Grammar Debugging

Michael Maxwell

University of Maryland, College Park MD 20742 USA
mmaxwell@umd.edu

September 2015

"If debugging is the process of removing software bugs, then programming must be the process of putting them in." – Edsger Dijkstra

Why?

Question: How do you know *whether* your grammatical description is correct?

Why?

Question: How do you know *whether* your grammatical description is correct?

Answer: By testing it!
(see my “A System for Archivable Grammar Documentation”, SFCM 2013)

Why?

Question: How do you figure out *why* your grammatical description is *in*correct?

Why?

Question: How do you figure out *why* your grammatical description is *in*correct?

Answer: By debugging it!

Previous work

We have developed an XML-based representation for morphology and phonology. Current coverage:

- Affixes (prefixes, suffixes…affixes-as-processes, including reduplication)
- Inflectional affix templates (encode order of prefixes/ suffixes; processes can override)
- Morphosyntactic features (including nested features; extended exponence)
- Inflection classes (= conjugation classes and declension classes)
- Phonemes/ graphemes, boundary markers
- Classes of phonemes/ graphemes
- Regular expressions over phonemes, classes…
- Phonological rules (including epenthesis, deletion, metathesis)
- Rule exception features (positive and negative)
- Suppletive wordforms ("irregular forms")
- Dialectal and spelling variation, alternative scripts

Previous work (continued)

- We write the formal grammar in XML; a converter program (written in Python) reads the XML and creates the code for the target parsing engine (currently Stuttgart FST).
- We "Compile" that SFST code, together with lexical entries (usually derived from electronic dictionaries), and the output is a parser/ generator.
- XML grammar schema is designed to abstract away from a particular parsing engine's programming language.
- XML grammars can therefore outlive the parsing engine.
- This has been used to build morphological parsers for a variety of languages (Bangla, Pashto, Somali, Swahili, Persian...)

Previous work (continued)

What's still missing or in progress:

- Rule strata, compounding, derivational affixes, "stem names"
- Debugging (this talk!)
- Visual editor displaying objects in a linguistic format (no XML tags!)
- Typesetting in linguistic style
- Generic dictionary import methods

Some motivations for an XML-based declarative linguistic description language

- Ease of use by linguists
- Software independence
- Longevity
- Linguistic basis…
- …But theory agnosticism (“Basic Linguistic Theory”, R.M.W. Dixon)
- Allow alternative analyses
- Reproducible research

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.” – Martin Fowler

A debugger

- Why doesn't my grammar + parsing engine parse word X?
- Desired output: a trace of the derivation, showing where the parse goes wrong.
- Naively:

tienes	surface form
tenes	diphthongization
...	(other phonological rules)
$[\text{ten}]_V$-es	suffixation
$[\text{ten}]_V$-3sgPresInd	lexical lookup

Naive view of debugger

...or if the diphthongization rule failed to (un)apply, perhaps:

tienes	surface form
tienes	*diphthongization
...	(other phonological rules)
$[\text{tien}]_V$-es	suffixation
$[{*}\text{tien}]_V$-3sgPresInd	lexical lookup

("*tien" represents non-existent lexeme)

Problem 1

In reality, the search space is branching, and often large:

tienes					surface form
tienes			tenes		diphthongization
...			...		(other rules)
tienes	[tien]$_V$-es	[tien]$_N$-es	[ten]$_V$-es	[ten]$_N$-es	suffixation
*tienes	[*tien]$_V$-3sg	[*tien]$_N$-Pl	[ten]$_V$-3sg	[*ten]$_N$-Pl	lexical lookup

–which complicates debugging, since the user sees uninteresting paths in the search space.

(N.B. For reasons of space, affix glosses simplified, adjectival parses omitted)

Problem 2

There is no search in the sense of de-constructing a derivation:

- Modern parsing engines (finite state transducers, or FSTs) “compile” a parser by attaching affixes to words in the lexicon(s), applying phonological rules, and finally removing any auxiliary characters (like boundary markers).
- The result is a network consisting of pairs of matched paths, with one path in each pair representing the lexical form, the other the surface form.
- Lookup consists of finding a path among the surface form paths that matches the word to be parsed, and returning the corresponding lexical path.
- As a result, the compiled network does not contain any intermediate stages in the derivations.

Exception: The Hermit Crab parser (a non-finite state parsing engine) in principle allows tracing of intermediate stages of non-parsing words.

...and More Problems!

Problem 3: As a further result of the way FSTs work, it's impossible to display what even the trivial (two stage) derivation of a word is, because *there is no path* corresponding to a non-parsing word.

