Systematic Issue Solving Workflow to Improve Single Library and Inter-library Interaction Issues in the Grammar Matrix

Tongxi(Tom) Liu
University of Washington
Department of Linguistics
Supervisor: Emily M. Bender
June 6, 2022

1 Introduction

The LinGO Grammar Matrix (Bender et al. 2002, Bender et al. 2010) is a meta-grammar engineering framework that using the Head-driven Phrase Structure Grammar (HPSG; Pollard & Sag 1994) and Minimal Recursion Semantics (MRS; Copestake et al. 2005) formalisms. The Grammar Matrix handles language variation and similarity cross-linguistically. Users interact with the Grammar Matrix through an interface of a web-based questionnaire, and filling out the prompts in the questionnaire for each linguistic phenomenon (e.g., Word Order, Number, Person, Gender, etc.) allows the Grammar Matrix customization system to produce an implemented grammar in the HPSG and MRS formalisms. Implementations of linguistic phenomena in the Grammar Matrix rely on the development of libraries, such as adnominal possession (Nielsen 2018), adjectives (Trimble 2014), case (Drellishak 2008, Drellishak 2009), constituent questions (wh-ques) (Zamaraeva 2021), etc. The interactions among these libraries are crucial for the Grammar Matrix to produce accurate and well-formed grammars. Because library creators cannot predict every usage scenario or interaction with other phenomena, there are issues of either single library or inter-library interaction that emerged in the application process. In this paper, I approach the question of **how to follow a systematic workflow to improve single library and inter-library interaction issues in the Grammar Matrix**.

Since there are more than 20 libraries in the Grammar Matrix, it is challenging to test all possible user inputs for accounting for library usage scenarios. However, it is possible to focus on a group of related single library and inter-library interaction issues. By generalizing my workflow of issue-solving processes, it is possible to build a systematic roadmap to guide developers to comprehensive solutions. Moreover, studying issues' similarities can reveal larger underlying computational linguistics processes. Thus, interpreting and solving these issues is not a trivial case-by-case process but requires a comprehensive and generalized approach to explaining the nature of how specific problems occur in the Grammar Matrix.

This paper is structured as follows. I introduce essential background on the Grammar Matrix in section 2. My methodology to collect and evaluate samples is presented in section 3, consisting

of Sample Collection (3.1) and Regression Tests (3.2). In section 4, I describe the issues-solving processes and performance of existing and newly-added regression tests. Then, I discuss a possible generalization of issue-solving workflow and its advantages in section 5. Finally, in section 6, I conclude the work in this paper and reflect on its potential to contribute to a future computational linguistics study.

2 Background: The Grammar Matrix System

This section presents necessary background information on the HPSG (2.1) and a high-level overview of system components on the Grammar Matrix (2.2).

2.1 HPSG

Head-driven Phrase Structure Grammar (HPSG; Pollard & Sag 1994) is a lexicalist, surface-oriented, and sign-based grammatical framework. The following paragraphs in this section explain partial basic concepts of the HPSG for readers to understand the theoretical basis of the Grammar Matrix library issues presented in this paper.

In the HPSG, the grammatical information is reflected as *typed feature structures*. A feature structure is a set of features and values. Each feature is distinct in a feature structure and is paired with its value. For example, we can consider the pronoun *she* in English, which conveys several pieces of information, such as *third* in person, *female* in gender, *nominative* in CASE, *singular* in number, etc.

(1)

$$\left\langle \text{she}, \begin{bmatrix} \textit{pron-lxm} & & & \\ \text{CAT} & \text{HEAD} & \begin{bmatrix} \text{CASE} & \text{nom} & \\ \text{AGR} & \begin{bmatrix} \text{PER} & \text{3rd} \\ \text{NUM} & \text{sg} \\ \text{GEND} & \text{fem} \end{bmatrix} \end{bmatrix} & \\ \text{CONT} & \begin{bmatrix} \text{INDEX} & i & \\ \text{RESTR} & \left\langle \begin{bmatrix} \text{RELN} & \textbf{\_pro\_n\_rel} \\ \text{INST} & i \end{bmatrix} \right\rangle & \end{bmatrix} & \end{bmatrix} \right\rangle$$

The model of typed feature structures allows it to describe linguistic entities comprehensively. Each sign in the feature structure description is treated as having its form (CAT) and meaning (CONT). Example (1) (Sag et al. 2003, p. 509) shows a lexical entry for the pronoun *she*. This lexical item consists of a pair of orthography and a feature structure description. It has type of *pron-lxm*, a pronoun lexeme type. The feature can have either an atomic value, such as [PER (person) *3rd*], or a collection of other features, such as feature ARG (agreement) with PER, NUM (number), and GEND (gender).

