Abstract

This paper sketches a new implementation of an engine that extracts predicate semantics from sentence parses. It is implemented in a cognitive modeling framework and employs operators to interpret content from portions of the parse. It differs from a prior implementation by following more closely an admonition by Jaime and others: “Listen to the architecture.”

1 Preamble

In 1987 the company I was working for sent me to the first Machine Translation Summit in Japan. After spending a few days in Tokyo I boarded an intercity bus to Hakone, the conference site. During the ride I found myself talking to two strangers, and we discovered that we were all headed to the same destination. They listened intently as I talked about my work, though my naiveté at the time makes me cringe now: I was talking to Jaime Carbonell and Masaru Tomita and had no idea who they were. They were very gracious and patient as they listened to me, then in turn explained about their work at CMU in the Center for Machine Translation. They even suggested that, if I ever decided to go to graduate school, I should consider applying to their institution. Their suggestion took root: three years later I was enrolled at CMU and working on the CMT’s Kant project.

During Jaime’s AI core course I became aware of an intriguing new emerging field—cognitive modeling—and wanted to do dissertation work in that area. In a meeting with Jaime I floated some ideas and his response was another piece of life-changing advice: “You need to go architecture shopping.” After memorable (to me) interviews with Marcel Just, John Anderson and Allen Newell, I decided to work with the Soar cognitive modeling system (Laird, 2012). Two decades later I’m still engaged in that framework.

2 The Problem

In my own teaching career I have been trying to fold the notoriously difficult Soar system into the learning environment in a linguistics department at a largely undergraduate-oriented university. Fortunately an occasional exceptional student arrives who is capable of learning Soar; one such student was William Taysom. I had managed to integrate the Link Grammar (LG) parser (another CMU product, (Sleator and Temperley, 1993)) with Soar, which provided a framework to reason about sentence parses. Taysom, a Soar novice at the time, volunteered to write Soar code to perform semantic interpretation of the parse, creating first-order predicate logic output from the parse. I lent him my copy of a large tome of relevant research (Kamp and Reyle, 1993) and within a few weeks he had implemented much of that work in Soar. The result, which we called LG-Soar, provided a parser/interpreter that built an intermediate structure from the LG parse and then generated the semantic predicates as output for the most basic structures. It also performed some limited pronoun reference and anaphor resolution, as well as resolving some types of ellipsis.

The system has been used as a front end to further processing for such applications as grading the English essays of second-language learners (Lonsdale and Strong-Krause, 2003), patient record matching with clinical trial solicitations (Lonsdale et al., 2008), family history text extraction (Lindes et al., 2012), and a natural language interface to a robotic platform (Mohan et al., 2012). Figure 1 shows some sample parsed sentences from a typical family history text (Savage, 1860).

With some minor modifications to the parser
George Smith was at Dartmouth in 1685.

Richard Smith, Boston 1657, had wife Sarah, probably widow of John Strange.

James Smith, Windsor 1643, was a poor, thievish servant.

John Smith, Plymouth, was killed by a cartwheel, June 1661.

Output from the LG Parser is then processed by the Soar component to generate basic first-order predicates that represent (much of) the semantic content of the sentence. The core LG-Soar reasoner consists of several hundred Soar productions, essentially declarative if/then rules, that build intermediate data structures from information contained in the LG parse. The basic pulse of the system is a sequence of operators that perform the various actions available to the system based on its current state and incoming parser input.

For the first sentence of Figure 1 LG-Soar generates:

George Smith was at Dartmouth in 1685.

to account for domain-specific patterns, the result is quite satisfactory. “G” links identify proper nouns, “S” links connect subjects with their verbs, “O” links connect verbs to direct objects, and “M” links connect nouns with various types of postmodifiers (prepositional phrases, appositives, etc.). Note that the robustness of the parser allows it to deduce that the word “thievish” is an adjective (connected to the head noun with an “A” link), even though that word does not appear in the system’s English word lexicon.

Output from the LG Parser is then processed by the Soar component to generate basic first-order predicates that represent (much of) the semantic content of the sentence. The core LG-Soar reasoner consists of several hundred Soar productions, essentially declarative if/then rules, that build intermediate data structures from information contained in the LG parse. The basic pulse of the system is a sequence of operators that perform the various actions available to the system based on its current state and incoming parser input.

