1996). In our system, the *translation units* we identify are syntactic units, motivated by cross-linguistic considerations. Each unit is a constituent that can be translated independently. Its translation is insensitive to the context in which the unit occurs, and the order of the units is preserved by translation.

One motivation for segmenting is that processing is faster: syntactic ambiguity is reduced, and backtracking from a module to a previous one does not involve re-processing an entire line, but only the segment that failed. A second motivation is robustness: a failure in one segment does not involve a failure in the entire line, and error-recovery can be limited

only to a segment. Further motivations are provided by the colloquial nature of closed captions. A line often contains fragments with a loose syntactic relation to each other and to the main clause: vocatives, false starts, tag questions, etc. These are most easily translated as individual segments. Parenthetical expressions are often also found in the middle of a main clause, thus making complete parses problematic. However, the solution involves a heavier intervention than just segmenting. Dealing with parentheticals requires restructuring a line, and reducing it to a 'normal' form which ideally always has parenthetical expressions at one end of a sentence (under the empirical assumption that the overall meaning is not affected). We will see how this kind of problem is handled in segmentation. A third motivation is given by the format of closed captions, with input lines split across non-constituent boundaries. One solution would be delaying translation until a sentence boundary is found, and restructuring the stored lines in a linguistically principled way. However, the requirements of real time translation (either because of real time captioning at the source, or because the MT system is embedded in a consumer product), together with the requirement that translations be aligned with the source text and, above all, with the images, makes this solution problematic. The solution we are left with, if we want lines to be broken along constituent boundaries, is to further segment a sentence, even at the cost of sometimes separating elements that should go together for an optimal translation. We also argued elsewhere (Toole et al., 1998) that in a time-constrained application the output grammaticality is of paramount importance, even at the cost of a complete meaning equivalence with the source. For this reason, we also simplify likely problematic input, when a simplification is possible without affecting the core meaning.

To sum up, the task at hand is broader than just segmentation: re-ordering of constituents and removal of words are also required, to syntactically 'normalize' the input. As with name recognition, we aim at using efficient and easy to implement techniques, relying on limited linguistic information. The segmenter works by matching input lines against a set of templates represented by pushdown transducers. Each transducer is specified in a fairly standard way (Gazdar and Mellish, 1989, 82), by defining an initial state, a final state, and a set of transitions of the following form:

(14) ⟨`State1,State2,Label,Transducer`⟩

Such a transition specifies that `Transducer` can move from `State1` to `State2` when the input specified by `Label` is found. `Label` can be either a pair ⟨`InputSymbol, OutputSymbol`⟩ or the name of another transducer, which needs to be entirely traversed for the transition from `State1` to `State2` to take place. An input symbol is a ⟨`Word, Tag`⟩ pair. An output symbol is an integer ranging from 0 to 3, specifying to which of two output segments an input symbol is assigned (0 = neither segment, 3 = both segments, 1 and 2 to be interpreted in the obvious way). The output codes are then used to perform the actual split of a line. A successful match splits a line into two segments at most. However, on a successful split, the resulting segments are recursively fed to the segmenter, until no match is found. Therefore, there is no limit to the number of segments obtained from an input line. The segmenter currently contains 37 top-level transducers, i.e. segmenting patterns. Not all of them are used at the same time. The implementation of patterns is straightforward and the segmenter can be easily adapted to different domains, by implementing specific patterns and excluding others. For instance, a very simple patterns split a line at every comma, a slightly more sophisticated one, splits a line at every comma, unless tagged as a coordination; other patterns split a final adverb, interjection, prepositional phrase, etc.

Note that a segment can be a discontinuous part of a line, as the same output code can be assigned to non-contiguous elements. This feature is used, e.g., in restructuring a sentence, as when a parenthetical expression is encountered. The following example shows an input sentence, an assignment, and a resulting segmentation.

(15) `this, however, is a political science course.`

(16) `this/2 ,/0 however/1 ,/1 is/2 a/2 political/2 science/2 course/2.`

(17) `1. however ,`

2. this is a political science
course .

We sometimes use the segmenter's ability to simplify the input, e.g. with adverbs like *just*, which are polysemous and difficult to translate, but seldom contribute to the core meaning of a sentence.

6 Performance

We ran a test to evaluate how the recognizer and segmenter affected the quality of translations. We selected a sample of 200 lines of closed captioning, comprising four continuous sections of 50 lines each. The sample was run through the MT system twice, once with the recognizer and segmenter activated and once without. The results were evaluated by two native Spanish speakers. We adopted a very simple evaluation measure, asking the subjects to tell whether one translation was better than the other. The translations differed for 32 input lines out of 200 (16%). Table (3) shows the evaluation results, with input lines as the unit of measurement. The third column shows the intersection of the two evaluations, i.e. the evaluations on which the two subjects agreed. The three rows show how often the translation was better (i) with pre-processing, (ii) without pre-processing, or (iii) no difference could be appreciated.

The results show a discrepancy in the evaluations. One evaluator also pointed out that it is hard to make sense of transcribed closed captions, without the audio-visual context. These two facts seem to point out that an appropriate evaluation should be done in the operational context in which closed captions are normally used. Still, the intersection of the subjects' evaluations shows that pre-processing improves the output quality. In three of the four cases where the two evaluators agreed that pre-processing yielded a worse result, the worse performance was due to an incorrect name recognition or segmentation. However, in two of the three cases, the original problem was an incorrect tagging.