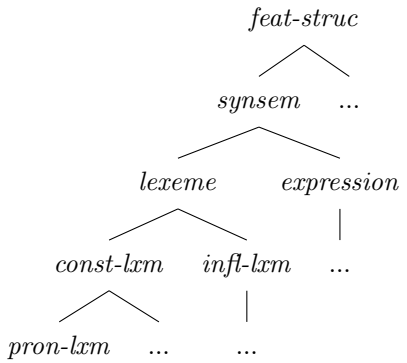
Typed feature structures are arranged in an inheritance hierarchy. Each typed feature structure has specific constraints associated with it, inherited by all subtypes and instances of that type. Example (2) (Sag et al. 2003, p. 498) below shows the constraints of lexeme types *pron-lxm*, which

inherits from *const-lxm*, *lexeme*, *synsem*, *feat-struc* as shown in (3). Thus, example (1) also has constraints in example (2) since *she* inherits constraints from the type *pron-lxm*.

(2) Lexeme type of *pron-lxm*

$$\begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{noun} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{MODE} & / \text{ ref} \end{bmatrix} \\ \text{ARG-ST} & \langle\rangle \end{bmatrix}$$

(3) Partial type hierarchy showing the position of *pron-lxm* in the grammar in Sag et al. 2003, p. 492.



2.2 The Grammar Matrix

As introduced in section 1, the LinGO Grammar Matrix expedites the grammar-engineering process by an interface of a web-based questionnaire. This section will describe a high-level overview of system components on the Grammar Matrix for readers to understand the pivotal background of where and how library issues can occur.

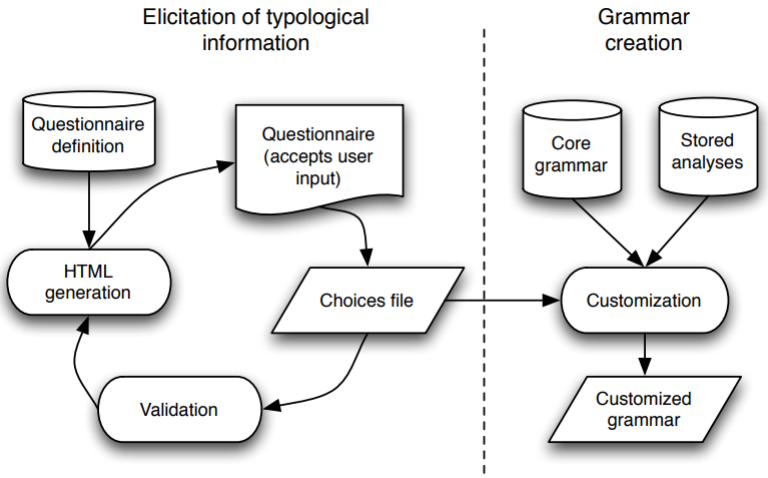


Fig. 1 Schematic system overview (Bender et al. 2010, p. 29)

A user can interact with the Grammar Matrix starting with the HTML questionnaire. As shown in Figure 1 (Bender et al. 2010), the Grammar Matrix reads the user's input from the questionnaire and writes the records of input into a choices file. The choices file contains serialized data with the answers to questions in the questionnaire. Answers to different libraries are grouped into sections. Then, the Grammar Matrix passes the choices file to **validation**, which checks if the answers are consistent and complete to define a well-formed grammar. If not, it will provide feedback of *errors* and/or *warnings* to the questionnaire. The questionnaire will generate a new HTML with feedback to prompt the user with errors in answers or required further input. The user iteratively answers questions and improves answers in order to describe a language. Once the user has a choices file without *errors* after the validation process, the Grammar Matrix customization system can send the choices file to the **customization** script. Finally, the customization script produces a customized grammar from the choices file based on the Matrix core grammar and stored analyses.