Problem 4: FSTs can be very slow to compile: up to 20 or 30 minutes, depending on size of lexicons and other factors.

Problem 5: Using XML interposes an extra level of abstraction between what the linguist *writes* and what the computer *does*.

How then to debug?

Problems:

- Problem 1: In parsing, there may be more than one search path to explore.
- Problem 2: Compilation throws away intermediate stages.
- Problem 3: If the parser doesn't parse a surface word, the surface form doesn't even exist in the parser, so its derivation could't be followed (even if there were intermediate stages).
- Problem 4: Life is short.
- Problem 5: XML $\neq$ SFST (or XFST or...)

Solution:

- Problem 3: Start with the underlying form and see what you get.
- Problem 2: Compile the surface form from that underlying form step-by-step, and display the output of each step.
- Problem 1: Since we start with the underlying form, there is no search (branches occur only with free variants or optional phonological rules).
- Problem 4: Compile only the target lexeme.
- Problem 5: This turns out to be an advantage!

What can cause failure to parse a word?

- Failure to extract a lexeme from a dictionary.
- Lexeme is spelled incorrectly (typo, spelling variation, missing diacritics, similar letters that differ in Unicode, upper-lower case issues...).
- Surface form is spelled incorrectly (same issues).
- Incompatibility of affix(es) with lexeme (wrong part of speech?).
- Incompatibility of affixes with each other (incompatible features in multiple exponence).
- Affixes in wrong order.
- Expected allomorph cannot appear in phonological environment.
- A phonological rule unexpectedly fails to apply (rule written wrong, rule ordering problem).
- A phonological rule applies when unexpected (same reasons).

Remainder of talk shows how we've achieved (most of) this.

How the debugger works

- Assumptions:
  - There is a word which won't parse correctly.
  - The linguist (thinks he) knows how it should parse.
- Two ways to run the debugger:
  - Command line
  - GUI (talk will concentrate on this)
- In either case, linguist provides a description of how the word should parse:
  - a lexeme
  - its part of speech
  - an inflectional template
  - a list of (inflectional) affixes, or a set of morphosyntactic features
- The debugger either says "You can't do that because...", or it generates a derivation. (Presumably the derivation results in a surface form different from the expected one.)

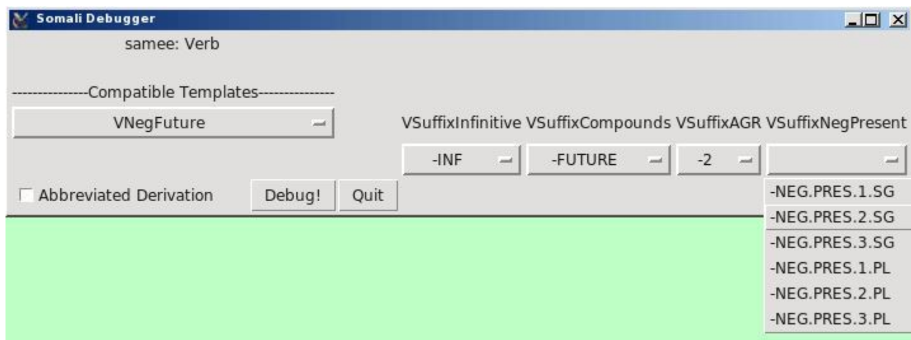
Step 1: Lexeme selection

Failure at this point indicates one of two possible errors:

- Failure to extract a lexeme from a dictionary.
- User spelled lexeme incorrectly.

Since FST network will contain only this lexeme, compilation is fast.

Step 2a: Choose affixes

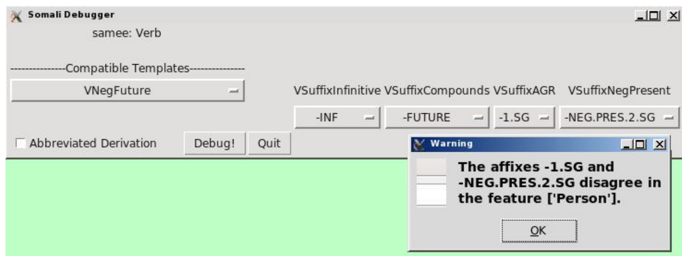


Failure at this point indicates one of three possible errors:

- Incompatibility of affix(es) with lexeme (indicated by absence of desired affix in list of possible affixes)
- Incompatibility of affixes with each other (incompatible features, indicated by error message–see next slide)
- Affixes in wrong order (indicated visually by order of affix slots in templates).

