Comments on Kornai

Emmanuel Roche
Mitsubishi Electric Research Laboratories
201 Broadway
Cambridge, MA 02179
roche@merl.com

As observed in the announcement of this workshop, the bulk of actual language processing is in fact of finite-state nature (see [6]). A. Kornai presents here an information extraction task for which finite-state processing outperforms more powerful formalisms both in terms of accuracy and, not surprisingly, in terms of speed efficiency. The key for building accurate devices is to use a very simple formalism which allows the linguist to focus on syntactic constraints in a natural way and at the right level of lexical detail. Kornai introduces here Vectorized FSAs (VFSAs) which are shown to be very well adapted for information extraction from natural language texts.

As pointed out by the author, VFSAs are closely related to Register Vector Grammars (RVGs) [1]. [1] argued that the use of vector based FSA allows more compact representations of language related structures while retaining the runtime efficiency of finite-state processing. This earlier work was very intriguing but it also posed a number of unanswered questions. From a practical point of view it was not clear whether this formalism could be used to design realistic size grammars. The examples given in [1] were not very convincing in that respect, experiments were performed on small size grammars (Tomita's 220 rule grammar).

Kornai's first contribution is to show that vectorized FSA is in fact a natural answer to a practical language problem. The ability to use several layers of information for a given word (the word itself, morphological features, whether it belongs to a specific lexicon, etc) is indeed crucial for the design of information extraction devices.

From this information extraction point of view, Kornai's work is related, among others, to [7] and [4] for which VFSAs also provide a possible formal framework. In fact, both [7, 4], use the simplicity of automata design to draw precise restricted grammars. Their grammars, as well as Kornai's, are restricted in two ways: they focus on very specific problems within restricted domains and they don't require the sentence to be parsed in full. While [7] and [4] initially built finite-state automata whose transitions were purely lexical, they later added the possibility for a transition to match various types of information (the word, the canonical form of the word, morphological features, whether it belongs to special lexicon or conjunctions of any of the above). While their work has also been formalized in a different way [5], VFSAs provide a natural and elegant framework for this type of task.

From a more formal point of view, VFSAs as well as RVG, relate to more traditional finite-state automata and finite-state transducers in many ways. The first similarity is between the runtime application of VFSAs and the lookup of an input sequence into the intersection of several automata computed dynamically. In fact, in the particular case were the features are independent, the vectorized FSAs are similar to the intersection of several automata, each applying to each feature independently. It is therefore an important contribution from Kornai to show that features should be interdependent; this was not so clear from Blank's original paper in which the compactness of RVG often comes from the fact that while RVGs implicitly represented intersections, these intersections were not precomputed statically but rather they were computed dynamically at runtime which is where most of the compactness came from.

It is however unlikely that various linguist practitioners will switch from their formalism to VFSA for the sole formal elegance so it is probably important to be able to use VFSAs in conjunction with more traditional finite-state machines. Moreover, vector based FSAs are compact and natural in particular situations while more traditional FSMs remain very well adapted for a huge number of simple tasks; it would be natural to use both simultaneously. Obviously it is possible to do simple things like computing the intersection between a vectorized FSA and a classical FSA. However, in doing so, state information and vector values have to be handled independently. It seems that FSAs could be extended with vectors by considering the state as one variable and by adding other features (i.e. variables, dimensions). From this perspective, VFSAs are related to push-down automata and push-down transducers with bounded stack as well as to non-recursive RTNs. Doing so might provide compactness and simplicity. In fact, an automaton like the one of Figure 1 in which each letter is an abbreviation of a possibly big automaton, could become more compact by merging states 2 and 3 and 4 and 5 while adding an additional feature (i.e. variable) that distinguishes the states of F originating from 1 and those coming from 2. This could be done at design time, like in the information extraction task presented here or as a postprocessing compaction task. As a postprocessing task, this could be done by representing the initial automaton as a string whose alphabet is a set of pairs of letters and offsets to the arrival state[1] The

[1] This automaton would look like :$(A, +1)$ $(END, 0)$ $(B + 2)$ $(END, 0)$ $(F, +2)$ $(END, 0)$ $(F, +2)$ $(END, 0)$ $(A, +2)$ $(END, 0)$ $(B, +1)$ $(END, 0)$ $(END, FINAL)$.

© 1996 E. Roche
Proceedings of the ECAI 96 Workshop
Extended Finite State Models of Language
Edited by A. Kornai.

methods used to detect repeated factors in text [2, 3] could be applied here to detect long repeated automata such as F in the example. Operations like determinization might also benefit from inserting additional variables to control space expansion.

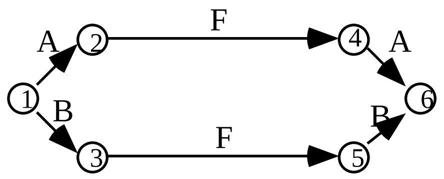
$$\begin{tikzcd}
 & 2 \arrow[rr, "F"] & & 4 \arrow[dr, "A"] & \\
1 \arrow[ur, "A"] \arrow[dr, "B"'] & & & & 6 \\
 & 3 \arrow[rr, "F"] & & 5 \arrow[ur, "B"'] &
\end{tikzcd}$$

Figure 1. Example of automaton

Finally, several other questions come to mind. First, the power of finite-state computing comes from several properties; among them: runtime efficiency and a large number of formal operations which allow both to build complexes devices from simple ones and to check formal properties of a given finite-state machine. Kornai's contribution shows that VFSAs achieve fast run-time performances (does it scale to very large VFSAs?), what about the possibility of building complex devices from simple ones (although intersecting VFSAs is straightforward).

A practical question: how difficult is it to decide what set of variables should be used. Also, a critical point is to be able to debug large system, how difficult is it to detect design mistakes? Do VFSA have a VFST counterpart?

REFERENCES

[1] Glenn David Blank, 'A finite and real-time processor for natural language', *Communications of the A.C.M.*, **32**(10), 1174–1189, (1989).

[2] A. Blumner, J. Blumner, D. Haussler, A. Ehrenfeucht, M.T. Chen, and J. Seiferas, 'The smalleest automaton recognizing the subwords of a text', *Theoretical Computer Science*, **40**, 31–55, (1985).

[3] Maxime Crochemore, 'Transducers and repetitions', *Theoretical Computer Science*, **45**, 63–86, (1986).

[4] Maurice Gross, 'The construction of local grammars', in *Finite-State Devices for Natural Language Processing*, eds., Emmanuel Roche and Yves Schabes, MIT Press, (1996). Forthcoming.

[5] Eric Laporte, 'Experiences in lexical disambiguation using local grammars', in *COMPLEX'94, Proceedings, Readings in Computational Lexicography, Budapest*, (1994).

[6] *Finite-State Devices for Natural Language Processing*, eds., Emmanuel Roche and Yves Schabes, MIT Press, 1996. Forthcoming.

[7] Max Silberztein, *Dictionnaires Electroniques et Analyse Lexicale du Français— Le Système INTEX*, Masson, 1993.