When a user interacts with the Grammar Matrix, issues can occur in validation and/or customization processes. The validation reports problems in the choices file (i.e., user's input). A validation issue occurs when there is a problem in the choices file, but the validation script does not report the issue, or when the validation script reports an issue with a legitimate choices file. Customization issues occur in three situations. (1) The customization script crashes in producing a customized grammar from a legitimate choices file. (2) The parser fails to load a customized grammar (i.e., the customization script generates a broken grammar). (3) The customization script produces a grammar prone to spurious ambiguity. When the validation script does not catch the problem in a choices file, the choices file will be sent to the customization script and will likely produce an issue. This issue may be a customization problem, but it might be better addressed in terms of validation.

The implementation of the libraries is examined by Bender et al. 2010 both in the validation and customization processes. Each library has its validation function to check for the validity of answers in choices files. The validation script calls these functions to check for the choices file. The validation script calls these functions to check for the choices file. When the customization script generates customized grammar from a choices file, it sends the choices file to each library when handling

different sections. A library may trigger other libraries in this process. Thus, the causes of issues that occur may relate to inter-library interaction.

3 Methodology

This section describes my methodology to study single library and inter-library interaction issues in the Grammar Matrix. Two subsections are sample collection (3.1) and regression tests (3.2).

3.1 Sample Collection

As introduced in section 2.2, the Grammar Matrix system has two primary sources of issues from validation or customization processes. The current (unsolved) issues of the Grammar Matrix are listed on its GitHub Repository.[1] Most issues are posted by Matrix developers while developing new libraries or working with students in Ling 567, a University of Washington computational linguistics course about grammar engineering. As of this writing, there are 327 open issues and 268 closed issues. There are 187 open issues labeled as Customization and 30 open issues labeled as Validation.

Rather than selecting random issues from Grammar Matrix's GitHub Repository, for this paper, I focus on several problems involving related libraries from customization and validation processes. The purpose is to experiment with how issues, from validation and customization processes and single library and inter-library, behave similarly and differently and generalize the solutions to a workflow, which developers may adopt in future work on the Grammar Matrix. Thus, it will help developers design more comprehensive solutions to account for a wider group of issues, including future cases not reported in Grammar Matrix's GitHub Repository yet.

Since I have more experience in the adnom-poss library, selected issues are preferred to relate to this library. Here is the list of 7 issues:

1. poss-phrase-1 does not have the binary-non-loc-phrase supertype (GitHub issue #632).
2. poss lex rules leave NON-LOCAL features underspecified (GitHub issue #625).
3. adnom-poss library does not fully create disjunctive case types (GitHub issue #621).
4. poss-unary-phrase type does not have a supertype (GitHub issue #598).
5. validation fails to block giving two separate values for the person on a possessive pronoun (GitHub issue #622).
6. validation needs to check for nouny features on verby morphology (GitHub issue #623)
7. validation fails to block for verb types and supertypes both indicating valence (GitHub issue #627).

Issues #1 and #2 are the inter-library interaction issues of the adnom-poss library and wh-ques library. Issue #3 is related to the adnom-poss library and case library. Issue #4 represents the surface problem of a missing supertype of poss-unary-phrase type in the adnom-poss library. The

[1] https://github.com/delph-in/matrix/issues

essence of this issue relates to the inter-library interaction between the adnom-poss and morphotactics library. Issues #5 to #7 reveal insufficiency in the validation process. Based on the description of these validation issues, I notice these failures may occur in a broader scope. Studying them to build generalized solutions will help construct an effective workflow.

3.2 Regression Tests

To evaluate the correctness of the Grammar Matrix system after adapting certain solutions, Pydelphin (Goodman 2019) based Regression Tests (Bender et al. 2007) aid me to verify that improvements do not cause problems in other fragments of system, and to create new tests to account for new features. This section introduces the structure and process flow of regression tests.

A regression test has three parts, a choices file, test suite, and gold-standard parse profile. As introduced in section 2.2, a choices file contains serialized data with the answers to questions in the questionnaire. The test suite has a set of sentences with positive and negative (ungrammatical) items. A regression test passes if items are parsed or ruled out with analyses matching the MRSes stored in the gold standard profile. It is possible that a test report *pass* with stored profiles where some of the grammatical sentences are not parsed, or some of the ungrammatical are. If a regression test fails, its corresponding log file indicates which item is parsed differently in the current and the gold profile.