For the first sentence of Figure 1 LG-Soar generates:

George Smith was at Dartmouth in 1685.

Note that the temporal adjunct (“in 1685”) is not accounted for in the output. To be sure, this output only specifies a portion of semantic content: the predicate content; this suffices for a downstream application that processes the extracted predicates.

The semantic component, while serviceable, did not handle all possible constructions identified by the LG Parser. Furthermore, it became less scrutabale and more daunting to maintain and enhance over time. As LG-Soar was applied to further applications and we tried scaling it up, this difficulty became problematic. In addition, advances in the basic Soar architecture necessitated changes to the entire LG-Soar codebase. Finally, the system was fairly brittle: in the presence of parsed constructions that were not accounted for in the reasoning engine, the system was prone to ungraceful and unpredictable performance failures—precisely negating downstream a core advantage of the LG parser: its robustness.

Reflecting on this issue, I recalled the mantra that I often heard from Jaime, from my dissertation advisor Jill Fain Lehman and from other Soar researchers: “We should listen to the architecture...” (Newell, 1990, p. 331). It was apparent that we were trying to shoehorn Link Grammar output into a complex but increasingly inscrutable system that needed overhauling. Staying closer to the architecture would provide a more workable solution.
3 The Solution

With these observations in mind, I have recently re-implemented LG-Soar. I deliberately associated the processing more directly with the LG output since each parse link specifies information that can be better employed by the Soar reasoner. The solution was to spawn and execute an operator for each link identified in the parse.

Accordingly, the system parses—in an approximately left-to-right fashion, though this is not strictly required—each link from the parse. Some link-based operators are—at least for now—vacuous, since they contribute little to the semantics. On the other hand, semantically rich links, like those that associate the main verb with its subject or object, spawn the creation of entities, properties, and events in memory. Variables are created for the relevant predicates and their objects, and limited types of resolution are performed by adding layers of interpretation to relations already established. Finally the various entities, events, modifiers, and other objects are generated from the resultant data structures in memory. The system is also more robust, simply ignoring unrecognized links.

Figure 2 shows a simplified and annotated operator trace for interpreting the last parse from Figure 1. Predicates are output in steps 17-20. Note that, in neo-Davidsonian style, a variable has been introduced for the event so that the temporal modifier (“in June 1661”) can be associated with it.

The re-implementation is capable of handling most of the semantic phenomena that the original LG-Soar system processed. However, the new processing is more sophisticated for multi-word expression processing, recognition of temporal adjuncts and appositives, and handling of passive constructions. Output of results is also more elegant. The new system is also more perspicuous, consisting of only some 75 productions. It is thus more flexible and extensible.

4 Future work

Several possible directions are possible for follow-on work. Obviously, this prototype needs to be scaled up to handle all link constructions that the parser provides. Currently about a quarter of all link types have been implemented in the new system, but prospects for addressing the other ones are improved with the system’s new-found agility.

We plan to do a full test of the system’s performance, perhaps in a head-to-head bakeoff with LG-Soar. This would probably take the form of a traditional precision/recall/F-score evaluation on gold-standard, human annotated text in the family history domain (for which we have copious annotations).

Soar is capable of learning as it executes tasks, so it should be possible to implement a system capable of recognitional processing when encountering similar problems.

To be clear, the system as described to this point makes no assumptions about human cognition, so its value as a modeling system for the syntax-semantics interface remains to be demonstrated. It is likely, though, that processes that require conscious deliberation, such as ambiguity resolution, discourse interpretation, or pragmatics processing, could be modeled directly. More investigation in these areas is warranted.

So far we have developed Link Grammar parsers for several other languages including French, Persian, Arabic, and the (Native American) Coast Salish language Lushootseed. We plan to integrate these into the new architecture to investigate claims made about cognition-based processing in these languages.

References


Figure 2: Operator trace for processing a sentence of LG parser output.

Papers from Robots Learning Interactively from Human Teachers (AAAI Fall Symposium Series).

