

Issue Statement: Mike Maxwell (CASL/ University of Maryland)

There is an open ended list of software-related needs that linguists have, ranging from the simple to the sophisticated. I will address two needs, at opposite ends of this spectrum.

At the low end, what is the minimal documentation that might be useful to linguists a century from now? Consider the following scenario: The last speaker of a previously undocumented language is an 80 year old woman. Her granddaughter wants to document the language, but there are no linguists available to do elicitation or transcription. How might the granddaughter document the language, in a way that might some day facilitate reconstruction of some portion of the language? One method might be to make audio or video recordings of her grandmother telling stories, and then write down translations of those stories into a major language. Would this work? Could any useful information be recovered in the future, or is it a useless form of documentation? Assuming that it might work for at least some purposes, at what level of granularity would the translation need to be done (story level, paragraph level, sentence level...)? How could alignment of the translations be done at this level of granularity? How much data is needed for various kinds of reconstructions (phonology, morphology, syntax, lexicon...)?

Near the opposite end of the spectrum of sophistication, consider how language grammars are documented today. Unlike lexicons and texts, grammars are generally described in plain prose, on paper. This is the way it has been done for thousands of years. But prose descriptions leave much to be desired. Being written in a natural language, they are ambiguous; being tested at best by hand (and that usually by the original author), they have gaps. Computational grammars, on the other hand, are unambiguous, and while they may have gaps in coverage, those tend to be smaller because the grammar can be tested on corpus data. On the other hand, computational grammars are largely uninterpretable without the original software program that was used to run them, a program that will almost certainly be unavailable a century hence.

But what if a tested computational grammar could be expressed in a formalism, and then embedded into a prose grammar such that the formal grammar disambiguated the prose grammar, and the prose grammar explicated the formal grammar? We would have the best of both worlds, with the strengths of each way of expressing the grammatical generalizations making up for the weaknesses of the other.

As it happens, a technology developed for making computer software code more comprehensible—Literate Programming—does just that: it embeds the code into a prose description, while still allowing the code to be extracted and run. While this idea is not widely used among computer programmers, it finds a very natural application in grammar documentation. And implementations of Literate Programming frameworks are available today in a widely used XML format (DocBook). Remaining to be done is the creation of an unambiguous, iconic formalism for morphology (and perhaps other aspects of linguistic description), and building test/ example cases.