To test the validity of new features or improvements introduced to the system, designing reasonable choices files and corresponding test suites is essential to create new regression tests. Some issues are reported with a complete or partial choices file on its GitHub Issues page. Some choices files are not real-world languages because they are made up for testing purposes. These choices files assist the Grammar Matrix developers in reproducing the issue. However, most of these choices files can only account for the surface of the problem, which requires developers to investigate deeply into issues and develop more meaningful choices files. For example, although the choices file attached in issue #1 shows a missing binary-non-loc-phrase supertype, its significance is causing more ambiguity in parsing when combined with the rules for in-situ wh questions. Thus, rather than using the original choices file to test if the missing supertype is satisfied, it is crucial to design a choices file that can test if it reduces ambiguity in in-situ-wh sentences.

I applied two design methods for designing test suites in regression tests to verify the validity of compiling or parsing processes. If an issue only has to do with the process of compiling, then focus on its regression test is to check if there are compilation errors in loading the grammar from its choices file. It does not require introducing a list of test sentences, so the method is to only add a simple sentence as a placeholder into test suites and leave comments in the test description. The other design method focuses on how well improvements are performed in parsing sentences. In this case, the method is to extract key features from the choices file and make up positive and negative sentences to ensure the test can cover comprehensive aspects of improvements.

Before improvements, there were 527 regression tests in Matrix's trunk, with 516 passed and 11 skipped. Skipped tests were labeled as *no longer worked tests* or *unfinished works*.

4 Results

This section describes the results of issues solving (4.1) and the results of regression tests after improvements (4.2).

4.1 Results of Issues Solving

4.1.1 Issue #1 and #2

As introduced in section 2.1, the types in typed feature structures in the HPSG formalism adopted by the Grammar Matrix are arranged in an inheritance hierarchy. In issue #1, the possessive phrase type (poss-phrase-1) does not have the binary-non-loc-phrase supertype, while a binary-non-loc-phrase type contains a constraint on the NON-LOCAL feature. Thus, missing this supertype leaves an underspecified NON-LOCAL feature in poss-phrase-1. An underspecified feature indicates the parser will unify regardless of constraints imposed by other grammar entities on the value of this feature. An example of triggering the possessive phrase type and NON-LOCAL feature is an in situ wh question, such as *poss n1 iv?*[2] Therefore, when parsing in situ wh question sentences with the possessive phrases, the NON-LOCAL feature of *poss n1* is underspecified. The parser will generate one legitimate result and one inaccurate result. The legitimate one has the wh-in-situ rule instantiated since the possessive phrase has a non-empty value for QUE inside of NON-LOCAL. The inaccurate one treats it as an intransitive sentence with no NON-LOCAL feature in the possessive phrase.

Issue #2 shows that lexical rules produced by the adnom-poss library inherit from lex-rule rather than same-non-local-rule, leaving an underspecified NON-LOCAL feature. Similar to issue #1, the consequence is also ambiguity in generated grammar in the interaction of the adnom-poss library and wh-ques library.

Both issues #1 and #2 are inter-library interaction problems involving the adnom-poss and wh-ques libraries. Both indicate an underspecified NON-LOCAL feature in the adnom-poss library. Since it is not the wh-ques library making the adnom-poss library miss NON-LOCAL features, issues #1 and #2 are not problems in the wh-ques library. However, the wh-ques library reveals the insufficiency of the adnom-poss library, that possessive phrase type and lexical rule in adnom-poss library fail to cover NON-LOCAL features. Although the solutions to issues #1 and #2 are simply adding constraints on NON-LOCAL features by adding supertypes to the types, the crucial inspiration from these issues is to check for other types that may need NON-LOCAL features, and to build comprehensive regression tests to check for unambiguous results.