Step 2a: Incompatible affixes

Incompatibility of affixes with each other due to incompatible features, indicated by error message:



Step 2b: Choose morphosyntactic features

Somali Debugger

samee: Verb

---------------Morphosyntactic Features---------------

Mood:	None selected	imperative	conditional	optative	potential	infinitive	declarative
Person:	None selected	first	second	third			
Number:	None selected	singular	plural				
Reduce:	None selected	No value	reduced				
Valence:	None selected	No value	autobenefactive				
Tense:	None selected	past	present				
Aspect:	None selected	No value	progressive				
Gender:	None selected	masculine	feminine	unknown			
Negation:	None selected	No value	negative				
Exclusivity:	None selected	inclusive	exclusive				
Case:	None selected	absolutive	subject				

---------------Compatible Templates---------------

VNegFuture

Abbreviated Derivation   Debug!   Quit

Failure at this point indicates one of three possible errors:

- Incompatibility of morphosyntactic feature(s) with lexeme/ POS (indicated by absence of desired feature in list of possible features)
- Incompatibility of features with each other (indicated by error message).

Step 3: Follow derivation

SFST “compiler” is called automatically to generate each intermediate step (= output of each phonological rule) plus final output, and display this in a browser.

Failure to generate target surface form indicates one of two possible errors:

- A phonological rule unexpectedly fails to apply (rule written wrong, rule ordering problem).
- A phonological rule applies when unexpected (same reasons).

Because each step of the derivation is visible, the linguist can see where the derivation went wrong.

Step 3: Follow derivation

Mozilla Firefox

file:///home/m...ebugTrace.xml | file:///home/m...ebugTrace.xml

file:///home/mmaxwell/Data/Somali/DebugRe | Google

Derivation of stem="samee" with glosses="<-INF><-FUTURE><-2><-NEG.PRES.2.SG>"

Underlying Forms	samee+n+ doon+t+o			samee+i+ doon+t+o	
StemShiftLANtoLM	"			"	
NGemination	"			"	
LongAAtoAY	"			"	
LongEEtoE	"		samey+n+ doon+t+o	"	
GlideInsertion	"	sameey+n+ doon+t+o	"	"	sameey+i+ doon+t+o
TbecomesS	"	"	"	"	"
BGemination	"	"	"	"	"
WtoOB	"	"	"	"	"
LNtoLL	"	"	"	"	"
RNtoRR	"	"	"	"	"
MidVowelCollapseGeneral	"	"	"	"	"
MidVowelCollapseRFinal	"	"	"	"	"
StemShiftGtoK	"	"	"	"	"
LTtoSH	"	"	"	"	"
DhdeGemination	"	"	"	"	"
AfterWVoicing	"	"	"	"	"
DGemination	"	"	"	"	"
TdeGemination	"	"	"	"	"
IYtoSH	"	"	"	"	"
DictionaryNtoUnderlyingM	"	"	"	"	"
Gemination	"	"	"	"	"
DeGemination	"	"	"	"	"
DeGemination2	"	"	"	"	"
UnderlyingMtoN	"	"	"	"	"
UnderlyingMtoN2	"	"	"	"	"
NVowelDrop	"	"	"	"	"
Surface Forms	sameendoonto	sameeyndoonto	sameyndoonto	sameeidoonto	sameeyidoonto

Implementation

- Parser converter (XML-to-FST) and debugger are both implemented in Python.
- GUI is implemented in Python-Tkinter.
- Currently in Linux; could probably be ported to Windows.
- Remember problem 5?
  Using XML interposes an extra level of abstraction between what the linguist *writes* and what the computer *does*.

Implementation

- Parser converter (XML-to-FST) and debugger are both implemented in Python.
- GUI is implemented in Python-Tkinter.
- Currently in Linux; could probably be ported to Windows.
- Remember problem 5?
  Using XML interposes an extra level of abstraction between what the linguist *writes* and what the computer *does*.
  SFST is not a general-purpose programming language; we could not have written the debugger in SFST alone.
  The Python converter from XML to SFST gives us the programatic control over the compilation!

Planned enhancements to debugger

- Explanation of allomorph choice.
- Diagnosis of an incorrectly spelled lexeme.
- Diagnosis of an incorrectly spelled surface form.
- Better ability to determine why a rule doesn't apply (by iterative simplification of rule's environment or input).
- Better explanation of why rule applies when it shouldn't (by alignment of rule input and environment with input form).
- Port GUI to browser (HTML + Javascript)
- Open source

- Probably not possible: Try all possible phonological rule orderings (N! in number of rules)

With thanks to

- Olivia Waring (now at Microsoft)
- Nikki Adams (Somali and Swahili parsers)
- Erin Smith-Crabb (Somali parser)