SYNCHRONETICS: MUC-3 TEST RESULTS AND ANALYSIS

James Mayfield
Computer Science Dept.
University of Maryland, Baltimore County
Baltimore, MD 21228-5398
mayfield@umbc3.umbc.edu
(301) 455-3099

Edwin Addison
Synchronetics, Inc.
3700 Koppers St., Suite 131
Baltimore MD 21227
76366.1115@compuserve.com
(301) 644-2400

RESULTS

The Synchronetics entry in the MUC-3 competition is a full-parser, semantic net-based system written in C. Our system attempts to fill the first four slots of each template and, in some cases, the three perpetrator slots and the **human-target-ids** slot. The Synchronetics system achieved the following official scores on the tst2 corpus:

SLOT	REC	PRE	OVG	FAL
template-id	31	51	49	
incident-date	17	55	0	
incident-type	19	61	0	0
category	24	56	28	11
indiv-perps	0	*	*	
org-perps	0	*	*	
perp-confidence	0	*	*	0
phys-target-ids	0	*	*	
phys-target-num	0	*	*	
phys-target-types	0	*	*	0
human-target-ids	2	100	0	
human-target-num	0	*	*	
human-target-types	0	*	*	0
target-nationality	0	*	*	0
instrument-types	0	*	*	0
incident-location	0	*	*	
phys-effects	0	*	*	0
human-effects	0	*	*	0
MATCHED ONLY	18	55	25	
MATCHED/MISSING	7	55	25	
ALL TEMPLATES	7	35	53	
SET FILLS ONLY	7	58	14	0

These official results were achieved despite a system bug that caused almost half of the roughly 1400 sentences

in the corpus to be thrown away without being processed at all. The bug arose because a buffer that was supposed to be 200 items long was inadvertantly changed to be 20 items long. With this bug fixed, we achieved the following unofficial scores:

SLOT	REC	PRE	OVG	FAL
template-id	48	49	51	
incident-date	22	47	0	
incident-type	34	69	0	0
category	36	54	27	18
indiv-perps	0	*	*	
org-perps	0	*	*	
perp-confidence	0	*	*	0
phys-target-ids	0	*	*	
phys-target-num	0	*	*	
phys-target-types	0	*	*	0
human-target-ids	2	62	0	
human-target-num	0	*	*	
human-target-types	0	*	*	0
target-nationality	0	*	*	0
instrument-types	0	*	*	0
incident-location	0	*	*	
phys-effects	0	*	*	0
human-effects	0	*	*	0
MATCHED ONLY	20	54	26	
MATCHED/MISSING	10	54	26	
ALL TEMPLATES	10	33	55	
SET FILLS ONLY	11	62	13	0

We do not at present have any settings that can be modified to alter the recall/precision tradeoff.

ALLOCATION OF TIME

The most time-consuming of our activities were the development of the semantic net software, and the development of the phrase and sentence interpreters. Next came the development of the grammars for the two parsers, and the template generation software. Development of the dictionary was quite rapid, thanks to our automatic acquisition software. The activity we spent the least amount of time on was the encoding of world knowledge into the knowledge base.

LIMITING FACTORS

Our primary limiting factor was the tenuous nature of the lines of communication between our team members. With personnel spread across six different sites, we were forced to rely on weekly meetings to resolve problems that would ordinarily be cleared up on a daily basis if everyone were working at the same site.

The second limiting factor for our system was the amount of time we had available to us. Most of the system was developed from scratch (only the NL-Builder software was written prior to the commencement of our project). We had only a few weeks between the time we were first able to process 100 texts and the time that the final test was due. Thus, we were unable to be as careful as we would have liked to be in the development of the final system configuration.

The third limiting factor for our system was the lack of a detailed and well thought out world model. We did most of our development using a very small world model that had fewer than 50 concepts. Just before running the final test, we quickly developed and switched to a world model containing almost 900 concepts. However, we did not have time to examine it closely before running the test. We believe that we could considerably increase our system's performance for the slots we are currently filling by improving the world model.

SUCCESSES AND FAILURES

Our biggest successes were the development of the dictionary, and the speed of the parsers. Our automatic acquisition software allowed us to obtain a dictionary of 10000 words quite painlessly. Together, both parsers took less than one hour to process every word of all 100 texts, running on a DecStation 3100.

Our biggest failures were lack of development of the knowledge base and the speed of the semantic net I/O routines. Our knowledge base was a last-minute effort, which significantly degraded system performance. The semantic net I/O routines were slow enough to be the main time drain on the three non-parser components. For these reasons the knowledge base and the semantic net I/O routines are our prime candidates to be rewritten.

REUSABILITY

We expect to be able to reuse all system components except for the template generator in other projects. We are currently working on a project to automatically convert linear text to hypertext. We plan to use our MUC system as the front end to the conversion system. This will require only the development of software to generate hypertext links based on the semantic net built by the MUC system, and the development of a new knowledge base for the target domain.

LESSONS LEARNED

Participation in MUC-3 has led us to the following conclusions:

- Our software engineering paradigm (which is thrust upon us by virtue of the fact that our personnel are spread out across several sites) is a poor one, but it is not fatal.
- Several person-years of work is needed to build a parser-based system that has the *potential* to do well at the MUC task. Even then, a weakness in any component can easily reduce the system's abilities to those of a stupid keyword-matching system.

- Evaluation of natural language processing systems through a MUC-like competition is significantly complicated by the fact that it is hard to know what is being measured. Nonetheless, we believe that our architecture will be excellent for evaluation of the various components of a natural language processing system, because we will be able to mix and match the components that go into our system. We will have this flexibility because each of our components is a stand-alone program, and because all of our programs communicate with each other via the same semantic net representation language. For example, if we develop both a script-processing component and an anaphora component, we will be able to put them together in either order, or omit either or both of them. By comparing the results of each of these configurations, we will gain insight into the relative merits of these two forms of processing.