4.1.2 Issue #3

Issue #3 presents an interaction issue with the adnom-poss library and the case library. The system fails to create a type of disjunctive case when a possessive pronoun is defined. The resulting grammar fails to load, stopping in particular at the point in loading possessive phrase types with the disjunctive case. Upon investigation, I found that the functionality of disjunctive case creation only works without interaction with the adnom-poss library. Therefore, the cause of this issue has to do with the disjunctive case creation process in the case library. The case library generates necessary

[2]This is a hypothetical sentence with possessive pronoun *poss*, a noun *n1*, and an intransitive verb *iv*. *poss n1* forms a constituent of a possessive phrase, and this phrase and verb *iv* forms another constituent.

case types and saves case types before initializing the adnom-poss library. The adnom-poss library builds disjunctive case types too late to save them into the grammar. Therefore, the solution is to reconstruct the existing functionality of case types initialization. Rather than moving the case types initialization section from the case library to the adnom-poss library, my solution is to extract case types initialization as a function and invoke the function after case and adnom-poss libraries initialization.

4.1.3 Issue #4

The adnom-poss library generates a poss-unary-phrase type without any supertypes in issue #4, which leads to a grammar that does not compile. My approach is to test how the system generates a legitimate poss-unrary-phrase type. It turns out that a non-empty morphology choices file makes correct supertype generation for poss-unary-phrase type. In other words, if a user has answers in the morphology section (and poss-unrary-phrase relevant answers), the resulting grammar produces poss-unary-phrase with supertype. It is noticeable that the procedure of poss-unary-phrase supertype generation involves interaction with the morphology library. The lack of morphological rules invoking the features enabled by the adnom-poss choices file makes the supertype creation functionality not work. Since the supertype of a phrase type should be pre-determined no matter the existence of another library, the supertype generation flow in poss-unary-phrase is not accurate. Thus, the solution needs to be focused on reconstructing the generation process correctly. Therefore, my improvement is to remove poss-unrary-phrase supertype generation from the morphology library and reallocate it to the adnom-poss library.

4.1.4 Issue #5 to #7

Issues #5 to #7 are single library issues involved in the validation process in the Grammar Matrix. Each of these issues has a property that applies beyond the specific library they were noted for (in the GitHub issue tracker). Issue #5 shows a possible input of two different values for the person on a possessive pronoun. Its broad applicability is that users can input duplicated features for other libraries. Thus, the solution is to check for all repeated features in all types on the Grammar Matrix.

Similar properties are shown in issues #6 and #7. Issue #6 reveals that the validation fails to provide errors if features appropriate only for nouns are on the verbal morphology. The broad applicability is inaccurate features on specific morphology or lexicon. The solution is to create case-by-case validation for each part of speech. Issue #7 indicates that the validation does not block the case where verb types and supertypes indicate valence. This issue inspires me to consider the phenomenon of other types and their supertypes showing the same features. For example, if a noun has a case feature with value X, the subtype of this noun has a case feature with value Y. If case value Y does not inherit from X, the hierarchy structure is incorrect, thus causing an issue. Moreover, it is difficult for the validation script to check for noun's case hierarchy since some case types (e.g., disjunctive case) are only generated in the customization process. Thus, for issue #7, the solution I implemented is to have the validation script indicate an error if a verb inherits another verb and has valence and indicate a warning to the user of other duplicated features (maybe different values) with inheritance hierarchy.aybe different values) with inheritance hierarchy.

4.2 Regression Tests

This section reports the quantitative regression results. There are two groups of data to compare: (1) performance of the system before any improvements (baseline) and (2) performance of the system after improvements. Regression Tests mainly cover the improvements for the customization process, which are issues #1 to #4. There are 4 new tests introduced into the system to evaluate solutions, as shown below.

1. (Issue #1) Verify that there is only one reading in the interaction of in situ wh questions sentences with possessive pronoun produced by adnominal possession library.
2. (Issue #2) Verify that there is only one reading in the interaction of in situ wh questions sentences with lexical rules produced by the adnom-poss library.
3. (Issue #3) Verify that the Grammar Matrix can compile grammar from choices files and parse sentences without ambiguity with the disjunctive case in the adnom-poss library.
4. (Issue #4) Verify that the Grammar Matrix can generate legitimate poss-unary-phrase type without morphology definition.

		Before Improvements	After Improvements
Newly Created Tests	# passed	0	4
	# total	4	4
Previously Created Test	# passed	516	516
	# total	527	527
All Tests	# passed	516	520
	# total	531	531
	Passed / Total	97.18%	97.93%

Chart 1 Results of regression tests

The results are shown in Chart 1 above. None of the new tests pass before improvements, and all 4 tests pass after improvements without breaking any existing regression tests. Finally, improvements lead to 97.93% in all passed tests over the total number of tests, with 11 skipped tests, while the baseline is 97.18%.