Note that even when the name recognizer and segmenter are off, the system can identify some names, and recover from translation failures by piecing together translations of fragments. Therefore, what was being tested was not so much name recognition and segmenting per se, but the idea of having separate modules for such tasks in the system front end.

Finally, the test did not take into account speed, as we set higher time thresholds than an embedded application would require. Since segmentation reduces processing time, it is also expected to reduce the impact of tighter time thresholds, all other things being equal.

We are planning to conduct an operational evaluation of the system. The goal is to evaluate the system output in its proper visual context, and compare the results with parallel results for human translated closed captions. Different groups of participants will watch a video with either human- or machine-translated subtitles, and complete a questionnaire based on the subtitles in the video. The questionnaire will contain a set of questions to elicit the subject's assessment on the translation quality, and a set of questions to assess the subject's level of comprehension of the program.

7 Conclusion

It is apparent that the peculiarity of closed captions, both in terms of transcribed speech characteristic and constraints due to the input format, require an ad hoc treatment, considerably different from the approaches suitable for written documents. Yet the knowledge about a language (or the bilingual knowledge about a language-pair) is largely invariant across different applications domains and should therefore be portable from one application domain to another. The architecture we have proposed strives to combine the need for domain independent linguistic resources and linguistically principled methods with the need for robust MT systems tuned to real world, noisy and idiosyncratic input, as encountered when embedding MT in real world devices.

In terms of adequacy, a standard evaluation and a comparison among different MT systems from different domains is hard, as the adequacy of a system depends on its application (Church and Hovy, 1993). This is even truer with closed captions, where the use of translation output is heavily influenced by operational constraints (time constraints, the presence of images, sound, etc.). In some cases such constraints may place a heavier burden on a system (e.g. the time constraint), in some other cases

	Judge 1	Judge 2	Both agreed
Better with pre-processing	21	16	15
Better without pre-processing	4	12	4
No difference	7	4	3

Table 3: Evaluation results.

they can make an imperfect translation acceptable (e.g. the presence of images and sounds). We did not attempt an assessment in absolute terms, which we believe should take into account the operational environment and involve real-world users. More modestly, we aimed at showing that our pre-processing techniques provide an improvement in performance.

Our work on closed captions also shows that the challenges coming from this domain, even in terms on low-level issues of input format, can lead to interesting developments of new linguistic techniques. We believe that our solutions to specific problems (namely, proper name recognition and segmentation) in the closed caption domain bear relevance to a wider context, and offer techniques that can be usefully employed in a wider range of applications.

References

Doug Beeferman, Adam Berger, and John Lafferty. 1997. Text segmentation using exponential models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-2)*, Providence, USA.

Kenneth W. Church and Eduard H. Hovy. 1993. Good applications for crummy machine translation. *Machine Translation*, 8:239–258.

Eva Ejerhed. 1996. Finite state segmentation of discourse into clauses. In A. Kornai, editor, *Proceedings of the ECAI-96 Workshop Extended Finite State Models of Language*, Budapest,Hungary.

Gerald Gazdar and Christopher S. Mellish. 1989. *Natural Language Processing in PROLOG: an Introduction to Computational Linguistics*. Addison-Wesley Publishing Company, Wokingham, England.

Donald Hindle. 1983. Deterministic parsing of syntactic non-fluencies. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL-83)*, pages 123–128, Cambridge, Massachusetts, USA.

Alon Lavie, Donna Gates, Noah Coccaro, and Lori Levin. 1996. Input segmentation of spontaneous speech in janus: a speech-to-speech translation system. In *Proceedings of ECAI-96 Workshop on Dialogue Processing in Spoken Language Systems*, Budapest,Hungary.

Eric Nyberg and Teruko Mitamura. 1997. A real-time MT system for translating broadcast captions. In *Proceedings of the Sixth Machine Translation Summit*, pages 51–57, San Diego, California, USA.

Fred Popowich, Davide Turcato, Olivier Laurens, Paul McFetridge, J. Devlan Nicholson, Patrick McGivern, Maricela Corzo-Pena, Lisa Pidruchney, and Scott MacDonald. 1997. A lexicalist approach to the translation of colloquial text. In *Proceedings of the 7th International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 76–86, Santa Fe, New Mexico, USA.

Janine Toole, Davide Turcato, Fred Popowich, Dan Fass, and Paul McFetridge. 1998. Time-constrained Machine Translation. In *Proceedings of the Third Conference of the Association for Machine Translation in the Americas (AMTA-98)*, pages 103–112, Langhorne, Pennsylvania, USA.

Nina Wacholder, Yael Ravin, and Misook Choi. 1997. Disambiguation of proper names in texts. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, pages 202–208, Washington, DC, USA. Association for Computational Linguistics.

Francis Wolinski, Frantz Vichot, and Bruno Dillet. 1995. Automatic processing of proper names in texts. In *Proceedings of the 7th Conference of the European Chapter of the Asscociation for Computational Linguistics (EACL-95)*, pages 23–30, Dublin, Ireland.