5 Discussion

In this section, I discuss a possible generalized workflow to assist the Grammar Matrix contributors in quickly identifying the source of problems and designing comprehensive solutions in the Grammar Matrix. It first describes the advantages of using a generalized workflow compared to a conventional-issue evaluation strategy, then previews the features and properties of a helpful issue-solving workflow. Finally, it reflects on the 7 issues solved in this paper and proposes the skeleton of the issue-solving workflow for the Grammar Matrix.

A systemic issue solving workflow provides a roadmap for researchers and developers to bring forward comprehensive solutions in the Grammar Matrix. A more conventional issue evaluation strategy (i.e., case-by-case method) focuses on the problem's environment of occurrence, behaviors

of occurrence, and outcomes to the whole system. This strategy analyzes issues based on their in-scope description, which can be challenging for researchers or developers when the information in the original bug report is sparse or even. Moreover, case-by-case analysis fails to build up solutions to cover more phenomena for issues that can reflect in a broader scope. Therefore, I propose a systemic workflow to assist Grammar Matrix developers in finding more comprehensive solutions.

Before working on the developments in the Grammar Matrix, I was a student who only interacted with the customization system. When solving issues in the Grammar Matrix system, it was challenging for me to extract information about the issue to build up reasonable solutions. Thanks to my supervisor, Emily M. Bender, who guided me with leading questions to discover the nature of the problems and generalize them into suitable solutions. Thus, I think a helpful issue-solving workflow should guide developers to abundant information to develop solutions. Solving problems comprehensively requires developers to ask and answer questions about the underlying reasons. In this way, developers can extract more information while answering more questions. However, this skill is based on how familiar and experienced a developer is with the system. If the developer knows to ask questions but cannot organize the information into a comprehensive solution, the path to improvement is still hindered. Therefore, the issue-solving workflow should provide a roadmap with questions for developers to extract information based on the behaviors of issues and generate more information based on current information. Until the developer reaches answer *Issue Fixed*, the questions and answers will keep iterating, gradually leading to better iterations of solutions.

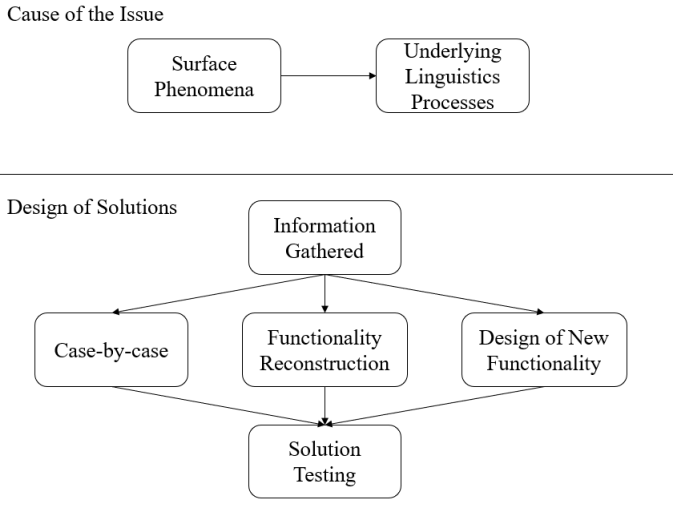


Fig. 2 A brief issue-solving workflow in the Grammar Matrix

A brief issue-solving workflow is presented in Figure 2. When identifying the source of problems, as in section 4.1, I found each issue's bug report is described as encountered. The person putting the issue into the issue tracker may not have done much work towards trying to generalize. Thus, bug reports are not comprehensive enough to trace the cause. The path requires developers to investigate both **surface phenomena** and **underlying linguistics processes**. For example, in issues #1 and #2, the surface phenomena are missing supertypes with NON-LOCAL features.

From a linguistics perspective, the significance of missing features is the outcome of ambiguity in in-situ wh questions. Discovering the source of problems helps developers reflect on issues on a broader scope and design suitable solutions.

After enough information about the problem's cause is gathered, the next path is to come up with reasonable and comprehensive solutions. It is possible to classify the solutions into a few groups, which may bring insights for developers to follow. In 7 issues in this paper, three solution methods help improve the Grammar Matrix without breaking other parts of the system. Methods are **case-by-case**, **functionality reconstruction**, and **design of new functionality**. Case-by-case solutions account for issues with particular occurrence environments. For instance, the occurrence environment on issues #1 and #2 is only in two types in the adnom-poss library when interacting with in-situ wh questions, which does not require developers to either reconstruct this functionality or bring a new function to add the feature. In this case, solving issues #1 and #2 only requires specific improvements to the system, which is the case-by-case method.[3] Another method to build solutions is to reconstruct existing functionalities. The reconstructions can involve changing the order of logic (e.g., re-ordering of case initialization functionality in issue #3), reallocating functionality (e.g., moving supertype generation from morphology library to adnom-poss library), or refactoring one or more functions. Moreover, when issues reveal the system's insufficiency, it is possible to design new functionalities to account for it. In issues related to validation, the creation of new functions elevates surface phenomena of issues into a wider scope, such as creating the function that checks duplicated features for all types. Finally, once developers design a solution, it is necessary to test it to check if the solution reaches the goal and it does not break other parts of the system.

6 Conclusion

In this paper, I have presented the work of fixing several issues with the Grammar Matrix customization system and then proposed a generalized workflow based on the processes I used. The workflow generalizes the commonalities and differences among 7 fixed issues, including single library issues, inter-library interaction issues, validation issues, and customization issues. The solutions are tested both manually and by regression tests.

Since the issue-solving workflow in this paper is based on the generalization of 7 issues and is not evaluated by more issues on the Grammar Matrix yet, the workflow is most likely not a complete guide to assist developers. However, it could be a clue to help developers on how to extract more information in solving problems with a complicated system. Significantly, it can potentially accelerate the workflow for new developers working on issues in the Grammar Matrix or similar system and reduce the potential to re-solve problems.

References

Bender, E. M., Drellishak, S., Fokkens, A., Poulson, L., & Saleem, S. (2010). Grammar customization. *Research on Language and Computation*, *8*(1), 23–72.

[3]However, it is still possible to generalize these issues further. As suggested by my supervisor, (we can) see if any other libraries are adding lexical entries or rules with underconstrained NON-LOCAL values.

Bender, E. M., Flickinger, D., & Oepen, S. (2002). The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In J. Carroll, N. Oostdijk, & R. Sutcliffe (Eds.), *Proceedings of the workshop on grammar engineering and evaluation at the 19th international conference on computational linguistics* (pp. 8–14). Taipei, Taiwan: ACLCLP.

Bender, E. M., Poulson, L., Drellishak, S., & Evans, C. (2007, June). Validation and regression testing for a cross-linguistic grammar resource. In *ACL 2007 workshop on deep linguistic processing* (pp. 136–143). Prague, Czech Republic: Association for Computational Linguistics. Retrieved from `Mhttps://aclanthology.org/W07-1218`

Copestake, A., Flickinger, D., Pollard, C., & Sag, I. A. (2005). Minimal recursion semantics: An introduction. *Research on language and computation*, *3*(2), 281–332.

Drellishak, S. (2008). Complex case phenomena in the Grammar Matrix. In *The proceedings of the 15th international conference on head-driven phrase structure grammar* (pp. 67–86). CSLI Publications, Stanford.

Drellishak, S. (2009). *Widespread but not universal: Improving the typological coverage of the Grammar Matrix* (Unpublished doctoral dissertation).

Goodman, M. W. (2019, October). A python library for deep linguistic resources. In *2019 pacific neighborhood consortium annual conference and joint meetings (pnc).* Singapore.

Nielsen, E. K. (2018). *Modeling adnominal possession in the Lingo Grammar Matrix* (Unpublished master's thesis).

Pollard, C., & Sag, I. A. (1994). *Head-driven phrase structure grammar.* University of Chicago Press.

Sag, I. A., Wasow, T., & Bender, E. M. (2003). *Syntactic theory: A formal introduction* (Vol. 92). CSLI.

Trimble, T. J. (2014). *Adjectives in the Lingo Grammar Matrix* (Unpublished master's thesis).

Zamaraeva, O. (2021). *Assembling syntax: Modeling constituent questions in a grammar engineering framework* (Unpublished doctoral dissertation).